TU WIEN Informatics

Power Consumption of TLS

How the additional „S“ in HTTPS impacts the power consumption of power-plugged devices

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Software Engineering and Internet Computing

eingereicht von

AmraČaušević, BSc
Matrikelnummer 0649241

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Priv.-Doz. Dipl.-Ing. Dr.techn. Edgar WEIPPL
Mitwirkung: Dipl.-Ing. Dr. Johanna ULLRICH

Wien, 30. März 2021

Amra Čaušević	Edgar WEIPPL

TU WIEN Informatics

Power Consumption of TLS

How the additional „S“ in HTTPS impacts the power consumption of power-plugged devices

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Software Engineering and Internet Computing

by

AmraČaušević, BSc
Registration Number 0649241

to the Faculty of Informatics

at the TU Wien

Advisor: Priv.-Doz. Dipl.-Ing. Dr.techn. Edgar WEIPPL
Assistance: Dipl.-Ing. Dr. Johanna ULLRICH

Vienna, 30th March, 2021

Amra Čaušević	Edgar WEIPPL

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Amra Čaušević, BSc.

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 30. März 2021

Amra Čaušević

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Johanna Ullrich for her valuable feedback and guidance through this thesis. Special thanks goes to Markus Maier and Leonhard Alton. Markus participated in numerous discussions, especially in the process of finding the perfect device. With his support, it was possible to increase the sampling rate of the used device - EmonPi greatly. Leonhard worked on his bachelor thesis, with a focus on the power consumption of SSH. Thank you for the idea-sharing, brainstorming and your input for the test automatization, which helped enormously.

Above all, I want to thank my parents and my sisters for their support and guidance throughout my years of study and through writing this thesis. This accomplishment would not have been possible without them. Thank you.

Kurzfassung

Der Bedarf an Protokollen, welche die Integrität, Verfügbarkeit und Vertraulichkeit von Daten gewährleisten wächst stetig. Jede Kommunikation die über reguläres HTTP zwischen einem Client und einem Server gesendet wird, sollte Daten verschlüsselt übertragen. Wenn dieser Kommunikationskanal durch Verschlüsselung gesichert ist, wird HTTP zu HTTPS (HTTP mit TLS - Transport Layer Security). TLS stellt das am häufigsten verwendete Protokoll zum Schutz von Daten während der Übertragung im Internet dar und ist zum State-of-the-Art Protokoll geworden.

Seit Beginn der 2000-er steigt die Anzahl an Forschungsarbeiten, welche sich mit der Evaluierung des Energieverbrauchs von HTTPS befassen. Insbesondere im Bereich von batteriebetriebenen Endgeräten und im Umfeld von IoT Geräten. Durch die wachsende Bedeutung an Umweltbewusstsein und dem bewussten schonen von Ressourcen, gewinnt die Energieeffizienz immer mehr an Bedeutung. Die Reduktion des Energieverbrauches bringt nicht nur Kosteneinsparungen, sondern wirkt sich positiv auf den Klimawandel aus. Das Feld der Informationssicherheit muss mit diesen wachsenden Herausforderungen mitwachsen und bereits jetzt an einsparenden und effizienteren Energielösungen arbeiten.

Ziel der vorliegenden Arbeit ist es, die Auswirkungen der Verwendung von TLS auf die HTTP-Kommunikation auf Leistung und Stromverbrauch zu ermitteln. Dazu verfolgen wir einen zweistufigen Ansatz. Zuerst erstellen wir in der ersten Phase der Arbeit eine kostengünstige Alternative einer Messumgebung, die auf einem open-source Projekt basiert. Diese Umgebung ermöglicht die Messungen des Stromverbrauchs von TLS. In einem zweiten Schritt betrachten wir, in einer vergleichenden Analyse, die Leistung von verschiedenen Verschlüsselungsalgorithmen (Cipher Suites) in Bezug auf Energieeffizienz und Reaktionszeit an Geräten, welche direkt an das Stromnetz angeschlossen sind. In diesem Zusammenhang stellt die folgende Arbeit eine der ersten Analysen des Energieverbrauches von TLS, an Geräten die direkt an das Stromnetz angeschlossen sind, dar. Indem wir den Stromverbrauch von unterschiedlichen Cipher Suites erforschen, wollen wir erkennen ob und welche Algorithmen sich positiv auf den Energieverbrauch auswirken. In diesem Kontext beobachten wir, ob sich energieschonendere Alternativen aufzeigen, die einen signifikanten Einfluss auf den Energieverbrauch der nächsten Jahre im Bereich IT-Sicherheit nehmen könnten.

Abstract

The increasing relevance of security within the Internet has led to a growing demand to ensure the confidentiality, integrity and availability of data. Nowadays, former HTTP traffic is encrypted and is now referred to as HTTPS traffic. Thereby, TLS is used to protect data in transit and has become a state-of-the-art protocol.

Research in the field of energy consumption of secure communication channels has become increasingly important in scientific studies since the early 2000s. However, it predominantly focused on Internet-of-Things devices and battery-powered devices, e.g. smartphones, to increase battery life. In recent years, energy efficiency is becoming more important through the growing environmental awareness and climate change. Therefore, the field of information security has to provide energy-efficient solutions now, not only for battery-oriented devices, but also those directly connected with the power grid, e.g. servers or PCs.

The goal of this thesis is to determine the impact on energy consumption encrypting HTTP with TLS, i.e. HTTPS. In order to achieve this, we pursue a two phased approach. In the first phase, we evaluate various measurement environments for this purpose and decided on an open-source project. In the second phase, we conduct a comparative analysis between different cipher suites regarding their energy efficiency and execution time. This research is one of the first to conduct such an analysis on a plug-socket device. Our goal is to determine whether and which cipher suites have a lower energy consumption and might pursue the way for energy savings in the future.

Contents

CHAPTER 1

Introduction

The introduction of the Hyper Text Protocol (HTTP) in 1989 by Tim Berners-Lee has revolutionized the way the modern world communicates. Since then, the Hyper Text Transfer Protocol has become one of the most popular protocols on the Internet [1].

The popularity of HTTP in combination with the need for secure communication, inspired researchers to take a closer look at the HTTP protocol and reflect on the fact that HTTP does not contain any security mechanisms. Malicious programs could use vulnerabilities to spread, while remaining hidden. This was one of the biggest flaws of the HTTP protocol. This led to the development of a more secure version of HTTP - HTTPS which supports encryption [2]. Any type of data that is sent over regular HTTP between a client and a server can be end-to-end encrypted using Transport Layer Security (TLS). In case the communication channel is secured by encryption, HTTP becomes HTTPS (HTTP over TLS) [3]. TLS is the most used protocol for protecting data in transit, and it became a state-of-the-art standard [4]. With this background in mind, the central question that motivates this thesis is how TLS impacts the power consumption of power-plugged devices. By empirically examining the power consumption of TLS as one of the most used security protocols for transit over the Internet, we hope to provide a more complete understanding of the impact on securing communication with encryption.

In this chapter, we will give a short overview of the problem statement, as well as our motivation and goal. Also, a short guide explaining the structure of this thesis is included.

1.1 Motivation

Due to the spread of mobile devices, the need for users to access web-based services at any time has increased. This has put pressure in the field of ensuring integrity, privacy and authenticity over public networks. To securely transfer sensitive data, services depend on secure end-to-end transactions. The goal of the HTTPS protocol is not only to establish

this secure connection for data transfer between client and server, but also to be as efficient as possible, while achieving the required level of security. Programmers should be offered an easy way to develop applications, where secure communication can be established without knowing much of other applications involved. An easy, straightforward way (e.g. a framework) for providing a possibility of adding new public key and encryption methods should be ensured [5].

W3Techs [6] investigates different technologies of website, where they include the top 10 million websites in their statics. In March 2022 over 78.7% of the ten million websites analyzed use HTTPS as their default protocol. The trend is rising - since March 2021, this number increased even by 8,1%.

The expected outcome of this thesis is a methodology analysis of the power consumption of various cipher suites, which are used to provide the cryptographic support of TLS.

1.2 Problem Statement and Research Questions

The significance of energy efficiency is becoming increasingly important. When taking energy efficiency into account, we think about doing tasks while using *less energy* to perform these tasks under the same conditions [7]. The reduction of energy comes with different benefits, e.g. reduced climate change impact, reduced need for energy import, and reduction of the price (lowering costs). Concerning any security solution, the reduction of power does not come for free. It is a trade-off between power consumption and cryptographic complexity.

This is why the research in the field of analyzing the power consumption of secure communication connections started in the late 90s [8]. Ayala et al. [9] stated that of all network activities HTTP is the most energy-consuming. In most real-world scenarios it is still unknown how the energy consumption of different cipher suites of a given security protocol, like TLS, manifests itself. There are various studies about how TLS influences the power consumption of mobile devices or the Internet of Things (IoT) environments and how it affects battery life. There are only a few studies about the influence of TLS on the power consumption of a plug socket device on which this thesis will have its focus.

The following research questions will be answered:

Q1: How to measure power consumption for power-plugged devices?

This research question addresses the construction of the test environment and which setups are suitable for measurements of power consumption for power-plugged devices. We aim to test different setups and build an environment, which can be reconstructed with low-cost devices. In the research phase of the thesis, different approaches will be considered and compared for this purpose, which have not been investigated by literature before. A part of this research question will also be addressing what problems arise while trying to measure the power consumption of TLS on different tools (e.g. oscilloscopes).

Q2: How much power is consumed by HTTPS for security?

In the research part of this thesis, we are interested how the energy consumption relates to the additional „S“ in HTTPS, which indicates the encryption part of the protocol (HTTP over TLS). This consumption of energy is mostly caused by increased resource usage during TLS handshake and encryption. In the first step, an approach on how to measure the different protocols in TLS will be described. Numerous research papers concentrate their research on the consumed energy of battery life or the consumption in IoT devices. Our goal is to show the presence of significant differences regarding the consumption of energy and execution time by empirically examining the behavior of different cipher suites.

Q3: Are there more energy-efficient configurations?

We aim to provide measurement results from different cipher suites within TLS. There are hundreds of different cipher suites that are composed of different combinations of algorithms. The overall concern in this analysis will be the impact of executing TLS determining the consumption on the server and client side with different cipher suites. Hereby, the most popular and robust TLS authentication algorithms (e.g. ECDSA, RSA) will be used.

1.3 Structure of the Work

In chapter 2.1 and chapter 2.2 we will introduce the theoretical concepts, which will provide the background for the following sections. These chapters focus on the most important concepts that are used throughout this work. In the introduction, the different phases of the TLS protocol and their most important characteristics will be explained. This section also provides fundamental electrotechnical concepts for measuring power consumption.

The main contributions of this thesis are provided in chapter 3 and chapter 4. While chapter 3 presents different approaches on how to measure the power consumption on the level of milliseconds, chapter 4 presents the result of our measurements used on the selected testbed. Chapter 5 concludes this thesis.

CHAPTER 2

Background

The following chapter provides definitions of TLS, the description of the protocol flow, as well as the different principles of measurements, followed by a discussion of the correlation of these definitions. Finally, we go into more detail regarding the TLS protocol and problems of measuring power consumption will be discussed.

2.1 TLS Overview

After the introduction of the HTTP protocol in 1989, HTTP has become one of the most popular protocols on the Internet, not only for access to websites, but also for tunneling data of other applications (e.g. Skype, Dropbox) [2]. Since HTTP alone does not ensure the privacy or security of data sent over the Internet, it was extended to support encryption using TLS as a sublayer. Now, it is known as HTTPS - HTTP over TLS, which became the most used communication protocol when it is necessary to ensure security and integrity of data. Therefore, if a website uses HTTPS as their protocol, all sent data will be encrypted by TLS certificates [10].

TLS provides a secure connection during the transfer of sensitive data across the Internet. A TLS connection occurs between a client and a server, where a client can be just a browser or an IoT device and the server e.g. a homepage or a remote machine in the cloud. TLS is based on the Secure Sockets Layer Version 3.0 protocol (SSL 3.0). It is designed to encrypt data and to authenticate and ensure the integrity of it. Already in the early process of developing the HTTP protocol, the developers of Netscape felt a higher need for encryption. For this reason, they developed the Secure Socket Layer - SSL protocol to meet this requirement [11].

TLS was introduced by the Internet Engineering Task Force (IETF) in 1996 and is similar to SSL 3.0. That is why it is also called TLS/SSL protocol. Over time, vulnerabilities in the protocol were discovered, and newer versions of TLS were introduced. In August

2018 the IETF standardized the latest version of TLS, TLS 1.3, to eliminate known flaws at the previous versions. The new version of TLS did not only correct known vulnerabilities, but also improved security features and performance. As of today, the most used standard is TLS 1.2 with TLS 1.3 becoming more applicable [12].

Secure communication should always be used when sensitive data is sent over the public Internet, e.g. online banking, online shopping or transaction of healthcare data. The TLS protocol is layered on top of the transport protocol. As presented in figure 2.1 above the transport protocol, the application protocol presents itself [13]. TLS will send as little information as possible via an unencrypted transport system. Only the most needed information, like the common session key and HMAC key (keyed-hash message authentication code), is exchanged in an unencrypted way.

The TLS protocol consists of two main protocols:

- The handshake protocol
- The record protocol

In our further work, we will distinguish between these two and take a closer look at each of them. The focus of the record protocol lies in the field of privacy and reliability while using different symmetric cryptographic and hash functions (AES, RC4, SHA-1). Whereas the handshake protocol concentrates on the establishment of the authentication between server and client. The handshake protocol defines the cryptographic key, which will be used in the process. Also, it will be decided on an encryption algorithm, the version of TLS, as well as the cipher suite which will be used [14].

Consequently, in the handshake protocol, the connection between server and client will be established, whereas the record protocol secures this connection. The TLS Handshake is the core of the TLS protocol, its most consuming part is the creation of the pre-master secret [15].

The handshake protocol establishes the session, which defines a set of cryptographic security parameters. These sessions are exchanged across multiple connections [4]. Therefore, the benefit is in the reduction of the time-consuming negotiation of the security parameters. The following parameters are defined in a session state [14]:

- **Session-Identifier**: a random byte sequence which is generated by the server to identify an active session
- **Peer-Certificate**: X509.v3 certificate of the client (only if available)
- **Compression-Method**: algorithm for compression
- **Cipher-Spec**: defines the encryption algorithm and the one-way hash function for the HMAC

- **Master-Secret**: for deviation of secret keys (48 bytes)
- **Is-Resumable**: a flag indicating whether this session is ready to be used for new connections

X.509 public key certificates are used to authenticate the server to the client and vice versa [13]. If the certificate is provided from a trusted Certificate Authority (CA) it is ensured that a website belongs to the domain name owner. Without HTTPS all activities are exposed to be observed. In the handshake phase, the server and the client are authenticated over public key cryptography. A secure communication channel will be established by generating unique keys for each established session. In the further communication between the server and client, this key will be used for the actual communication. This is done to ensure the integrity of the data [10].

We differ two important phases of the TLS protocol, called layers or protocols [14]:

- In the higher layer, the TLS partial protocols e.g. the ChangeCipherSpec (CCS), alert and handshake protocol are defined
- In the record layer, we find the record layer protocol

In addition to the described handshake and the record protocol, the application data protocol, the CCS protocol and the alert protocol should not be overlooked [11]. The application data protocol sends the data transparently through the layer. This is the main difference in comparison to the layers above. In the handshake protocol, the negotiated parameters used are agreed on. These parameters will be fragmented, compressed, and encrypted to be protected against falsification and then used for the active session (see 2.1). The ChangeCipherSpec consists only of the message shown as one byte with the value 1. It is used for showing differences in the cryptographic algorithms.

The alert protocol used in the TLS/SSL procedure is used to show special states e.g. errors or connection loss and consists of the following two fields [14]:

- **Security level:** can take the value **1** - which stands for warning or the value **2** for fatal. In the value **2** - fatal, the TLS connection will be ended as soon as possible. Other sessions remain, but no new connections can be made.
- **Error code** - describes the error in more detail

The main feature of the TLS handshake consists of the authentication of client and server to each other and the negotiation of a set of cryptographic keys before sending sensitive data. The record layer's main functionality takes care of data fragmentation, their encryption and decryption. Also, TLS messages send and received on the transport layer level are handled [16]. The main focus of this work lies on these two protocols as being the two most major layers in TLS. Therefore, in the following, we take a closer look at these two major protocols.

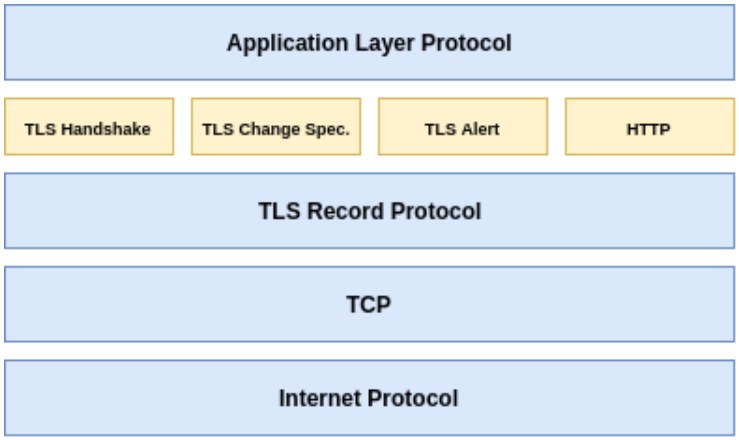


Figure 2.1: TLS Architecture [11]

2.1.1 Handshake Protocol

The TLS handshake protocol starts in the session layer, where the cryptographic parameters as well as the session key is generated. On the presentation layer, the encryption of the data is carried out. This is achieved by agreeing on the cryptographic algorithm and the underlying session key. The server and client can authenticate each other and negotiate encryption, the MAC algorithm and the session keys to encrypt data later on in the TLS record. The negotiation between the server and the client has to be done before any data is sent on the application level [11].

This means that the handshake protocol is used to identify and authenticate the communication partners to each other. Furthermore, it is used to negotiate cryptographic algorithms, keys and parameters used later on in the TLS record layer protocol (described in section 2.1.3).

Suárez-Albela et al. list three basic properties, that the TLS handshake protocol approves [17]:

- All participants are authenticated using public key cryptography e.g., RSA (Rivest-Shamir-Adleman), DSA (Digital Signature Algorithm)
- The safe transfer of data is ensured and data remains inaccessible for eavesdroppers. It is even secure against man-in-the-middle attacks
- The agreement between the server and client is reliable. Thus, attackers are not able to alter any information sent between client and server without being detected

In the handshake phase, both parties have to confirm that they are going to use the same cipher suite, which will be described later on in more detail in section 2.1.4. After agreeing on the cipher suite, both sides use agreed parameters to share the session key

exchange for authentication, the bulk encryption and hashing. TLS is a peer-to-peer connection that is only used temporarily. This connection is associated with a new session which means several TLS connections can be established in parallel [14].

2.1.2 Overview of the protocol flow of TLS

The TLS protocol is designed to establish a secure communication channel between a client and a server to provide confidentiality, authenticity and integrity [4]. As stated, the role of the handshake protocol is to identify authentication and allow negotiation of a cipher suite. This allows data confidentiality and integrity in the next step.

The protocol flow of TLS occurs in two phases [14]:

1. The establishment of the connection, which consists of four steps:
   - **1. Phase:** negotiation of security parameters (Client Hello, Server Hello)
   - **2. Phase:** server authentication and key exchange (Server Certificate, Server Key Exchange, Certification Request, Server Hello Done)
   - **3. Phase:** client authentication and key exchange (Client Certificate, Client Key Exchange, Change Cipher Spec, Certificate Verify)
   - **4. Phase:** termination of the handshake (Finished)
2. The transfer mode ensuring encryption and integrity of transferred data

For our measurements, where we will be analyzing the energy consumption, the detailed understanding of the handshake phase is essential. Therefore, the complete TLS process is provided in figure 2.2 and described in the following [16], [14]:

1. Hello Request
The server might ask the client to start a negotiation, but this type of message is not used to establish the connection. This message has no parameters and the client might ignore it. It replies if it is not in a handshake and sends *Client Hello* only if it is available.

2. Client Hello
With the message type *Client Hello*, the client initiates a TLS connection. It is sent whenever a client first connects to the server. It is represented as plain text and contains the following information for generating the secret keys:

- Used TLS version (TLS 1.2 or TLS 1.3)
- Session ID, which is used to re-open a previous TLS connection and to skip a number of steps of the TLS handshake phase if possible
- Random Client - RC (4-byte timestamp + 28-byte random number)

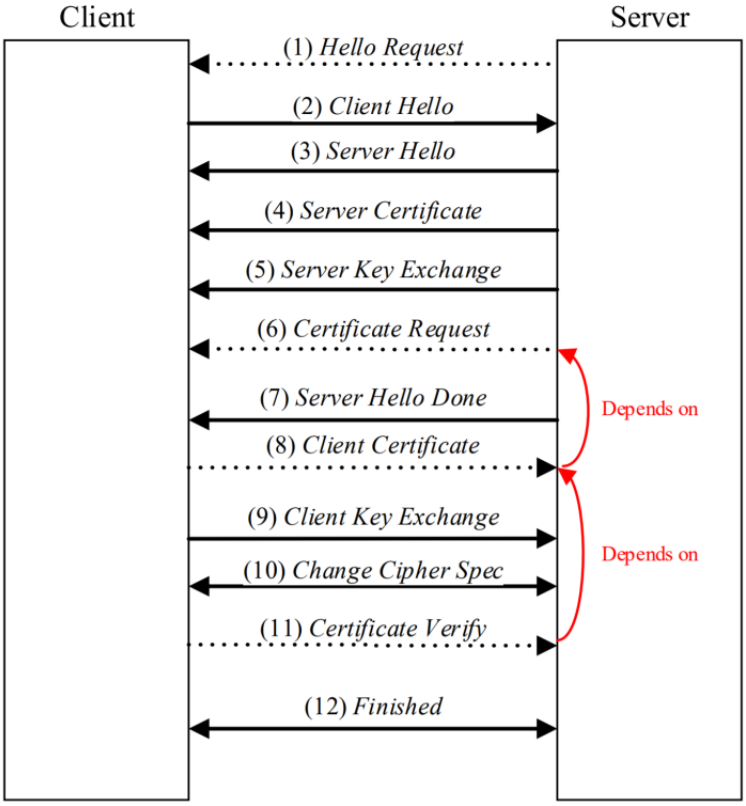


Figure 2.2: Overview of the TLS Handshake [16]

- A list of the preferred cipher suites that the client supports. Only cipher suites that satisfy the needed level of security should be proposed

3. Server Hello
With the message *Server Hello*, the server responds to the *Client Hello* message. It contains the same message parameters as the *Client Hello Message*:

- TLS Version
- Session ID - If the client has proposed a session ID, the server checks in the session cache if this session ID was already used to skip possible steps of the handshake phase
- Random Server - RS (4-byte timestamp + 28-byte random number) and
- The selected cipher suites of the client-side. The server chooses the strongest cipher suites of the proposed list from the client

4. Server Certificate
This is the start of the second phase where the server is authenticated and the key exchanged. The server sends the certificate message instantly after the *Server Hello*. The certificate type must match the selected cipher suite key exchange algorithm.

5. Server Key Exchange
The *Server Key Exchange* message is sent directly after the *Server Certificate* message. With this message, the server defines the cryptographic settings for Diffie-Hellman (DH), Ephemeral Diffie-Hellman (DHE) or RSA key exchange. If the server has a certificate with fixed Diffie-Hellman parameters or with an RSA key, this message will not be sent. Instead, it sends the client a temporary public RSA key in the *Server Key Exchange* message.

6. Certificate Request
If the server requests authentication of the client, this message type will be sent. The message *Certification Request* identifies the certificate types and all supported signature algorithms that the server can verify.

7. Server Hello Done
With this message, the server indicates the end of the *Server Hello* message. It confirms that the client has received all necessary messages of type *Certificate*, *ServerKeyExchange* and *CertificateRequest*. With this, the phase two of the protocol has successfully ended.

8. Client Certificate
In phase 3, the client might be authenticated by the server. This is only done if the server requests this authentication. Therefore, different signing algorithms, e.g. RSA or ECDSA are used. However, the certificate has to be suitable for the agreed key exchange algorithm.

9. Client Key Exchange
The client first checks the validity of the server certificate. If the certificate is accepted, the information is sent to the server, where the agreed cryptographic algorithm from the selected cipher suite, and the 48-byte pre-master secret is used. If the Diffie-Hellman key exchange algorithm is used, the client sends its public key to the server. However, if the server and client have decided on the RSA approach then the client will encrypt the pre-master secret with the public key of the server.

10. Change Cipher Spec
This message is sent by the client as well as by the server to confirm that from now on both parties have agreed on the use of the negotiated approaches. This message is the first message sent with the new security requirements.

11. Certificate Verify
Only in the case that the *Client Certificate* is sent, the *Certificate Verify* message will be sent as well (see figure 2.2). With this message, the client verifies the information and sends proof to the server that it has the secret key for the certificate. This is done

by transmitting to the server all messages the client sent or received from the message *Client Hello* up to this message. The following approach is used:

- For all exchanged messages, a hash value over all bytes so far is calculated, starting from *Client Hello* to *ClientKeyExchange*
- Client signs the calculated hash value with his secret key
- The Server uses the same handshake message and calculates a hash value. He confirms the signature with the public key from the certificate

12. Finished
With this message, both sides indicate that the key exchange and the authentication process has been completed. This message is the first message that is already executed with the new algorithms, keys and security parameters for the record protocol. However, before sending any encrypted, sensitive data, both parties create several keys, using the master secret, e.g. the encryption keys, MAC key.

After this step, the handshake phase is complete, and the data sent to the record layer between client and server is secured by the agreed cryptographic algorithms.

Thus, to ensure confidentiality the encryption algorithm for the cipher suites as well as the encryption keys are obtained in the handshake phase. The sender, who can be the client or the server encrypts the message using the generated encryption. Then the other party uses the same key to decrypt the message. Both server and client are using the same key to decrypt the message [13].

The message integrity is specified in the negotiated cipher suite. For every sent message the sender calculates the MAC using the MAC key. When the server or client receives the message, they compute an own version of the MAC, using the negotiated MAC algorithm. It is possible to generate these keys with two algorithms [13]:

- The Keyed-Hash Message Authentication Code (HMAC): the MAC keys are generated from the shared master secret. For the server-to-client messages, MACs are generated by the server write MAC key, and for the client-to-server messages, the client write MAC key is used.
- Counter with CBC-MAC (CCM) and Galois Counter Mode (GCM) (added in TLS 1.2): for encryption, as well as for integrity the sender uses its write-key. This write-key is used by the receiver who then decrypts the message with the key.

The use of TLS as a secure communication channel generates an overhead of computational work and exchanged data, which causes an increase in energy consumption. This consumed energy is divided in energy consumed by cryptographic and non-cryptographic components.

Operations like all public key operations and symmetric operations, i.e. encryption and decryption of the bulk data and hashing are considered cryptographic operations. All other operations with the focus on sending data and keeping the network interface active are considered non-cryptographic operations [18].

2.1.3 Record Layer Protocol

In the second protocol, the record layer protocol, the shared session key between server and client is used for the encryption of the message that will be sent. During the TLS handshake protocol, shown in section 2.1.1, the algorithms that are going to be used by both parties are negotiated and agreed on. After the successfully established handshake protocol, the protocol goes to the step of transferring the data.

In the record layer protocol, encrypted data is sent from the application layer to the transport layer. It offers two different cyber-security services, that can be used together or individually [14]:

- Client-to-Server encryption between two endpoints
- Ensuring integrity and authenticity of the message

The main tasks of the record protocol are [14]:

- Fragmentation of the plain text (application layer)
- Compression of the resulting fragments
- Calculation of the HMACs over the fragments
- Encryption of the TLS/SSL record

The application data should be fragmented into fragments with maximum size of 2^{14} Byte. These fragments are compressed and hashed with HMAC function by their sequence number (see figure 2.3). These fragments, together with the HMAC are further encrypted as TLS records and handed over to the transport layer [14].

In the record protocol, symmetric key algorithms are used for bulk encryption, so application data can be transported through a secure channel. Also, the different types of messages (e.g. handshake message, alert message) are identified as well. An additional task of the record protocol is to take care of the integrity of each message. In the following we describe the process of delivering application data as shown in figure 2.3 [19]:

- The record protocol receives the application data
- This data is divided into blocks called *fragments*

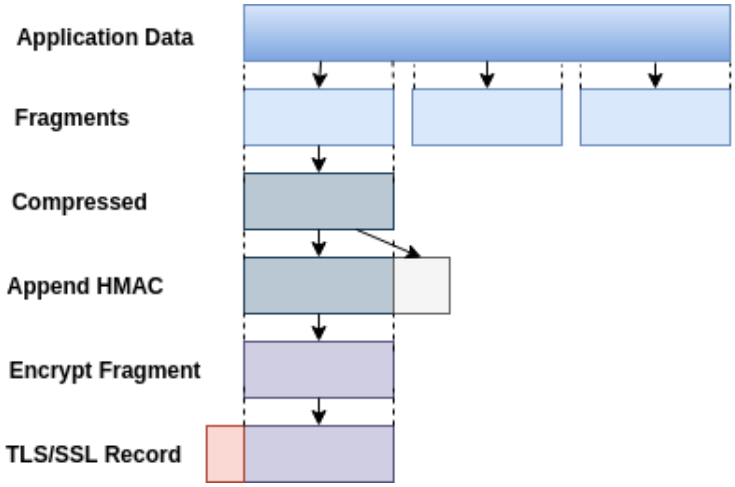


Figure 2.3: Record Layer Process [14]

- Each block is compressed into a fragment and to each record of the message the authentication code - HMAC is added
- Within each fragment the data is encrypted by using the agreed cipher suite between the server and client, done in the handshake phase of the protocol

After these steps are completed, the encrypted data is ready for transmitting to the TCP layer. The following consequences of the record protocol should be taken into consideration [19]:

- The maximum size of one fragment is 2^{14} bytes or 16KB per one record
- Each fragment consists of a header (5 bytes), a MAC (up to 32 bytes for TLS 1.2) and filling in case a block cipher is used
- The complete fragment has to be available to be able to decrypt and verify the fragment

The key properties regarding the security of a connection are connection privacy and connection reliability. Whereas the communication in the connection privacy is encrypted using symmetric cryptography, in the connection reliability the focus lies on the message transfer. It takes into account the message integrity and authenticity (using a keyed Message Authentication Code - MAC), which is achieved with using secure hash functions (e.g. SHA-1) for MAC [17].

One disadvantage of the TLS version 1.2 is that the data of the application protocol is encrypted and transmitted with HMAC protection, but the complete record layer header is unprotected and readable. In addition, also the type of the handshake message can be read out in the handshake protocol [14].

2.1.4 Cipher Suites

A cipher suite is defined as a set of different algorithms that are used to secure a network connection. As described in section 2.1.1 the client sends a list of all cipher suites it will accept in the handshake message. The server then chooses one of the most secure of the offered for use. If we take into consideration the way people surf on the Internet and take a closer look at the connections between client and server it is clear, that the browser connects to the same servers more often.

In the process of certificate validation, the server offers a certificate to the client during the session, which is digitally signed by a certificate authority (CA). They represent the trusted validation of an entity to the client. CA's are third party organizations, which are responsible for issuing digital certificates and managing their validity. They are responsible to confirm the identity of the website server. Thus, the CAs ensure that the identity of the website is valid by digitally signing it with its authority root certificates. All web browsers have a root store, where the trusted root certificates are saved. Thus, a web browser trusts all certificates that are verified using the root certificate from this store. CAs have control over a variety of root certificates (authority certificates) [20]. So, the digital signature has to be verified by the certificate authority, to confirm that this certificate is truly issued by the CA. This algorithm for the validation of the certificate is the same that is used for digital signatures (for example: RSA, ECDSA).

The main responsibilities of the domain certificates are [14]:

- Mapping of the public key to a domain
- By issuing the certification, the authority guarantees not only the correctness of this mapping, but also that the organization truly exist
- It is impossible to change the domain certificate without the private key of the certification authority

With TLS certificates attributes of domains are checked. Such domain certificates consist of the following attributes [14]:

- Name of the organization, which is authenticated through this domain certificate
- Public key of the domain
- Name of the issuing certification body
- Period of validity

We differ between asymmetric algorithms, which are mainly used e.g. RSA, DH and symmetric algorithms. Asymmetric algorithms use two different keys: a public key for encryption and a private key for decryption. Elliptic curve based algorithms provide

the same security level as RSA algorithms, but are able to perform this with a smaller key size. That's interesting because according to Suárez-Albela et al. [17] the energy consumption of asymmetric algorithms is dependent on the key size. On the other hand, in the symmetric algorithm, where only one key is used the energy cost is not considerably affected by the key size.

The cipher suite of the TLS protocol defines which cryptographic algorithm will be used for a session. The cipher suites include the following four cryptographic algorithm types that can be used [21]:

- **Key Exchange Algorithm:** As the name suggests, these algorithms define how the keys are generated and exchanged between the client and the server. The most common key exchange algorithms are RSA, ECDH (Elliptic Curve Diffie-Hellman), ECDHE (Elliptic Curve Diffie-Hellman Ephemeral) and DHE.
- **Authentication Algorithm:** The authentication algorithm is important to verify that the client communicates with the correct server. This is guaranteed by exchanging certificates e.g. Digital Signature Algorithm (DSA), RSA or Elliptic Curve Digital Signature Algorithm (ECDSA). They use different mathematical operations as the basis for their algorithm.
- **Bulk Cipher Algorithm:** Bulk cipher algorithms are used to encrypt the data between client and server. The bulk cipher algorithms are symmetric algorithms, which indicates that they use the same key for encryption and decryption. We distinguish between block-based algorithms e.g. Advanced Encryption Standard (AES), Data Encryption Standard (DES), 3DES and stream-based algorithms like Rivest's Cipher RC4.
- **MAC Algorithm:** MAC algorithms are used to ensure the integrity of the data, so the receiving part of the communication knows that the data has not been changed. The most common algorithms are Message Digest 5 (MD5) and Secure Hash Algorithm (SHA).

Hence, for each connection the client and server agree on the use of these four cryptographic algorithms, seen as an algorithm selection. Cipher suites in TLS are defined as [13]:

```
TLS_KeyExchangeAlg_WITH_EncryptionAlg_MessageAuthenticationAlg
```

Example TLS 1.2: `TLS_RSA_WITH_AES256_CBC_SHA`:

- Key establishment with parameters signed using RSA
- AES-256 provides the confidentiality

- Message authentication is achieved using HMAC SHA

In the newer version of TLS, TLS 1.3, the cipher suites are specified as `TLS_AEAD_HASH`, where AEAD annotates the authenticated encryption with associated data, which is responsible for data's integrity, confidentiality and authentication. The cipher suites of TLS 1.2. can only be agreed on between other TLS 1.2. cipher suites and cannot be mixed up with cipher suites of newer or older TLS versions [13].

Example TLS 1.3 [13]: `TLS_AES128_GCM_SHA256`

- For confidentiality and authentication AES-128 in Galois Counter Moder (GCM) is used
- SSHA256 is used as the function for the master secret

Security is established by combining specific algorithms and their key length. Thus, for example, 128-bits of security is achieved with 128-bit AES keys, but also with 3072 bit RSA keys or with 256 bit Elliptic Curve keys. These different algorithms with different bit key lengths will have the same level of security. Due to attacks on different algorithms, the key lengths have increased over time to strengthen security. Currently, it is recommended to use at least 128 bits for security [12].

In 2016 the Internet Assigned Numbers Authority (IANA) has identified over 300 cipher suites suitable for TLS. The national security agency BSI in Germany (Bundesamt für Sicherheit in der Informationstechnik) suggests the use of only 16 of these cipher suites [22]. Cipher suites in TLS 1.3 differ from cipher suites of the earlier versions. TLS 1.2 currently has 37 cipher suites. In TLS 1.3 there are only five different cipher suites recommended. Holz et al. [12] took a closer look at the different cipher suites and according to their research in April.2019:

- 79,2% of the connections are 128-bit AES in GCM mode established
- 14,4% use 256-bit AES in GCM mode
- 6,4% use ChaCha20+Poly1305

Coarfa et al. [20] showed that the RSA connection is the most expensive operation in TLS. TLS session resumption enables the involved parties to resume the same negotiated data between multiple connections. The engineers of TLS were aware that RSA as such, is expensive. For this reason, they added a session cache, allowing to reuse the result of an earlier RSA connection. With this, they tried to improve web server performance. It enables the implementation with session identifiers and session tickets [20].

The flexibility to use different authentication algorithms (RSA, DSS), key Exchange algorithms (RSA, EDH), encryption algorithms (RC4, 3-DES, AES) and MAC algorithms (MD5, SHA) ensured the security of the TLS. This phase of agreeing on common cryptographic routines strengthens the interactivity and resilience of the protocol for further use. The possibility of adding newer, stronger algorithms at any time and if necessary to reduce dependencies on any other algorithm, marked as vulnerable, makes TLS the current most flexible de facto standard for secure connections. The key factor of the security strength of TLS are strong cipher suites used by the protocol. So, it is crucial, that they are implemented correctly and held up-to-date regarding their vulnerability status [5].

In the next chapter, we describe the most important characteristics for understanding the analysis of power consumption discussed in chapter 4.

2.2 Fundamentals of Power Measurement

When we measure the power consumption of electrical devices such as mobile devices, Raspberry Pi's or Arduino's, two common physical characteristics are obtained: power and energy. They represent different things, but a transformation between these values is possible. Especially later, in this thesis, we will take a closer look at low-cost power measurement devices. In this section, we describe the most important quantities.

2.2.1 Power

Power is defined as the rate of doing work or the rate of using energy. The SI unit (International System of Units) for power is Watt, one Joule per second: $W = 1\ J/s$. Electric power is equal to the product of the current I and voltage V [23]:

$$Power = Current \cdot Voltage$$

Voltage is the size of the electrical potential between two points in a circuit and is given in Volts. Voltage can be measured as peak-to-peak, which stands for measurement from maximum to minimum of a signal. Whereas the voltage represents the difference between two points, the current stays for the flow of electrical charges between these points of an electric field and is given in Ampere [24].

Energy efficiency has proven to be an important design requirement in recent years. Many research papers address the issue on how to reduce the power consumption of different devices. In alternating current (AC - reverses the current flow in a set frequency), the instantaneous value of the voltage or current depends on time. The public electricity supply provides power to households and industries with a periodic alternating current and a chronological sequence [24].

$$U(t) = U_0 \cdot sin(\omega t + \varphi_U) \qquad I(t) = I_0 \cdot sin(\omega t + \varphi_I) \tag{2.1}$$

U_0, I_0 are the amplitudes (maxima) of voltage and current and ω is the angular frequency.

This representation is based on the idea to turn the trigonometric function sinus into a function of time, by using its argument as one in time evenly changing angles $\varphi(t) = \omega t$. φ_U, φ_I represent the phase-change coefficient, which is the value of the phase at time $t = 0$ [24].

Digital electricity meters are the simplest instruments for measuring voltage and current. In regular time intervals, the voltage and current are sampled and used to measure the power and energy. Measured samples are multiplied, averaged and displayed for the user. In the following chapters, we will compare different electricity meters, while trying to measure the consumption of energy in milliseconds on a Raspberry Pi. The error rate of this process should not be neglected, due to the precision issues of measuring within an analog circuit and the sampling process itself. In the literature, we differ between asynchronous sampling and synchronous sampling. In the asynchronous sampling, the error rate is higher, resulting in a longer measuring time for the error rate to be appropriately low and thereby negligible for the user. It is more complex to realize the synchronous sampling method, but in the end, the error rate sinks to a shorter measuring interval [25].

2.2.2 Energy

Under energy, we understand the capacity of doing work which is represented by the SI unit of Joule. Energy is specified as the total sum of power over a time period. Thus, an operation that uses e.g. 5 Watts of power over 30 seconds consumes over 150 Joules of energy. In everyday electricity measurements, the most used unit is the kilowatt-hour which is defined as $1kWh = 3.6 \cdot 10^6 J$ (3600 kJ or 3.6 MJ) [26].

Even though energy is proportional to power, it is not correct to assume that less power also has to indicate the energy has to drop as well. Especially in energy optimization research, it was shown that one task can have a higher consumption of power than another - even with the same characteristics, but can use much less energy (e.g. because the runtime is shorter) [7].

With the need to measure the total electricity consumption of devices, it has been established that this is achieved by using energy measurements, where the relation between power and energy is given as [24]:

$$E = P \cdot t$$

Thus, the energy consumption E depends on the power and the execution time t (for which it was used). So, if a low power device is operating for a longer time, it will use more energy than a high-power device running in a shorter time interval.

The focus of this thesis lies in the analysis of energy consumption using TLS over HTTP on a power-plugged device. Therefore, we will send specified resources from the client

to the server and evaluate the TLS consumption from the server-side as well as the client-side.

2.3 Related Work

The first paper that investigates the cost of TLS was presented by Apostolopoulos et al. in the year 1999, where they concentrate more on the performance regarding latency and throughput. Potlapally et al. [27] gave the first comprehensive analysis of the energy consumption of SSL. In this paper, they show the impact of security processing on the battery life of devices and analyzed the impact of different cipher suites as well as authentication algorithms. Their study concentrate on the battery life of low-power devices.

In 2011 Miranda et al. [18] studied the power consumption of different Secure Sockets Layer (SSL) 3.0 and TLS 1.0 implementations on mobile devices with the focus on different implementations of TLS. The evaluation concentrated also on the differences while using WLAN or 3G on a phone, where the authors evaluated that 3G consumes twice as much, in some cases four times as much energy as the requests called over WLAN.

Through the high acceptance of HTTPS and the increasing importance of this protocol in 2014, Nayloar et al. [28] analyzed the latency and also indirect costs regarding higher consumption of the network services. The most important finding was concerning caching, where the loss of caching could increase the energy consumption up to 30% and therefore the costs for the users.

Inmaculada et al. [9] pointed out that mobile devices are the most popular computer devices for use. In his research, he states that the HTTP-based traffic is the most power-consuming. That is why Inmaculada et al. measures and compares different HTTP-based communication models and analyzes three main communication methods for pushing data from a web server to a client browser. The goal was to give an overview for developers on how to reduce the power consumption on mobile devices.

Manuel Suárez-Albela et al. reviewed in their work [17] the power consumption and throughput of IoT gateways when used with Rivest-Shamir-Adleman (RSA) and Elliptic Curve Cryptography (ECC) signing algorithms. Their research concentrated on the possibility to use ECC signing algorithms instead of RSA for securing IoT gateways communication and they came to propose the use of ECC ciphers, which outperformed RSA in terms of both power consumption. Further, there are numerous research studies with a focus on the TLS protocol in IoT devices. Gerez et al. [29] analyzed the power consumption of TLS on an IoT device for a small subset of cryptographic functions and concentrated their research on the consumption of each step of the TLS workflow. In this work, they concentrated on the energy consumption of RSA and ECDSA, and came to the analogous conclusion as in [17], that the use of ECDSA certificate verification in IoT environments outperforms RSA. In the field of Internet of Things (IoT) the study [15]

investigates the power consumption impact of the Message Queuing Telemetry Transport (MQTT). Also concentrating on two of the most used and known TLS authentication algorithms (RSA and ECDSA), which were executed on an IoT node and compared in terms of energy consumption and average time per transaction. In their research paper [15] they go deeper in the analysis of the TLS authentication algorithms - RSA and ECDSA and compared these on an IoT node with different clock frequencies and different security levels (e.g. for ECDSA curve they evaluated the impact of secp224r1, secp256r1 and secp384r1).

As described, all of these research papers focus on the power consumption of IoT devices and wearables. However, none of these methodologies and approaches measure the power consumption directly from a plug-socket device. We want to show if and how security processing regarding TLS has an impact on power. For this, we first need to analyze and understand the inner workings of TLS and its potential energy consumption (see chapter 2.1). We aim to measure directly from the power-point while comparing our findings with the research done in other areas.

CHAPTER 3

Measurement Setup

Power and energy consumption have become an important consideration in today's research. The necessity of this research area arises from the growing need for low power devices in daily life. As stated, the goal of energy efficiency is to reduce the used energy, while performing tasks under the same conditions. However, to be able to optimize the power consumption of devices, in a first step information about energy consumption has to be gathered. That is why numerous research papers analyzed the energy consumption in different occurrences as well as the consumption in security protocols. Further on, we are not interested in the total power usage of a device just in the consumption of a specific security protocol being used on a low power device. Even though this step is essential, there are still many difficulties in achieving this goal. In the following chapters, we will summarize our attempts in finding the measurement setup.

In order to achieve a suitable measurement setup, we reviewed important specifications that have to be fulfilled for a measuring device to be taken into consideration for our practical part of the research. These characteristics of the devices are usually provided by the manufacturers. The most important characteristics of a device for our purposes are:

- **Sampling rate** - expressed as samples per second (S/s) or by some manufacturers in Hertz (Hz), refers to the frequency with which a device samples the signal. The higher the sampling rate and the resolution, the more details of the signal are captured.

- **Resolution** - Under resolution, we understand the ability of a measuring device to detect and show the range in which a change can be detected. Therefore, with resolution changes within the measurement are detected. These changes are expressed as a number of bits. It is in relation to the voltage measurements (number of bits of resolution : voltage measurements).

- **Costs** - We aim to test different setups and build an environment, which can be easily reconstructed with low-cost devices. For this reason, we could not actively test the PMU, but all other devices were tested and described later in this chapter.
- **Sample Period** - A signal that repeats itself has a frequency, which is measured in Hertz (Hz) and represents the number of repetitions of the signal in one second, also called cycles per second. The sampling rate represents the average number of samples per second. Each repetitive signal consists of a period, which represents the time that the signal needs to go through a cycle. Sampling period and frequency are reciprocal, i.e. sampling period = 1 / sampling rate [30]. Since we want to analyze consumption on the level of milliseconds, we must consider the sampling period of the devices. The sampling period must be sufficiently small to fulfill the purpose of use. For several devices we had to write custom software, to get a lower sampling period - this will be discussed in further chapters.
- **Accuracy** - With accuracy, the correctness of the measurement device is annotated. It states the error rate, which stands for the error between the real value and the measured one. This characteristic is combined by two values: on the one hand, the unsureness of reading the values and the one over the readiness of the full scale. Together these characteristics build the total measurement uncertainty annotated in percentage [24].
- **Measuring Voltage** - As the name suggests the measuring Voltage annotates the range, thus the Min/Max Voltage that is possible to measure with the device.

The limitations of our measurement setup lie in the hardware part of the testbed. It was challenging to get the necessary low-budget equipment, with which it is possible to measure small changes of power increase/decrease that occurs on a Raspberry Pi, while having the primary focus on the power consumption of TLS. We compared several available devices at the market and reviewed an oscilloscope (of the brand: HANTEK), two three-phase network analyzers (brand: ENA3) and the open-source project (OpenEnergyMonitor Project with the EmonPi device) and a Phasor Measurement Unit (PMU).

The intended use of some of these devices are the measurement of electrical parameters or for observations of used energy in households. The intended use of the oscilloscope is rather in form of hobby projects, where the HANTEK oscilloscope is used as a logic analyzer. This sort of energy usage operates generally at a much higher voltage level, and draw more current, which makes the usage of these devices more suited for such use. Only the PMU serves for the use of higher functionality for electric power professionals but was not available to us. This is why the process of finding an instrument to measure power consumption data on a low level (e.g. in the field of milliseconds) has turned out to be more challenging than expected. Even with existing, costly measurement equipment difficulties with inaccurate data, low sampling rates or operational issues could arise as problems. We concentrated on devices that are able to fulfill measurements around one measurement every 50 milliseconds and whit which fine-grained measurements of current

and changes on the level of milliamperes (mA) is possible, but still has to fulfill the low-cost characteristic with a price range of ~500 € and that with the highest possible accuracy. In the following, we give an overview of four different measurement devices we tested through our work.

3.1 Measurements with an Oscilloscope

Description: Oscilloscopes reconstruct an electrical signal, but are only able to measure and display voltage. Voltage is the size of the electrical potential between two points in one circuit and can be measured between the maximum and minimum of a signal. This voltage is called peak-to-peak voltage. Oscilloscopes enable the representation of the electrical signal graphically and belong therefore to the category of devices for graphical representation. For this reason, oscilloscopes became an essential tool for the development, production and repair of electronic devices. Physical quantity (e.g. vibrations or temperature) or electrical quantity - like current or power, are converted from a sensor into a graphical representation, which are then displayed as values for voltage [24]. The generated graph of the oscilloscope shows how signals change over time, where:

- The horizontal axis (X-axis) represents the time
- The vertical axis (Y-axis) represents the voltage

Oscilloscopes use an analog-to-digital converter (AD-converter) to convert the measured voltage into digital information. It captures and stores the signal as a sequence of sampling points until enough measuring points are collected to describe a signal. These points describe the signal as represented in figure 3.1, where an example of a common oscilloscope is shown [30].

The signal shows the graphical representation of a wave, whereas the cycle of a wave is its repeating section. In our further work, we are interested, in sinusoidal waves, which describe the basic waveform (showed in figure 3.2). They present harmonic mathematical properties and are in the form of a sinusoidal shape. Other signals displayed on an oscilloscope are the square wave, the sawtooth and triangle waves or step-and pulse shaped signals.

Even though oscilloscopes are primarily used for voltage measurement, it is possible to calculated other parameters according to the formula presented in section 2.2. Ohm's law states that the voltage between two points in a circuit is equal to the product of the current and the resistance. With two of these sizes, it is possible to calculate the third one using the following formula [30]:

$$Voltage\ (V) = Current\ (I)\ \cdot\ Resistence\ (R)$$

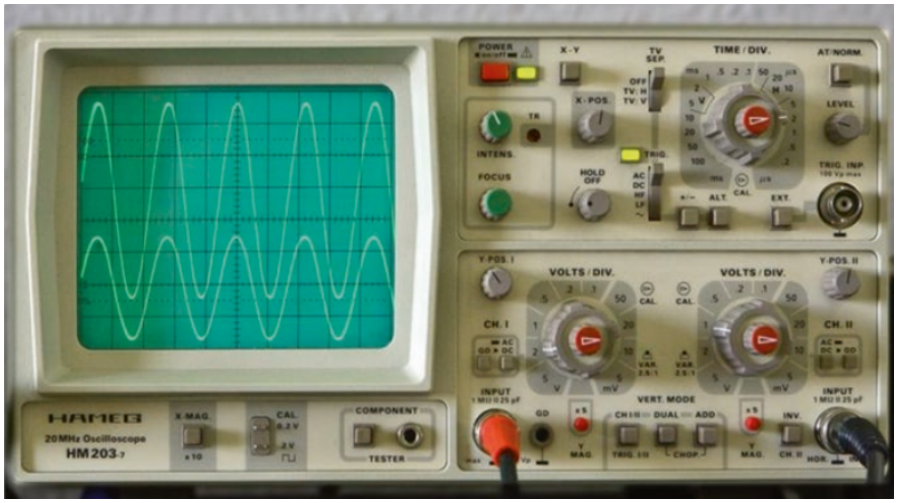


Figure 3.1: Example of an oscilloscope [30]

$$Current\ (I) = \frac{Voltage\ (V)}{Resistence\ (R)} \qquad Resistence\ (R) = \frac{Voltage\ (V)}{Current\ (I)} \tag{3.1}$$

The process of generating samples by an analog-digital converter, where each of these values presents a sample voltage, is called digitizing the analog input voltage. The record length is the number of points that together produce a complete signal. This record length specifies the amount of collected data. Since oscilloscopes are limited in the number of samples they are able to save, the recording time is inversely proportional to the sampling rate of the oscilloscope [30]:

$$RecordingTime = \frac{RecordingLength}{SamplingRate}$$

One further specification of oscilloscopes is the Volts-per-Scale setting - defined as Volts/Div. It shows a scaling factor that specifies the size of the signal changes shown on the screen as in figure 3.1. If the setting for Volts/Div is 5 Volts, each of the vertical divisions represent 5 Volts. Then the entire screen, with e.g. 8 scale divisions can be represented as 40 Volts from bottom to top. If the setting is e.g. 0.5 Volts/Div, the screen shows 4 Volts from bottom to top. The maximum voltage that can be displayed on a screen is the setting Volt/Div multiplied by the number of vertical divisions [24].

In the next section, we reviewed one common low-budget oscilloscope of the brand Hantek. We describe the specifications of the device, our setup and difficulties that have led us for further analysis of other devices.

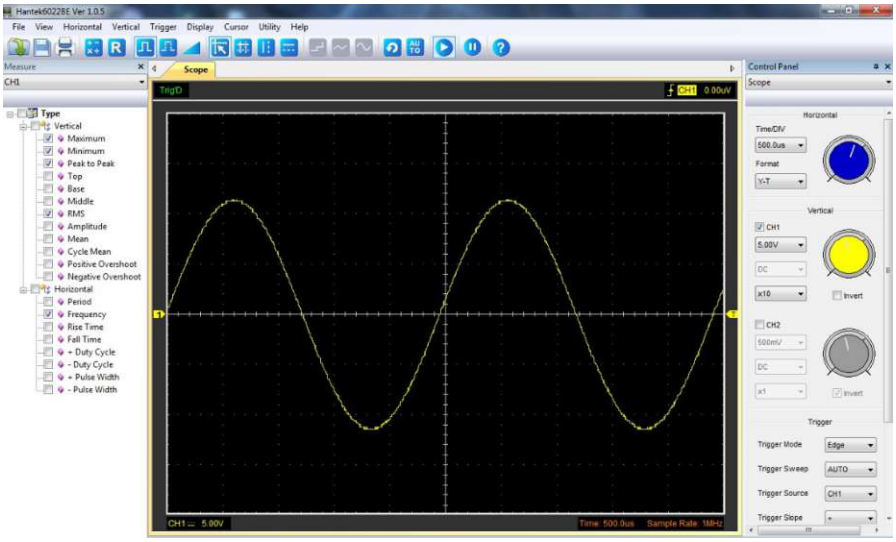


Figure 3.2: Sinus Wave shown on the Hantek Software

3.1.1 Measurements with the Hantek-6022BL Oscilloscope

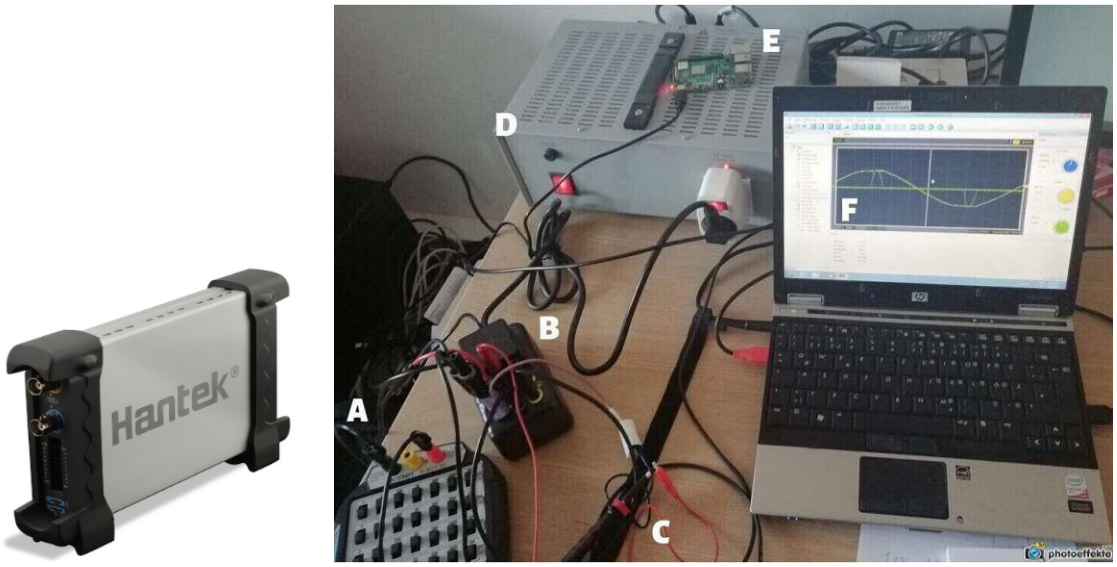
Given that oscilloscopes can be used in the field of research as well as in applications, which include analog circuit tests, we decided to use a low-cost oscilloscope of the brand Hantek (shown in fig. 3.3). The used oscilloscope was a Hantek-6022BL, which belongs to the category of PC-USB oscilloscopes and has the following characteristics (specified in table 3.2):

- 20 MHz Bandwidth
- 48 MS/s real-time sampling (sample rate) and
- 16 Channels Logic Analyzer

The Hantek oscilloscope provides its own software, which we used for our assessments.

As the oscilloscope display only values for voltage measurements, which are presented on channel 1 (CH1) of the device, the measurement of current and resistance are achieved by using an indirect measurement method. Since we planned to use only the oscilloscope - without adding other measuring devices - we had to put in a resistance generator (shown in fig. 3.3 (A)). For this reason, the setup contains the lowest possible resistance - a shunt resistor on which the voltage drop is measured. With a resistance decade, we placed a 10 Ω value in the circuit. The voltage value is directly proportional to the current and is converted into the current value. With this value, we measure our voltage on the resistance decade (U_R).

Measurement Setup: In our indirect measurement of a direct current I, the voltage $U = R \cdot I$ at an ohmic resistor can be calculated for given sample - in our case this



(a) Hantek6022BL (b) Setup

Figure 3.3: Setup of the Hantek6022BL device: A) Resistance generator, B) Voltcraft measurement adapter, C) Clip probes from oscilloscope, D) Isolation transformer, E) Raspberry Pi 4B (the measurement target), F) Software provided by Hantek

would be R = 10 Ω. Since only the voltage is measured, the value of the current must be calculated according to:

$$I = \frac{U}{R}$$

Due to the linear relationship and the available voltage value, we calculate the underlying current. The combination of a passive voltage probe with a current probe provides a good solution for performance measurements. Most passive probes consist of a damping factor, for example 10X, 100X, etc., where the letter X stands for the amplification factor. For example, the 10X probe would improve the accuracy of measurements, but at the same time reduce the amplitude of the signals at the oscilloscope input by a factor of 10. In our testbed, we used an amplification factor of 100X. With this self-calibration routine, we optimized the oscilloscope signal path to obtain better measurement accuracy and so that we do not reach the maximum values of the oscilloscope.

Our transmission ration was set to 1:100 (100X). The value of U_R is added by the resistance decade, which we set with 10 Ω. In a first phase, we calculate the current as:

$$I = \frac{U_R}{R}$$

where R represents the value in the setup and U_R our value on Channel 2 (CH2).

We then calculate the power in the next step as:

$$P = U \cdot I$$

where I is the current computed in our first step and U our voltage on CH2.

Evaluation: In the last step, we analyzed the performance of the device, whether the values are fluctuating or showing constant measurements. If the fluctuation of the measured values is too high, the average over these values should be calculated, e.g. the average over the last 10 values. This is called smoothening the function, which was not needed in our case. Oscilloscopes have numerous advantages: [30]:

- Good presentation of the waveform and their duration - even if signals progress irregularly or are too short (e.g. milliseconds, nanoseconds)
- Good presentation of results even though mixed signals of different shape and frequency are measured
- Waveforms can be stored internally as well as externally
- The results can be copied on an external device for storage and/or later analysis

However, we have noticed some disadvantages, which encouraged us to look for other devices on the market. One disadvantage of the oscilloscope lies in the ability to precisely display the signals with rising signal frequency. As already mentioned, the bandwidth defines the possibility of an oscilloscope to measure a signal, which then provides the frequency range in which the oscilloscope performs exact measurements. So, if a user only needs to analyze stable sinusoidal signals, a record length of ~500 points is enough [30]. However, we were also interested in timing deviations and had to isolate the digital data stream. Therefore, a record length of a million points or more was needed. The storage for this amount of data is difficult, since only a defined number of samples are stored. Aside from the fact, that we were limited by the time period of the measurement results, which was too short for our specific experimental part, we also had difficulties related to the trigger, which has shown to be noisy. Consequently, the representation of a stable wave was not possible. From 10 μ s/Div and faster the scope no longer takes the full 1M (million) data set. The small record size and the fact that the Hantek oscilloscope is a basic USB oscilloscope with only 48 MSA/s motivated us to look for further, more sensitive alternative devices.

3.2 Measurements with Phasor Measurement Unit - PMU

Encountering difficulties with other devices, we decided to search for a much more powerful instrument. For this, we analyzed a phasor measurement unit (PMU), which is

used for the evaluation of values such as e.g. voltage or current. One possible use of a PMU can be the measurement of the frequency in an electrical network, but their main service lies in the ability of fast obtainment of samples.

Description: Regarding the characteristic resolution, one PMU measures the highest and most precise measurements, e.g. 120 measurements per second [31]. This precision is reflected in the high market price of the device.

Dynamic activities can be easily measured in comparison to other traditional devices. The PMU we analyzed was one of the brand Arbiter Systems [32] showed in figure 3.4. This PMU is able to measure 45-65 Hz for specified accuracy. This is important for the precise representation of the waveform. The best measuring results are achieved with sinusoidal waveforms. In section 3.1 we described the definition of waveforms in more detail.

Figure 3.4: PMU - brand Arbiter Systems, Model 1133A Power Sentinel^{TM} [32]

Further, key characteristics of the Arbiter Systems PMU are an accuracy of 0.025 %, a resolution of $1\mu s$ and a phasors rate which is selectable from 1 to 60 measurements/second for 60 Hz [32]. A detailed list of the specification can be found in the overview section of all devices in table 3.2.

PMUs as well as oscilloscopes use an analog-to-digital converter (ADC) for each phase measured. The high sampling accuracy of $\pm 1\mu s$ is realized with the use of the phase oscillator with a Global Positioning System (GPS). It uses a general time source for time synchronization, which is mostly provided by GPS or the precision IEEE time protocol.

Thus, the analog-to-digital converter represents the amplitude and time values into a series of numerical data, where the represented values are called sampled values, which is represented in figure 3.5. These samples are used to reconstruct the values of voltage or current, called phasor quantity [31].

The phase angle and the amplitude are called phasor. As in the first step of PMUs operation, the input waveform and three phases are digitized and presented as numerical

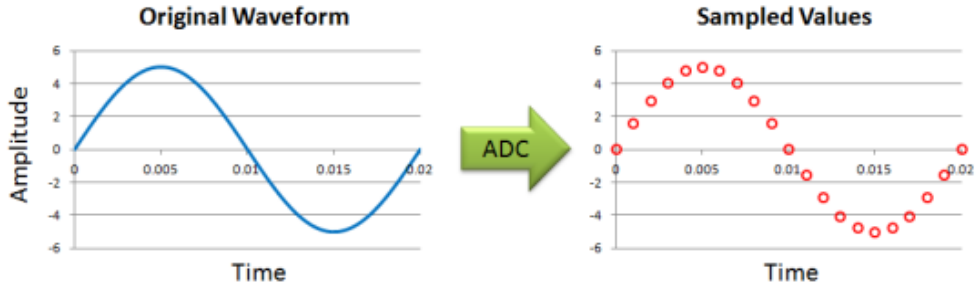


Figure 3.5: Sampling explained with an analog-to-digital converter as a series of numerical sampled values [33]

values, which are operated on the PMU to execute the measurements. Since the waveforms are annotated with a global time base, the results are called synchrophasor [34].

Evaluation: Phasor Measurement Units are by now the most sophisticated time-sensitive devices for real-time monitoring of current and voltage. The high accuracy and the high sampling rate make this device usable for complex measurement environments. The downside is that the PMU lies in a high price range, so it could not be tested actively.

3.3 Measurements with a Network Analyzer

When we take other power measuring instruments for measuring single to three-phase lines into consideration, network analyzers are a valid possibility. Network analyzers assess network parameters of electrical networks and are mostly used at high frequencies. One focus of a network analyzer is to send a signal to a device while measuring the relationship between the input and the output signal. Two categories of network analyzer are available on the market [35]:

- Scalar network analyzer (SNA)
- Vector network analyzer(VNA)

Like the oscilloscope and the PMU, the receivers of a network analyzer include analog-to-digital converters. Most used network analyzers are the vector network analyzer, which measures the amplitude ratios as well as the phase, whereas the SNA is only able to measure the amplitude. They perform time-domain measurements [36].

We looked at the market for low-budget devices with a high degree of precision and accuracy. Also, we searched for a portable, handheld power analyzer used for the testing and inspection of 3-phase electrical circuits.

3.3.1 Measurements with ETIMETER

After encountering difficulties with the oscilloscope, as stated in the previous section, we decided on a different approach and used a three-phase network analyzer of the brand Etimeter - represented in figure 3.6. The main task of this three-phase network analyzer is the measurement of electrical parameters, such as voltage, current, power, power factor (cos φ) and others.

Figure 3.6: ENA3 - Three Phase Network Analyzer [37]

The device under the name ENA3, which we used in our analysis, is a product of the Slovenian company ETI Elektroelement, whose main focus lies in the area of electronic installations.

Description: Currently, two versions of the device are available on the market: ENA3 and ENA3D. We decided on the ENA3 device, because of its sampling rate of 6kHz. In addition to a display, where the device shows the parameters separately for each of the phases, it also provides three independent programmable output relays.

Measurement Setup: The device is programmable directly or as we did over a communication adapter. For this purpose, we used the SCUSB485, which is a serial interface adapter that enabled us to connect the device to a computer via USB. The serial converter allowed the use of the following connectivity between TTL, RS485 and the USB interface, where < and > are representing the directions: **TTL > USB <> RS485**.

The TTL transistor is a basic transistor collector and emitter, sufficient for the use in short distance connections between circuit components, as in our case. They are able to implement logic functions, as well as amplifying function [38]. Whereas RS485 is a standard defined by the Telecommunications Industry Association and Electronic Industries Alliance (TIA/EIA). It defines the properties of drivers and receivers for use in serial communication systems.

It is not possible to directly connect a TTL encoder to a controller that consists of an RS485 interface. For this reason, we used the USB interface on our computer and connect

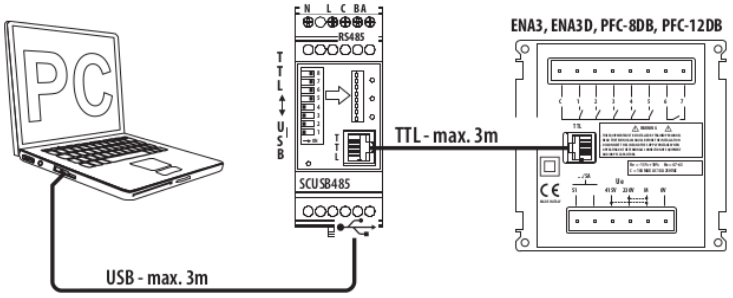


Figure 3.7: Serial Interface Adapter connection with ENA3 [37]

the remote serial device through the SCUSB485 interface as represented in figure 3.7.

In the instruction manual of the device, it is stated that the recording of values happens every 2 seconds. According to these instructions in figure 3.8, the power reading interval can be set in the range of 60 s to 360 s [37].

Basic Set-up Menu:

PAR.	NAME	DESCRIPTION	RANGE	DEFAULT
P.01	t.cur.	Current transformer ratio.	5 ... 50000	5
P.02	r.tu.	Voltage transformer ratio (ex. V_{LINE} / V_{SET} = 500 / 400 = 1.25)	0.40 ... 100	1.00
P.03 (1)	Int.	Power Reading Interval.	60s ... 360s	90s
P.04 (2)	AuEr.	Average time filter Value.	1 ... 20	4

(1) - Parameter P.03 adjusts the time window width for the integration of current and power maximum demand.
(2) - Parameter P.04 allows to modify the stabilising effect that the Average function applies to all readings.

Figure 3.8: ENI3 Basic Setup Menu [37]

To avoid cases where a single irregular peak disturbs or falsifies the measurement, the ENA3 energy meter calculates the average of 16 different values. If this value is above the stored maximum value, it will be overwritten and becomes the new max value.

Evaluation: Although this measurement device has its own usable data reading software and a sampling rate of 6kHz, the output of the measured parameters was not fast enough for our experimental part of this research. The measurements could only be read out every 2 seconds, and we could not use the full sampling rate. Consequently, we decided to develop a customized piece of software, which was able to gather the needed output on the level of milliseconds.

For this, we used the READ/WRITE registers, to get the needed parameters e.g. the values for current, real active power and real apparent power. An overview of all registers is provided on the ETI homepage [37]. In this process, we verified with our customized software, the sampling rate of 6 kHz of the ETI3. We learned that the device reads the

data according to the given sampling rate but was not able to provide the output faster than every 2 seconds over the RS485 interface. For this reason, we decided to look for another device with similar specifications, but with the possibility to read values in the required interval.

The benefits of the ENA3 device are the independently programmable alarm relay and the possibility to present the three-phase measurement of voltage and current, as well as the decent display presentation. The downside was the limitation to read out the values via the serial interface faster than the given interval of 2 seconds. Even though we accessed the sensor directly via the provided ENA3 API, the values could not be read out faster.

For the next step, we looked for a more powerful device, regarding the sampling rate as well as frequency and the sample period.

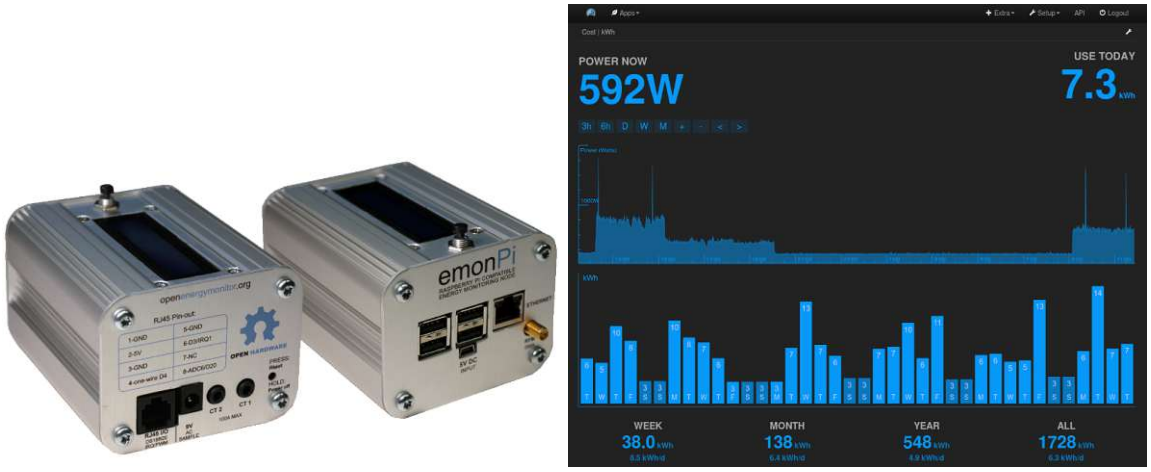
3.3.2 Measurements with EmonPi (OpenEnergyMonitor Project)

The next device we used for our measurements is an open-source project - the OpenEnergyMonitor project [39]. This project started at the beginning of 2000 and was founded out of the growing need to observe used energy in households, as well as to observe changes in the use of energy in energy systems. The entire work of the OpenEnergyMonitor is open source and can be found online, including a learning section and a community section [39].

Description: The underlying device of the OpenEnergyMonitor Project is called the EmonPi Energy Monitor (showed in figure 3.9a), and consists of a Raspberry Pi and a customized Arduino. The Arduino board is based on an ATmega328p 8 bit microcontroller and is responsible for the actual monitoring and measuring. The firmware communicates with the Raspberry Pi, which uses an emonSD software running on the SD card. The Raspberry Pi provides the data, but can also be used for user interaction [39]. Figure 3.10 shows a detailed system overview. The EmonPi is mostly used for basic monitoring of the consumption in households, e.g. for daily or monthly energy usage in kWh. It enables the user not only to monitor the real-time power consumption, but also the consumption of data which lies in the past. The EmonPi is fully customizable and full documentation on what can be adopted as well as how, can be found on the EmonPi homepage under section *Learning*. All this makes the EmonPi a highly practicable device [39].

The available application, that is used by the user is called MyElectric. On the main display, the power use of the consumption along with the daily usage of energy (in kWh) of the past is displayed. Figure 3.9b shows the representation of usage in the MyElectric-App.

Like the ETI3 network analyzer, the EmonPi Energy Monitor consists of an LCD display that makes the setup of the device easier and enables user interaction. In addition to network connectivity over Ethernet, the EmonPi provides also Wi-Fi capability. On the device, two sensors inputs and an AC-AC adapter input are placed. Additionally, temperature sensors may be attached via an RJ45 socket or an optical pulse sensor [39].



(a) EmonPi (b) MyElectric-App

Figure 3.9: EmonPi Device and the Energy View of the MyElectric-App [39]

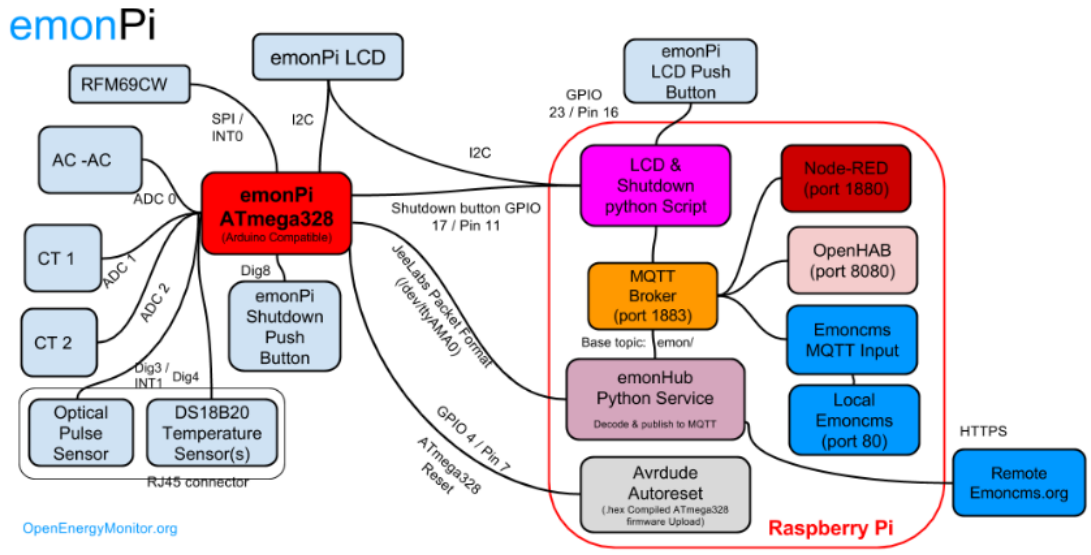


Figure 3.10: System Overview of the EmonPi [39]

In figure 3.10 an overview of the system is given. For our approach, we decided to use the emonHub Python Service interface, and used it to flash the Arduino firmware to concentrate on a smaller set of features. We described this approach in more detail in section 3.3.3.

The accuracy of the EmonPi is close to $> 89\%$. Under figure 3.11, the error rate of the EmonPi device is shown. The main three sources of errors are defined as [39]:

- The transformers, which are the first step in transforming and scaling the measurements. The standard current transformer, the SCT-013 sensor, which we used at the beginning has an error rate of 3%.

- The input circuit is used to convert the current output to a voltage. The manufacturing error lies around 1%.

- The A/D converter - at the last stage of the process, two main sources of errors occur: exactness of the conversion process and exactness of the reference voltage

	emonTx V2	emonTx V3.4 & emonPi	emonTx V3.4.4 CH1-3 / CH4
Current transformer	±3%	±3%	±3%
burden	±1.25%	±1.25%	±0.75% / ±0.35%
analogue reference	±9.1%	±1.1%	±1.1%
TOTAL	+13.78% -12.93%	+5.33% -5.27%	+4.91% -4.79% / +4.5% -4.4%
AC Adapter	±3%	±3%	±3%
voltage divider	±1.83%	±1.83%	±1.83%
analogue reference	±9.1%	±1.1%	±1.1%
TOTAL	+14.43% -13.44%	+6.04% -5.82%	+6.04% -5.82%
Power error	+30.2% -24.6%	+11.7% -10.7%	+11.25% -10.33% / +10.81% -9.97%

Figure 3.11: Error rate of EmonPi compared to other devices available [39]

Furthermore, in the OpenEnergyMonitor project the researchers analyzed the relation between power, voltage and current and observed that the power pull changes around 50/60 times per second. For this reason, they decided to use the average of the instantaneous power, which is called real or active power. In this context, also the reactive power must be considered, which is also called imaginary power, due to its characteristic properties. It is represented as the power going back and forth between the load and the supply, without doing any beneficial work. To calculate the real power, the average power as well as the instantaneous power are required. Consequently, the real power is calculated by multiplying the voltage measurements with the current measurements. The power measurement is summed up over the samples and then divided by the same number of samples as follows [39]:

Code section for EmonPi's calculation for real power

```
1  for (n=0; n<number_of_samples; n++)
2  {
3    inst_power = inst_voltage * inst_current;
4    sum_inst_power += inst_power;
5  }
6  real_power = sum_inst_power / number_of_samples;
```

Table 3.1: Real power computation on EmonPi device

In the practical part of this thesis, we are interested in the measurement of the apparent power. It is a product of the Root-Mean-Square (RMS) voltage and the RMS current. The relations of real power, reactive power and apparent power for ideal sinusoidal waves can be represented as [39]:

$$\text{Real Power} = \text{Apparent Power} \cdot cos\ \mu$$

$$\text{Reactive Power} = \text{Apparent Power} \cdot sin\ \mu$$

where $cos\ \mu$ is the power factor.

The RMS is calculated by first squaring the quantity and computing the mean of the values. In the next step, the square-root of the mean of the squares is evaluated. The measurements are facilitated by the fact that the RMS voltage is a fixed value (e.g. 230 V). Consequently, the apparent power can be estimated without first measuring the voltage, by setting the RMS voltage to 230 V. Most of the power devices show a non-linear load. In this case, the power factor is calculated as [39]:

$$\text{Power Factor} = \frac{RealPower}{ApparentPower}$$

Evaluation: The fundamental advantage of the EmonPi is the easy way for a user to show the daily or monthly energy consumption in kWh with the appropriate indication of the cost. If only current values are sampled the EmonPi enables 100 measurements every 20 milliseconds, which is more than acceptable for on-site consumption measurements. However, if the voltage needs to be sampled, as well as the current, the device reads 50 measurements every 20 milliseconds, which is limited by the Arduions read command and computation speed. Further, advantages of the EmonPi are the open-source part of the project, the programmable Arduino, the good documentation, as well as the easy use and configuration of the available application. This makes the EmonPi a good low-budget device for our measurements. In the next section, we describe our testbed in more detail.

3.3.3 Experimental Setup with EmonPi

Measurement Setup: The EmonPi device consists of a Raspberry Pi used with an Arduino and an external current transformer (CT) sensor. With the supplied software of the EmonPi, new values could only be read in a minimal interval of 5 seconds. Since the OpenEnergyMonitor is an open software project, we decided to make the needed adjustments and adapt the available Arduino firmware. Thus, we reduced the variety of different functionalities and adapted the firmware for more power calculation flexibility. The adopted firmware was able to calculate the values for real power, apparent power and the voltage on the level of milliseconds and compute the power factor up to 0.05 seconds.

For the measurements of the alternating current, EmonPi uses clipped current transformer sensors. They are clipped either on the live or neutral wire. For this reason, we added a measurement adapter (brand Voltcraft). It is powered by a line voltage of 230 V/AC, 50 Hz. The exact setup of our environment testbed is shown in figure 3.12.

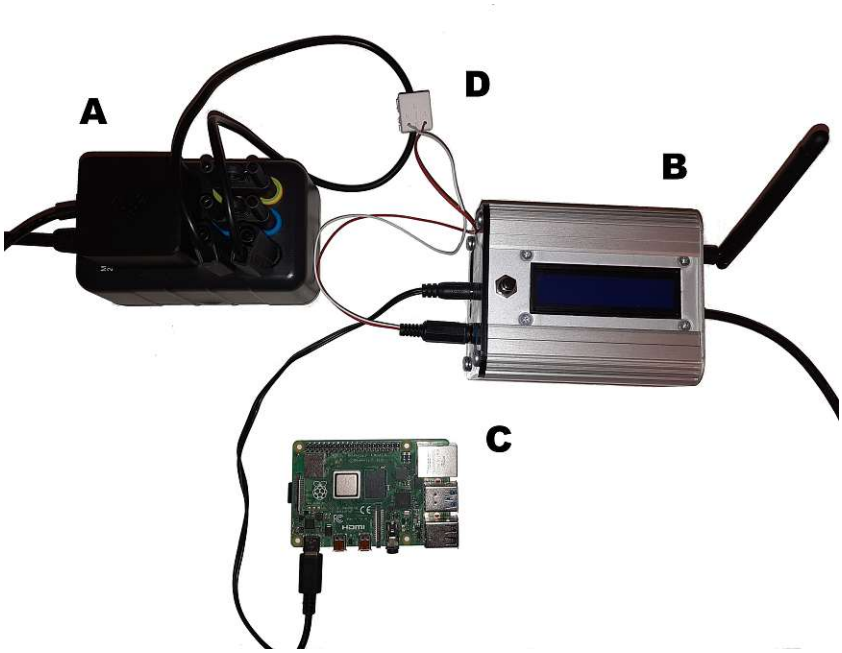


Figure 3.12: EmonPi SetUp: A) Voltcraft measurement adapter, B) EmonPi device, C) Raspberry Pi 4 Model B, D) SCT006 current transformer

We first tried our measurements with the CT Sensor Model: SCT-013 [39], with a current and voltage output type, a maximum current of 100 A, and a non-linearity of ±3% (10% - 120% of rated input current). Since we needed to perform more precise measurements and considering that the results with the SCT013 sensor were not sufficiently precise, we acquired a more sensitive CT sensor - the YHDC SCT006 current transformer [40]. Compared to the SCT-013, where the current lies by 50 mA, the SCT006 has an output

current of 25 mA and a linearity of ±2%. This sensor did seem more suitable for measurements of low-level current computation.

3.4 Conclusion

We evaluated the measurements on several commercially available power measuring devices. Table 3.2 shows the overview of these devices interpreted with the values for the described specifications from section 3.

Characteristic	**HANTEK**	**ENA3**	**EmonPI**	**PMU**
Sampling rate	48 MSa/s	6 kHz	20 Hz	1-60 meas./s for 60Hz
Resolution	8 Bit	NA	10 Bit	14 Bit
Costs	~60 €	~300 €	~130 €	~6000 €
Sample Period	1 s	2 s	0.05 s	1 μs
Accuracy	±3%	±5%	±2%	±0.02%
Measuring Voltage	0 - 5.5 V	100 - 270 VA	110 - 250 VAC	85 - 264 VAC

Table 3.2: Specifications of tested measuring devices

We have aggregated the findings realized at the course of this chapter. In the following table 3.3, we compare the introduced devices based on which characteristic is fulfilled and therefore applicable for our intended use:

Characteristic	**HANTEK**	**ENA3**	**EmonPI**	**PMU**
Sampling rate	X	X	✓	✓
Resolution	✓	NA	✓	✓
Costs	✓	✓	✓	X
Sample Period	X	X	✓	✓
Accuracy	✓	X	✓	✓
Measuring Voltage	X	✓	✓	✓

Table 3.3: Overview of the tested devices and how their specifications are fulfilled

Further measurements of this thesis will be conducted with the best fitting device: the EmonPi - the OpenEnergyMonitor Project. With this in mind, in the following chapter 4 we evaluate the energy consumption of TLS when used on the server (section 4.2) and when used on the client (section 4.3).

CHAPTER 4

Results and Discussion

In this chapter the results of this thesis are presented and discussed. In addition to the technical environment regarding the device used for our measurements, this chapter aims to analyze and compare different versions of cryptographic functions regarding TLS 1.2, concerning power and energy consumption. Section 4.1 details the structure setup with the evaluation device and environment. Section 4.2 discusses the measuring results of the server-side, whereas section 4.3 concentrates on the power consumption of the client-side of the environment. In section 4.4 we compare the evaluations of both sides. This research addresses if there is a significant difference in energy consumption by developing and testing a series of hypotheses indicating how the energy consumption impacts the additional „S“ in HTTPS.

4.1 Environment

For the evaluations, we employed the low-cost, low-power device EmonPi as our target device. The main characteristics, described in section 3.3.2 of the EmonPi are the following [39]:

- ATmega 328p 8 bit microcontroller running with Arduino
- Build-in Raspberry Pi
- 4x USB Ports
- 100 Mbit Ethernet and WLAN
- Micro HDMI output
- 2x CT sensor

During the measurements, the EmonPi was connected to a Voltcraft measurement adapter. This is done to be able to connect the CT sensor to the neutral wire as presented in figure 3.12.

The achieved results, using the measurement tool, analyzed and decided on in chapter 3, are presented in sections 4.2 and 4.3. We measured the power consumption of an HTTP connection and an HTTPS connection which was secured by TLS 1.2. The focus of our measurements lies in the power consumption of version TLS 1.2. We will use a subset of different cipher suites recommended by the National Institute of Standards and Technology (NIST) [13] and the National Security Agency BSI in Germany (Bundesamt für Sicherheit in der Informationstechnik) and perform a power consumption analysis over these subsets. The main metrics for our experimental part of this research are the data transfer time, the amount of executed HTTPS/HTTP requests and the used energy consumption.

We used Nginx, a popular web server with a market share of 32.5% as our HTTPS server. We considered also Apache, with a market share of 36.4%, but decided on the Nginx because of the good documentation [41]. Furthermore, we used PlatformIO for the EmonPi's Arduino to program and upload our adapted version of the firmware. In order to measure the power consumption for varying amounts of HTTP(S) requests we used a custom bash script. This script captured the EmonPi measuring data and generated a defined set of HTTP(S) traffic. Since our setup has to be able to support different cipher suites, we decided on the open-source TLS library - OpenSSL as the TLS implementation [42]. According to Manuel Suárez-Albela et.al. [17], when OpenSSL is used on different hardware or software platforms, the performance, as well as the power consumption while using different cipher algorithms, should be similarly close. Consequently, by using OpenSSL the power consumption of different cipher algorithms should be possible to differentiate over a wide range of different hardware and software platforms.

As discussed in section 2.1.4 a cipher suite is made up of the following cryptographic algorithms [21]:

- Key exchange algorithms such as RSA, ECDH (Elliptic Curve Diffie-Hellman) and ECDHE (Elliptic Curve Diffie-Hellman Ephemeral)
- Authentication algorithms such as Digital Signature Algorithm (DSA), RSA or Elliptic Curve Digital Signature Algorithm (ECDSA)
- Encryption algorithms such as Advanced Encryption Standard (AES), Data Encryption Standard (DES), 3DES and stream-based algorithms like the Rivest's cipher RC4 with its key length (e.g., 128 bits) and its defined mode of operation (e.g., CBC or GCM)

We based our decision, on which cipher suites should be evaluated on the recommendations of the National Institute of Standards and Technology (NIST). In the underlying guideline,

it is stated that one should prefer ephemeral keys over static ones (DHE over DH) and the GCM or CCM modes over CBC mode [13]. Their recommendations influenced our choice of the selected algorithm. We decided on the use of the following algorithms:

1. Elliptic Curve Diffie Hellman Ephemeral (ECDHE) and DHE for key exchange
2. Rivest Shamir Adleman (RSA) and Elliptic Curve Digital Signature Algorithm (ECDSA) for authentication

We considered the power consumption of these algorithms, since they are very common and intensively used [13], but have not been analyzed in the context of energy consumption when applied to a power-plugged device. Table 4.1 gives an overview of the selected ciphers:

Alias	Cipher Suite	Authentication	Key Exchange
CS1	ECDHE-RSA-AES128-GCM-SHA256	RSA	ECDHE
CS2	ECDHE-ECDSA-AES256-GCM-SHA384	ECDSA	ECDHE
CS3	DHE-RSA-AES128-CCM	RSA	DHE
CS4	DHE-RSA-AES128-GCM-SHA256	RSA	DHE

Table 4.1: Cipher suites configuration used for the measurement

For cipher suites CS1, CS3 and CS4 presented in table 4.1 we used a self-signed RSA certificate with a key size of 2048 bits, which is the minimum key size recommended by NIST. According to RSA Security LLC [43], 2048 bits for RSA are expected to be secure until the year 2030. 2048 bits for RSA are equivalent to a security level of 112-bit [13]. For cipher suite, CS2 we used a self-signed ECDSA certificate with curve secp384r1, which is equivalent to the security level of 192-bit. Table 4.2 gives an overview of these values:

Security Level	RSA Key Size	ECDSA Curve
112	2048 bits	secp224r1
128	3072 bits	secp256r1
192	7680 bits	secp384r1

Table 4.2: Comparison of security features for RSA and ECDSA ciphers [13]

The selected algorithms work with block cipher implementation Galois/Counter mode (GCM) or with cipher block chaining (CCM). We used AES-128 to encrypt and decrypt

data in the record layer phase for all ciphers except CS2 which uses AES-256. To ensure data authentications and integrity we used the hash functions SHA256 and SHA384 to create digital signatures.

In the following chapters, we will take a look at the energy consumption of security by comparing HTTPS against HTTP. Since we are interested in the impact of executing TLS, in the first step we will use HTTPS connections to evaluate the TLS energy used and will use HTTP as a reference benchmark. We implement two sets of experiments, where in the first part we will use the customized Nginx server with a self-signed certificate and a script representing the client-side application and measure the server-side. In the second part, we attempt to measure the consumption of the client-side.

4.2 Server-Side Measurements of HTTPS

For the evaluation of the power consumption concerning the server-side, we set up Nginx server on the Raspberry Pi (shown in fig. 3.12 (C)) and a Lenovo T490S, Intel Core i5-8265U CPU @ 1.60GHz x 4 notebook as the client.

We used a test script, which is presented under table 4.3 to measure the power consumption with the EmonPi. In a first step (4) the connection to the EmonPi over SSH is established. The EmonPi firmware is adapted in such a way to constantly output the measuring data, e.g. power consumption, to the serial output. Once the connection over SSH is established, the data of the serial output of the EmonPi is read until the defined number of HTTPS request (9) is executed. To be able to indicate the actual execution time of our test, the start and the end timestamp are recorded afterwards.

For the server-side measurements, we measured the average throughput by generating different amounts of TLS traffic. The client-side was established using the command-line tool and library for transferring data - cURL (9). cURL is capable of getting and sending data via URLs and supports TLS. We used cURL with a self-signed certificate and executed the HTTPS GET request (9) with the cipher suites according to table 4.1. To be able to compare the HTTP GET request with HTTPS, we used cURL and executed the same amount of GET requests as for the HTTP part.

We evaluated the power consumption of RSA (CS1, CS2, CS3) and compared the results while using the exact test setup to measure the consumption of the ECDSA (CS2) algorithm for authentication and analyzed further the impact of the key exchange algorithm (ECDHE/DHE).

In the first evaluation, we send one GET request to the server and observed the energy consumption between HTTP in comparison with the defined cipher suites according to table 4.1. In chapter 2.2 we defined the relation between power and energy. Thus, power multiplied over a time period (e.g. seconds) will specify the energy consumption in Joule. In table 4.4 we present the measuring values of one GET request. We measured the power consumption of the used Raspberry Pi in the idling state with 2,94 W and calculated the

Test script to measure and generate HTTPS requests

```
1  #! /bin/bash
2  date="$(date +%Y_%m_%d_%H:%M:%S)"
3  protocol=protocol_$date.txt
4  ssh emonPi "pipenv run python read_serial/display_serial.py"
       >> "$execution_name/$protocol" &
5  sleep 6
6  start="$(date +%s.%N)"
7  for (counter=1; counter <= Number of requests; counter++)
8  do
9  curl -L -X GET --cacert cacert.pem --ciphers CS1/CS2/CS3/CS4
       >> https://server.name/ </dev/null >/dev/null 2>&1
10 done
12 end="$(date +%s.%N)"
```

Table 4.3: Script to start the measurements and trigger the HTTPS request. The following parameters are used: number of requests, the cipher suite to use and the destination server.

values for *Surplus Consumption* by subtracting the energy consumption from the idle state. With this step, we compute the energy consumption of the complete TLS process.

Ciphers	**Avg. Power** *(Watt)*	**Duration** *(seconds)*	**Total Energy** *(Joule)*	**Idle** *(Joule)*	**Surplus Consum.** *(Joule)*
CS1	2,50	0,05	0,13	0,15	-0,02
CS2	NA	<0,05	NA	NA	NA
CS3	3,17	0,15	0,48	0,44	0,04
CS4	2,74	0,15	0,41	0,44	-0,03
HTTP	3,11	0,40	1,24	1,18	0,06

Table 4.4: Measuring results of one GET request

According to the results in table 4.4, it is not possible to differentiate between the consumed energy consumption of the defined cipher suites. In the case of the execution of one HTTPS call, the results are not clear, due to the lack of samples and the overhead due to the shell script. The values of the consumption are too close to the idling state. Especially, in the case for cipher suite CS2, we could not derive any measurements, since the EmonPi could not measure any values under 0.05 seconds. Further, with the execution of one GET request we collect the time once. However, with the execution of 1000 GET, we also collect the variable time only once. This inputs clearly the derived consumption. For this reason, we decided to evaluate the energy consumption of 10 GET-requests

presented in table 4.7. On the following pages, the results of these measurements are discussed and represented as plots, where on the x-axis the time needed for execution (in seconds) and on the y-axis the power consumption used (in Watts) is presented.

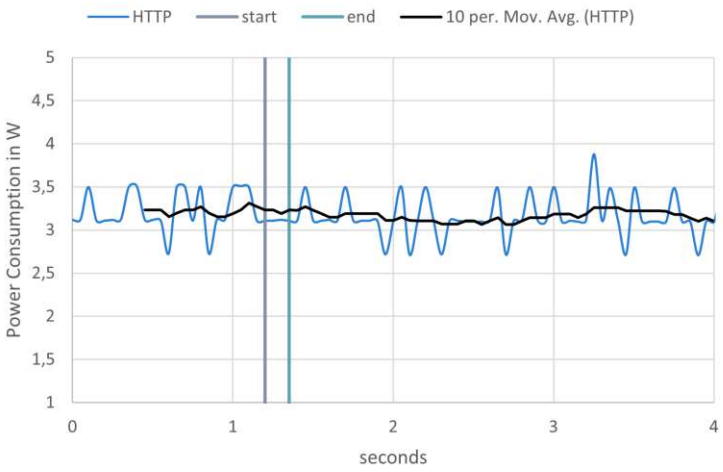


Figure 4.1: Server-side execution of 10 HTTP GET requests

The results of the payload of 10 requests show that ciphers that used Elliptic Curve Diffie Hellman (ECDHE) as their key exchange algorithm are more efficient than ciphers CS3 and CS4 which used Diffie Hellman (DHE). This output was expected, since ECDHE uses elliptic curves to compute the shared secret, while DHE uses modular arithmetic. In each handshake, a different DH public value will be created, which requires more cryptographic operations. For the ECDHE algorithm we used a secp384r1 curve, which provides a 192 bit security level. Research in the field of IoT devices confirmed our observations. According to the analysis of Gerez et.al. [29] the most computationally intensive step of the handshake process is the key exchange method of the pre-master secret, where the DHE key exchange is arithmetically more intensive than ECDSA which is visible in the measurements.

From figure 4.1 and figure 4.2 it is apparent that there is a significant distinction regarding the duration of the data transmission, when the server is called via HTTP and when it is called via HTTPS. The short duration of HTTP impacts the energy consumption visibly. In comparison to the duration of CS1, the HTTP request took 79% less time. For CS2, which performed the best for all ciphers, HTTP took 67% less time for the GET request to execute. When compared to CS3 and CS4 the HTTPS requests took 91% and 90% less time respectively.

The overall average power consumption, as well as energy consumption of the specified cipher suites, over the implementation of 10-GET request, are defined in table 4.7. Considering that the use of 10 GET requests is still a small number of requests for a real-world server, we analyzed the power consumption and the execution time of 100

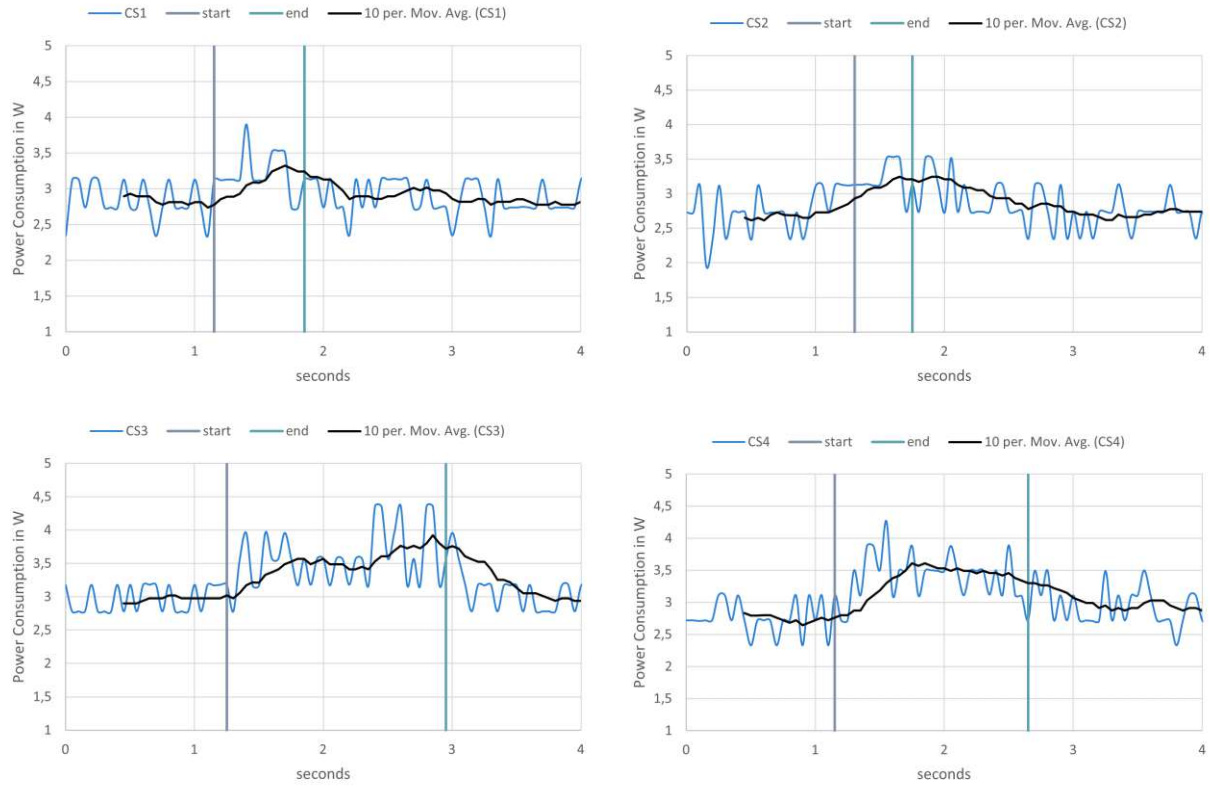


Figure 4.2: Server-side execution of 10 HTTPS requests for cipher suites CS1, CS2, CS3, CS4

HTTPS requests under the same conditions.

The following figures 4.3, 4.4, 4.5, 4.6 and 4.7 show the execution of these defined cases.

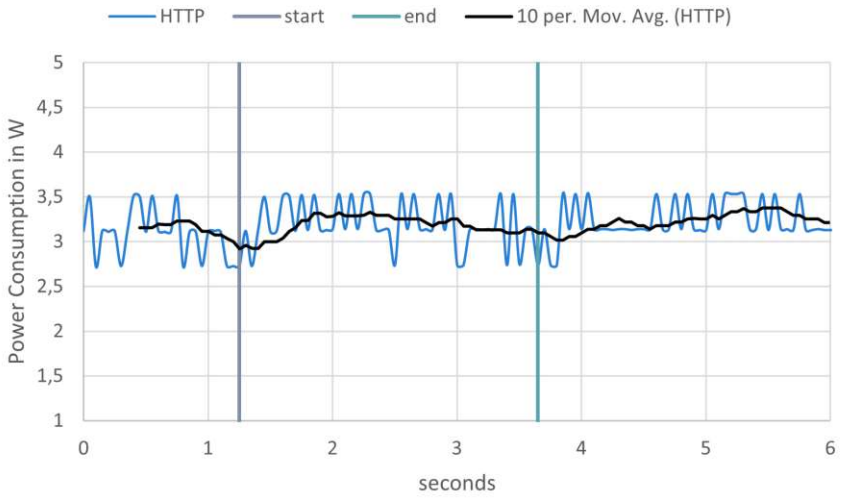


Figure 4.3: Server-side execution of 100 HTTP requests

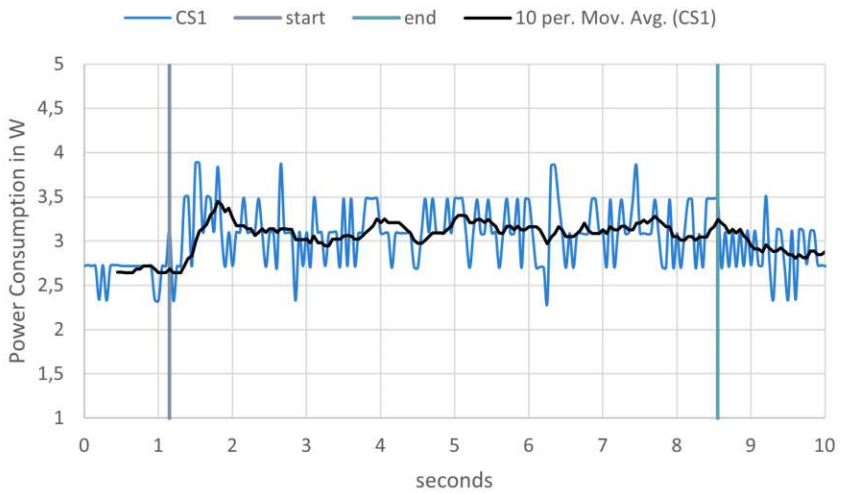


Figure 4.4: Server-side execution of 100 HTTPS requests with CS1

The power consumption (in *Watt*) over the time period (*seconds*) and the energy consumption (in *Joule*) of the cipher suites over the defined implementation of 100-GET request are defined in table 4.5. Contrary to the payloads before, with one hundred requests it was possible to see differences in the energy consumption between the idling state and the execution of 100 HTTPS GET requests.

We evaluated the total energy consumption and compared it to the idling state to be able to determine the energy consumption of TLS. Even with this small number of requests,

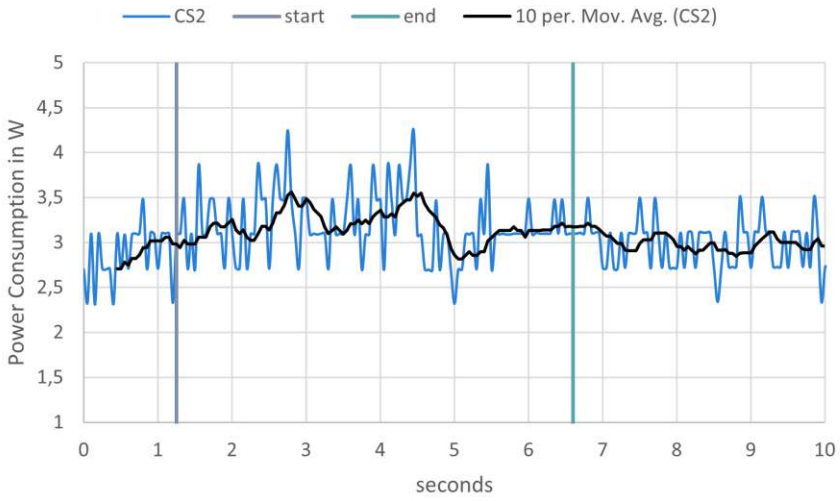


Figure 4.5: Server-side execution of 100 HTTPS requests with CS2

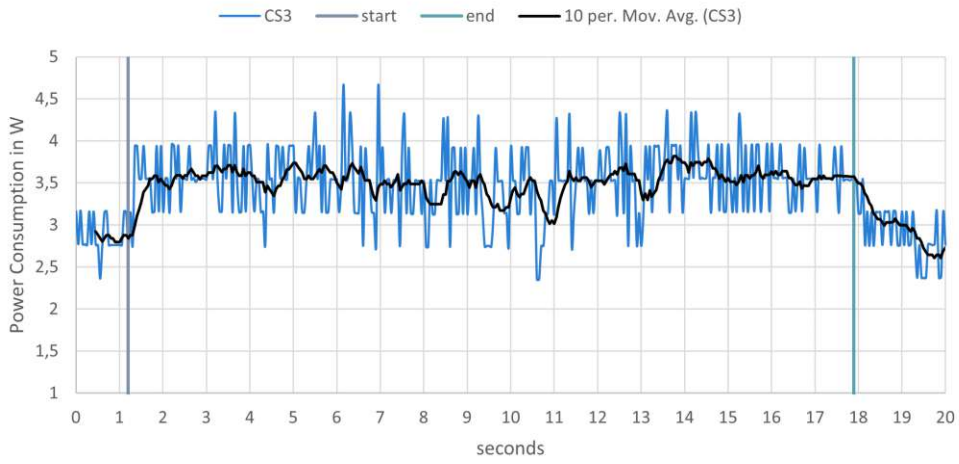


Figure 4.6: Server-side execution of 100 HTTPS requests with CS3

it was possible to measure changes in energy consumption. Cipher suite CS2 performed the best for one hundred requests. When compared with CS3, which consumed the most energy, CS2 used 86% less energy. CS4, which used the same key exchange algorithm (DHE-RSA) performed similar, where CS2 used around 84% less energy than CS4. The energy consumption of CS1 and CS2 was close, with CS1 consuming 4% more energy than CS2.

If we use HTTP as the baseline for 100-GET requests, we conclude the following:

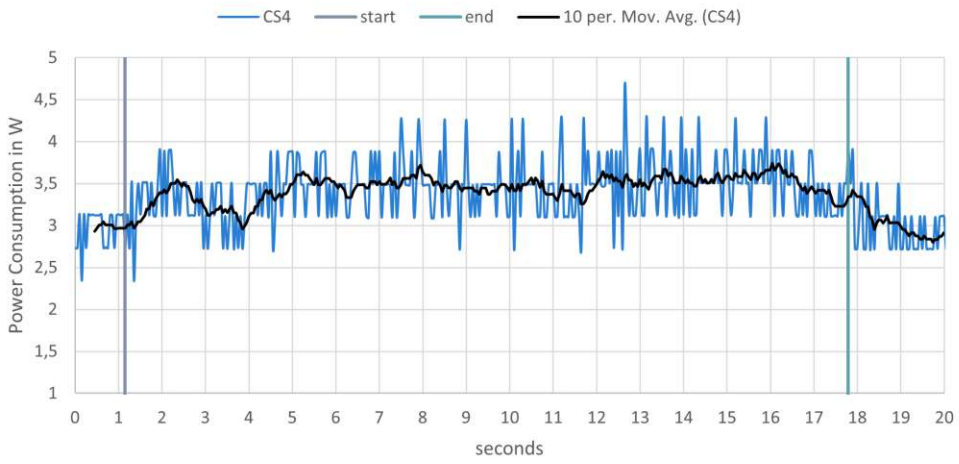


Figure 4.7: Server-side execution of 100 HTTPS requests with CS4

Ciphers	**Avg. Power** *(Watt)*	**Duration** *(seconds)*	**Total Energy** *(Joule)*	**Idle** *(Joule)*	**Surplus Consum.** *(Joule)*
CS1	3,13	7,40	23,18	21,76	1,42
CS2	3,20	5,35	17,09	15,73	1,36
CS3	3,53	16,70	58,93	49,09	9,84
CS4	3,45	16,64	57,39	48,92	8,47
HTTP	3,19	2,40	7,65	7,06	0,59

Table 4.5: Server-side measuring results of 100 GET requests

- HTTP used 58% less energy than CS1 and used 68% less time to perform the same actions.
- For HTTP, the energy consumption decreased 56% when compared to the consumption of cipher CS2, and the execution time is reduced for 55%.
- CS3 and CS4 performed relatively similar and HTTP consumed respectively 94% and 93% less energy for the same time period, and HTTP needed for the same amount of requests 86% less time.

For further analysis, we evaluated the energy consumption with four different amounts of requests. Table 4.7 shows the energy consumption of different amounts of HTTP or HTTPS requests with different cipher suites. It is obvious that higher amounts of requests increase the energy consumption approximately linearly. In order to remove any uncertainties due to low sampling rate and to smooth out the random power fluctuations

we did final measurements with 10.000 HTTPS requests as well. Table 4.6 gives an overview of these values:

Ciphers	**Avg. Power** *(Watt)*	**Duration** *(seconds)*	**Total Energy** *(Joule)*	**Idle** *(Joule)*	**Surplus Consum.** *(Joule)*
CS1	3,31	800,65	2642,61	2353,90	288,71
CS2	3,24	556,63	1800,82	1636,50	164,32
CS3	3,62	1750,83	6169,79	5015,13	1154,66
CS4	3,49	1650,60	5761,35	4852,76	908,59
HTTP	3,22	254,12	817,55	747,11	70,44

Table 4.6: Server-side measuring results of 10.000 GET requests

GET-request	**Cipher**	**Energy Consumption (EC)**			
		Total *(Joule)*	**Idle** *(Joule)*	**Surplus Consum.** *(Joule)*	**EC per 1-GET** *(mJ)*
10 GET	CS1	2,25	2,06	0,19	18,62
	CS2	1,44	1,32	0,12	12,14
	CS3	6,24	5,14	1,10	109,83
	CS4	5,09	4,41	0,68	68,23
	HTTP	0,47	0,44	0,03	2,59
100 GET	CS1	23,18	21,76	1,42	14,21
	CS2	17,10	15,73	1,37	13,66
	CS3	58,93	49,09	9,84	98,48
	CS4	57,39	48,92	8,47	84,65
	HTTP	7,65	7,06	0,59	5,95
1.000 GET	CS1	252,10	228,47	23,63	23,63
	CS2	174,26	155,99	18,27	18,26
	CS3	624,53	511,93	112,60	112,59
	CS4	575,64	484,74	90,90	90,90
	HTTP	82,58	75,57	7,01	7,01
10.000 GET	CS1	2642,61	2353,90	288,71	28,87
	CS2	1800,82	1636,50	164,32	16,43
	CS3	6169,79	5015,13	1154,66	115,47
	CS4	5761,35	4852,76	908,59	90,86
	HTTP	817,55	747,11	70,44	7,04

Table 4.7: Energy consumption (in Joule) for server-side test cases

Figure 4.8 shows the linear increase of energy consumption depending on the amount of

executed GET-request.

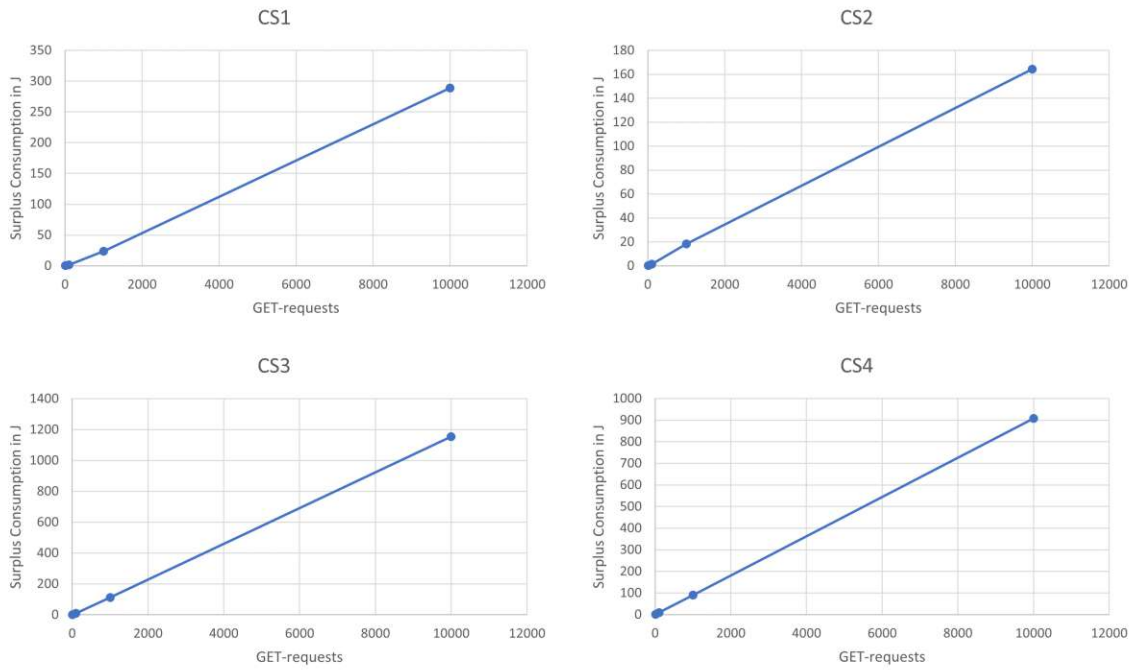
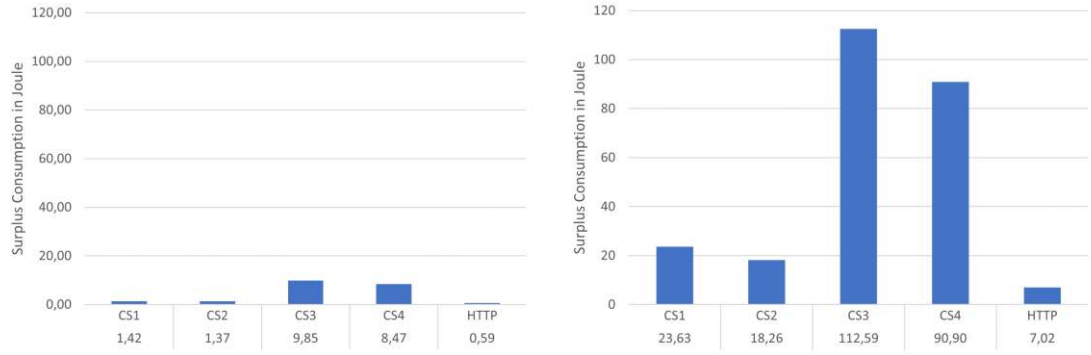


Figure 4.8: Linear Representation of the server-side surplus consumption over all GET-requests

NIST [13] states that one should prefer GCM or CCM modes over CBC mode. With the cipher suites CS3 and CS4 we analyzed the impact regarding consumed energy of these two modes. The results in table 4.7 demonstrate that CS4, with AES in GCM mode with 128-bit keys for bulk encryption performed on average 23,17% better than CS3 in CCM mode.

Furthermore, the data shows that cipher suite CS2, which used the Elliptic Curve Diffie Hellman for key exchange and the Elliptic Curve Digital Signature Algorithm for the authentication (ECDHE-ECDSA) consumed less energy for every single test case. The construction of ECDSA around elliptic curves and the use of shorter key length as for RSA enables ECDSA lower computation as well as lower network power, while achieving the same level of security [15]. For this reason, the energy consumption in comparison to RSA used in CS1, CS3 and CS4 is visibly lower.

The results indicate that CS2 has the shortest processing time, but CS1 also presented low values. In contrast, ciphers CS3 and CS4, which both use DHE for the key exchange, consumed more processing time. On average, cipher CS1 consumed in total 56% less processing time than CS3 and 53% less processing time than CS4. For cipher suite CS2, the reduction of the processing time is around 70% and 67% compared to ciphers CS3 and CS4. Figures 4.9 and 4.10 present a graphical overview of the values from table 4.7.



(a) Call of 100 HTTPS request

(b) Call of 1000 HTTPS request

Figure 4.9: Energy consumption (in Joule) of the server-side test cases and comparison of the different cipher suites defined in table 4.1

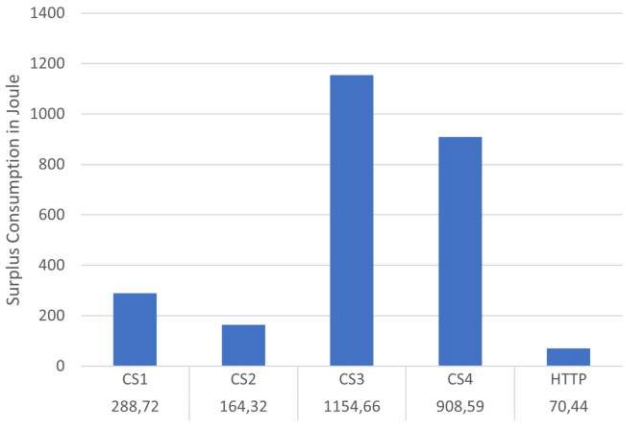


Figure 4.10: Energy consumption (in Joule) of the server-side test case for 10.000 GET requests

The figures confirm, that using ECDSA instead of RSA for the TLS signature, is beneficial for energy consumption. Cipher suite CS2 - ECDSA-ECDSA-AES256-GCM-SHA384 outperforms all other ciphers, which used RSA for key exchange algorithms in every test case.

It can be stated that from 100 GET-requests the consumption shows relatively similar proportional results when HTTP is compared to HTTPS. For the following evaluation, we analyze the energy consumption increase when HTTPS is used instead of HTTP. We evaluate the overall increase from 100 HTTPS requests up using table 4.7. CS1 energy consumption increased on average by a factor of 3,3 and for CS2 by a factor of 2,4 while CS3 and CS4 saw an increased energy consumption by a factor of 16,4 and 13,4 respectively.

The data furthermore shows that for all test cases the ciphers, which used ECDHE for the key exchange are more energy efficient than ciphers CS3 and CS4, which used Diffie Hellman (DHE). Cipher suites CS3 and CS4 express higher values, not only for the processing time, but for the consumed energy as well. This is not surprising, if we take into consideration that the key exchange with DHE is computationally more expensive than ECDHE [29]. The following table 4.8 presents the percentage of the decreased energy consumption when the best performing cipher CS2 is compared to the remaining ciphers.

GET-requests	**CS1** *(%)*	**CS3** *(%)*	**CS4** *(%)*
100-GET	3,52	86,08	83,83
1.000-GET	22,68	83,77	79,90
10.000-GET	43,08	85,77	81,91
Average	**23,09**	**85,21**	**81,88**

Table 4.8: Overall energy consumption reduction (in percentage) of ECDSA (CS2) in comparison to RSA (CS1, CS3 and CS4)

Suárez-Albela et al. [15] compared different cipher suites executed on an IoT node in terms of energy consumption. The results are similar in the fact that cipher suites which used ECDSA consumed less energy than RSA, even in the field of IoT. The results of this research paper reflect our findings: ECDSA outperforms RSA in every scenario. The energy consumption of the ECDSA algorithm used in CS2, is reduced by 27% when compared to cipher suite CS1, which uses RSA for the authentication algorithm. This pattern is also valid for ciphers CS3 and CS4, where the energy consumption is reduced by 86% for CS3 and 82% for CS4. This is due to the fact, that both of these cipher suites use DHE-RSA as their key exchange and authentication algorithm.

The order of the cipher suites from best to least energy efficient for the server-side is the following: CS2, CS1, CS4, CS3. The ciphers which use ECDHE consumed on average 81% less energy when compared to CS3, and 76% on average when compared to CS4. We showed that ECDSA outweighs RSA's energy consumption for all test cases. However, while deciding between RSA and ECDSA encryption it should be taken into account that RSA has stronger signatures than ECDSA, but ECDSA needs more computations for the certificate verification. Further, we should have in mind that the used cipher CS2 (ECDSA) provided a security level of 192 bits, while on the other hand we used a security level of 112 bits for the RSA encryption. Even in this case, cipher suite CS2 presents a better alternative due to lower consumption. In the following chapter, we evaluated the consumption of the client-side.

4.3 Client-Side Measurements of HTTPS

For the client measurements we used the same measuring environment as for the server. The only difference is within our test script where the cURL call is now executed on the Raspberry Pi. The server was set up on a Lenovo T490S Intel Core i5-8265U notebook.

In the following, we measure the average throughput from the client side by generating large amounts of TLS traffic. The same power measurements were taken for the client side. For CS2 we used the identical security level of 192 bits, with curve secp384r1 and for CS1, CS3 and CS4, 112 bit security level with 2048 bits for RSA. The derived values for 10 GET requests are presented in table 4.11. In figures 4.11 and 4.12 the payload of ten HTTPS and HTTP requests are presented.

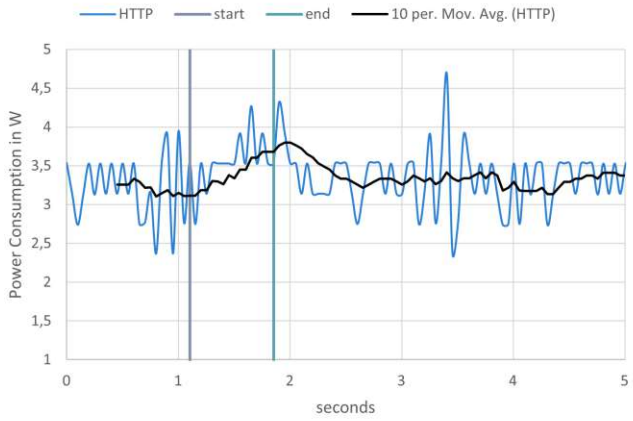


Figure 4.11: Client-side execution of 10 HTTP GET requests

It was not surprising, that the assessed values are similar to the server-side. For ten HTTPS requests the order of the cipher suites from best to least energy efficient is CS1, CS4, CS2, CS4. As for the server-side we take a closer look at the consumption of 100 HTTPS requests, presented in table 4.9 and figures 4.13, 4.14, 4.15, 4.16 and 4.17.

We again evaluated the total energy consumption and compared it to the idling state and presented the TLS energy consumption, named *Surplus Consumption*. For 100 GET requests cipher CS2 consumed around 14% more energy than HTTP without any encryption. CS3 and CS4 with DHE performed again similar, and the energy consumption increased by a factor of 4,1 for CS3 and by a factor of 3,1 compared to CS2.

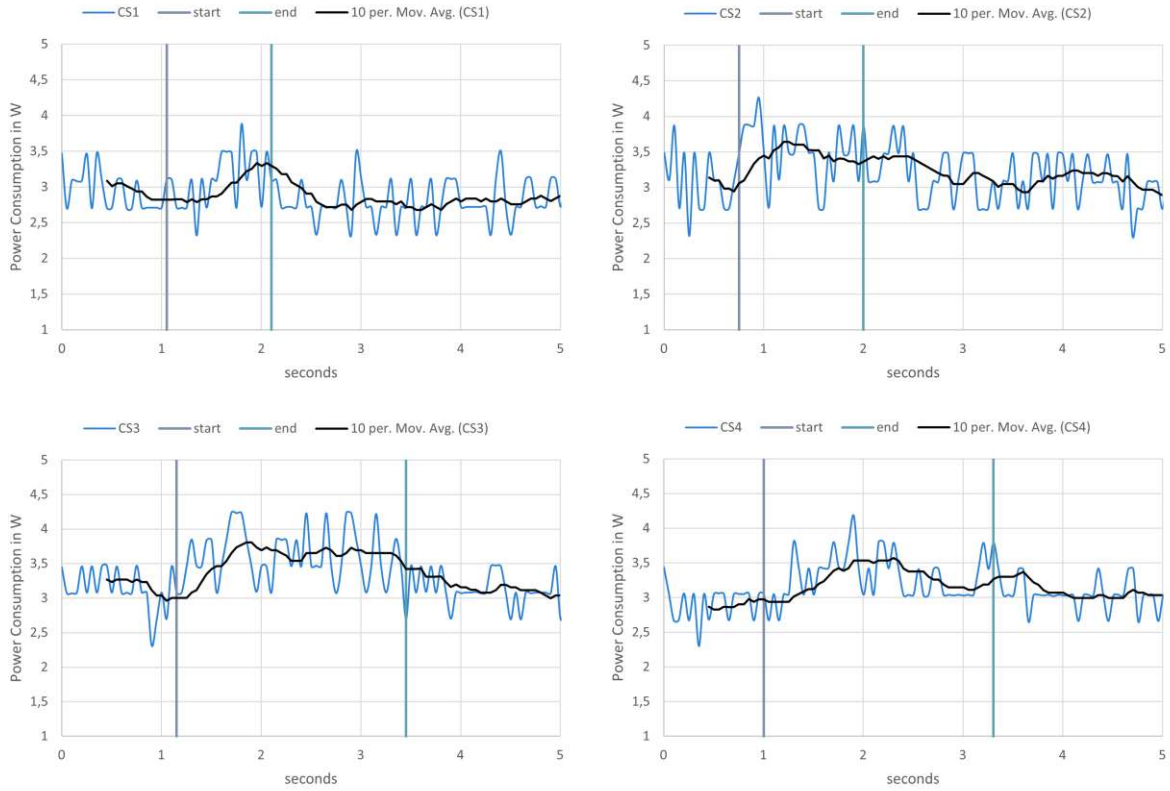


Figure 4.12: Client-side execution of 10 HTTPS GET requests for cipher suites CS1, CS2, CS3, CS4

Further, we observe that there is a significant distinction in the time duration between HTTP and HTTPS. If we use HTTP as our baseline, we conclude the following:

- HTTP used 49% less energy than CS1 and used 48% less time to perform the same actions.
- For HTTP, the energy consumption decreased 12% when compared to the consumption of cipher CS2, and the execution time period is reduced for 63%.
- CS3 and CS4 performed relatively similar and lasted the longest. HTTP consumed respectively 79% and 77% less energy for the same time period, and HTTP needed for the same amount of requests 81% less time.

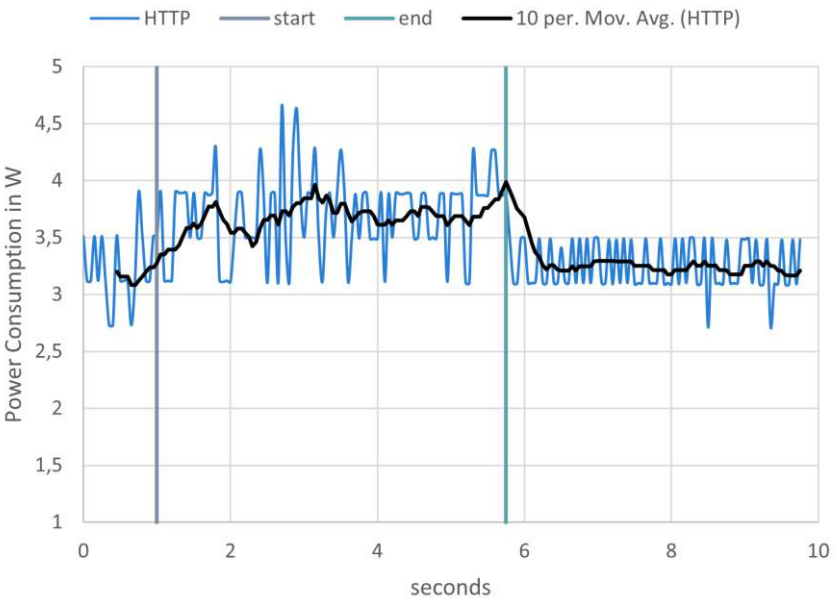


Figure 4.13: Client-side execution of 100 HTTP requests

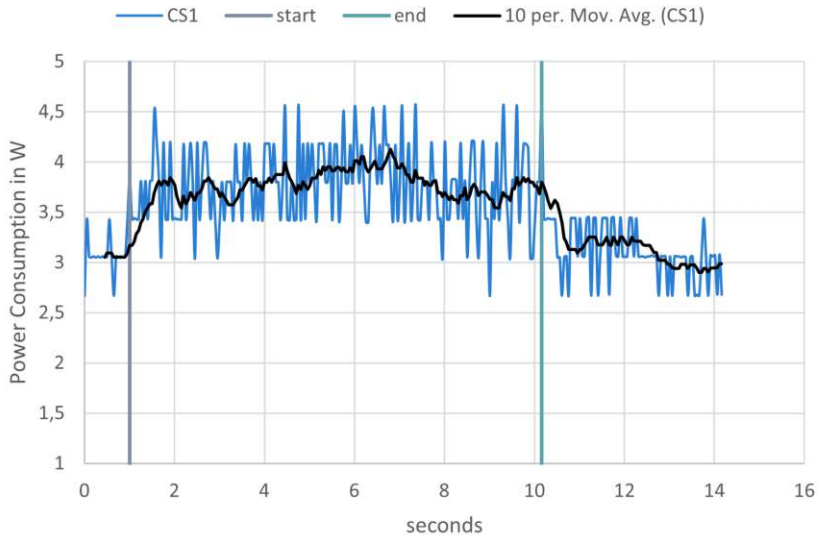


Figure 4.14: Client-side execution of 100 HTTPS requests with CS1

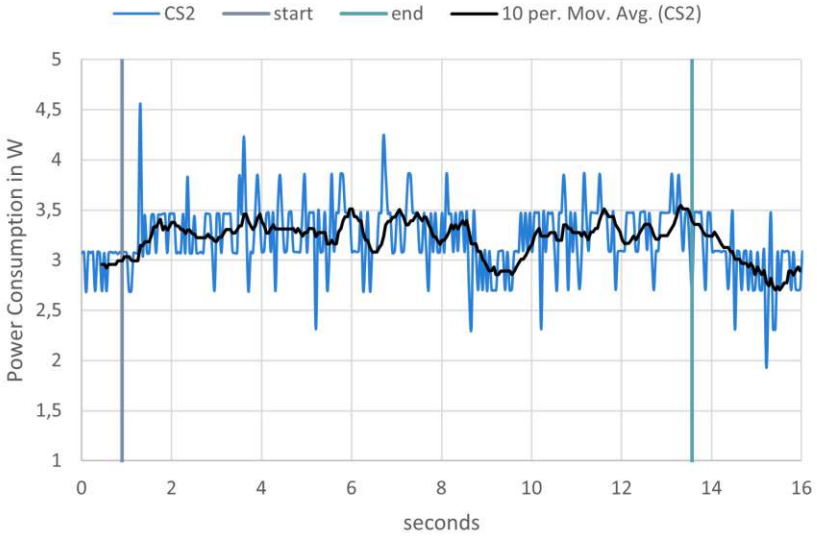


Figure 4.15: Client-side execution of 100 HTTPS requests with CS2

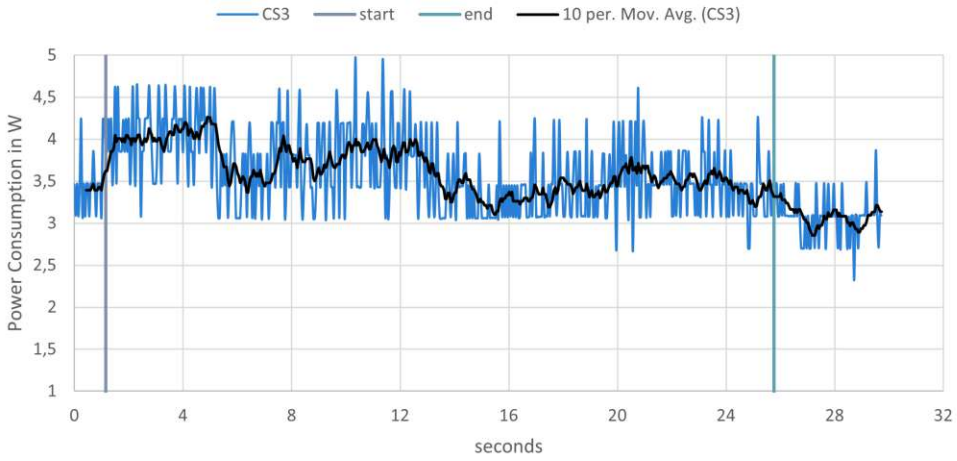


Figure 4.16: Client-side execution of 100 HTTPS requests with CS3

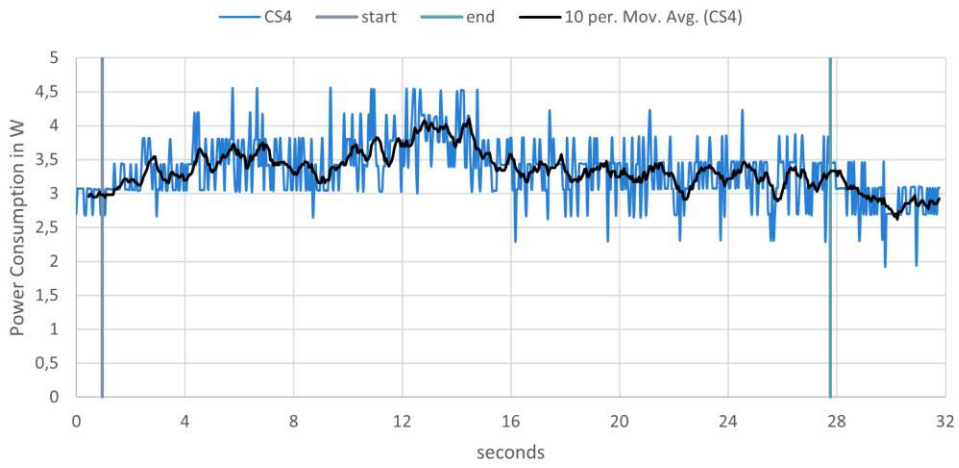


Figure 4.17: Client-side execution of 100 HTTPS requests with CS4

Ciphers	**Avg. Power** *(Watt)*	**Duration** *(seconds)*	**Total Energy** *(Joule)*	**Idle** *(Joule)*	**Surplus Consum.** *(Joule)*
CS1	3,78	9,16	34,64	26,92	7,72
CS2	3,27	12,67	41,36	37,24	4,12
CS3	3,63	24,61	89,32	72,35	16,97
CS4	3,41	26,81	91,40	78,83	12,57
HTTP	3,70	4,75	17,59	13,96	3,63

Table 4.9: Client-side measuring results of 100 GET requests

For further analysis, to review the consumption closer, especially for the case CS2, we evaluated the energy consumption with four different amounts of requests. Table 4.11 shows the energy consumption of these requests executed with different cipher suites. As for the server, in order to remove any uncertainties, we did final measurements with 10.000 HTTPS requests as well. Table 4.10 gives an overview of these values.

Figure 4.18 shows the linear increase of energy consumption depending on the amount of executed GET-request.

When we compare the energy consumption of the encryption algorithm with two defined modes of operation - CCM for cipher CS3 and GCM for CS4. Cipher CS4 with AES in GCM mode and 128-bit keys for bulk encryption, performed on average 29% better than CS3 in CCM mode.

Cipher CS2 performed the best for the execution up from 100 HTTPS requests. With

Ciphers	Avg. Power *(Watt)*	Duration *(seconds)*	Total Energy *(Joule)*	Idle *(Joule)*	Surplus Consum. *(Joule)*
CS1	3,50	905,34	3175,57	2661,69	513,88
CS2	3,62	1012,92	3669,17	2977,98	691,19
CS3	3,44	2520,81	8675,04	7411,18	1263,86
CS4	3,31	2740,15	9082,55	8056,04	1026,51
HTTP	3,70	483,01	1786,19	1420,04	366,15

Table 4.10: Client-side measuring results of 10.000 GET requests

GET-request	Cipher	Energy Consumption (EC)			
		Total *(Joule)*	Idle *(Joule)*	Surplus Consum. *(Joule)*	EC per 1-GET *(mJ)*
10 GET	CS1	3,26	3,09	0,17	16,82
	CS2	4,41	3,67	0,73	73,50
	CS3	8,28	6,76	1,52	151,97
	CS4	7,60	6,76	0,84	84,41
	HTTP	2,67	2,21	0,46	45,99
100 GET	CS1	34,64	26,92	7,72	77,13
	CS2	41,36	37,24	4,12	41,17
	CS3	89,32	72,35	16,97	169,65
	CS4	91,40	78,83	12,57	125,69
	HTTP	17,59	13,96	3,63	36,32
1.000 GET	CS1	372,63	324,08	48,55	48,55
	CS2	408,23	366,65	41,58	41,58
	CS3	864,53	727,82	136,71	136,71
	CS4	862,84	762,10	100,74	100,74
	HTTP	176,48	141,66	34,82	34,82
10.000 GET	CS1	3175,57	2661,69	513,87	51,39
	CS2	3669,17	2977,98	691,19	69,11
	CS3	8675,04	7411,18	1263,86	126,39
	CS4	9082,55	8056,04	1026,51	102,65
	HTTP	1786,19	1420,04	366,15	36,61

Table 4.11: Energy consumption (in Joule) for client-side test cases

the execution of more than 1000 HTTPS-requests, the cipher CS2 performed about 17% better than CS1. However, with the execution of 10.000 HTTPS calls, cipher CS2 consumed more energy. In this case, CS2 consumed 35% *more* energy than CS1. We

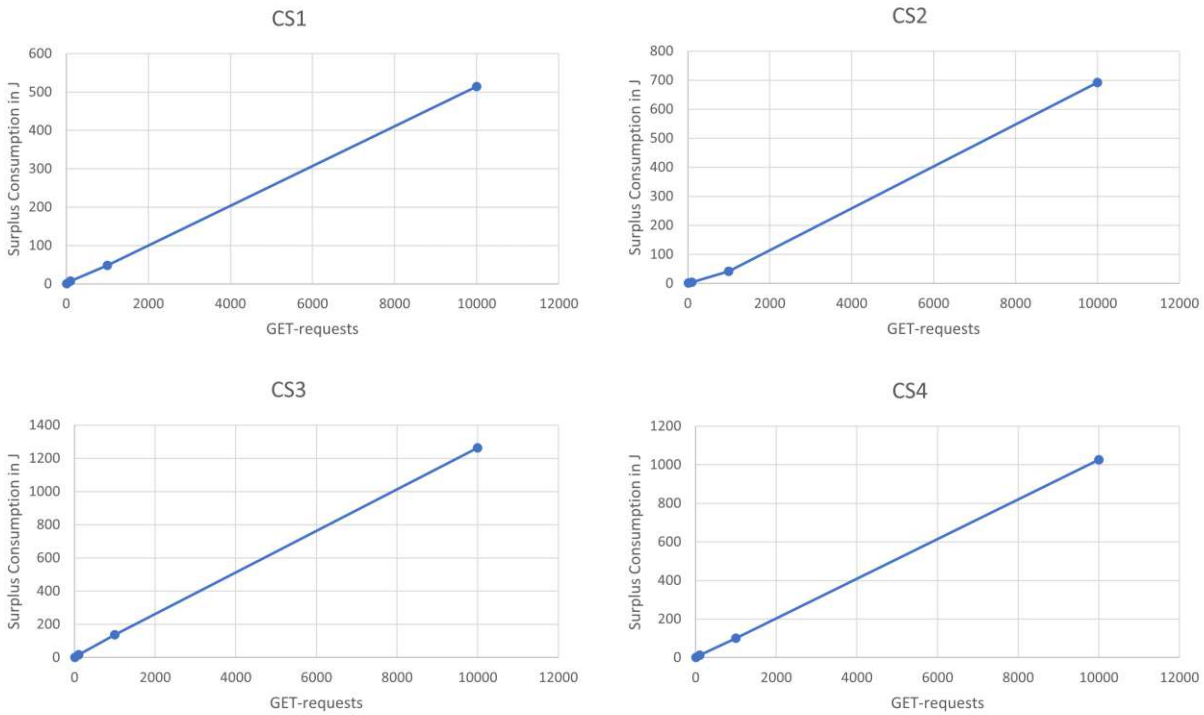
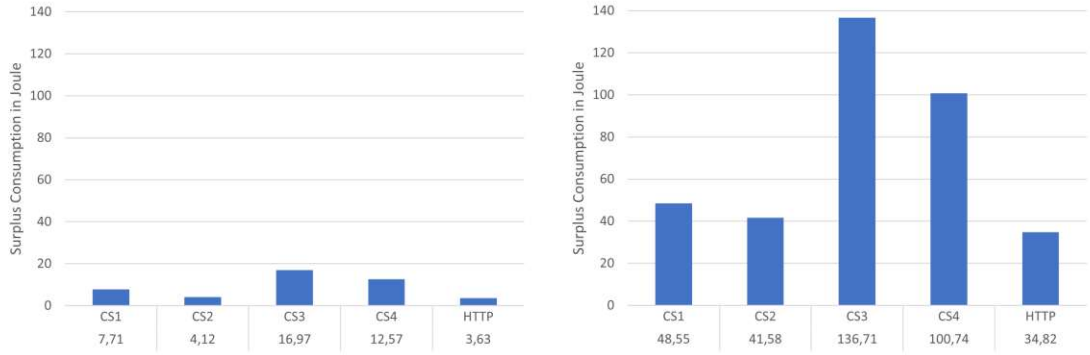


Figure 4.18: Linear Representation of the client-side surplus consumption over all GET-requests

can conclude, that with increasing HTTPS requests on the client site, CS1 performs continuously better. By contrast to CS3 and CS4 which both used DHE for the key exchange, ciphers CS1 and CS2 both present minimal differences in energy consumption. For the average use over HTTPS up from one hundred requests, we conclude:

- CS1 consumed 59% less energy compared to CS3 and used 61% less time to perform these actions.
- CS1 consumed 47% less energy when compared to the consumption of cipher CS4 and the execution time period is reduced for 63%.
- For cipher suite CS2, which performed similar to CS1, the ciphers CS3 and CS4 consumed respectively 63% and 53% less energy, for 53% and 56% less processing time.

For a better visualization of the energy consumption, the next figures 4.19 and 4.20 present a graphical overview of the values from table 4.7.



(a) Call of 100 HTTPS request

(b) Call of 1000 HTTPS request

Figure 4.19: Representation of cipher suites from table 4.1 compared when executed with different amount of GET requests

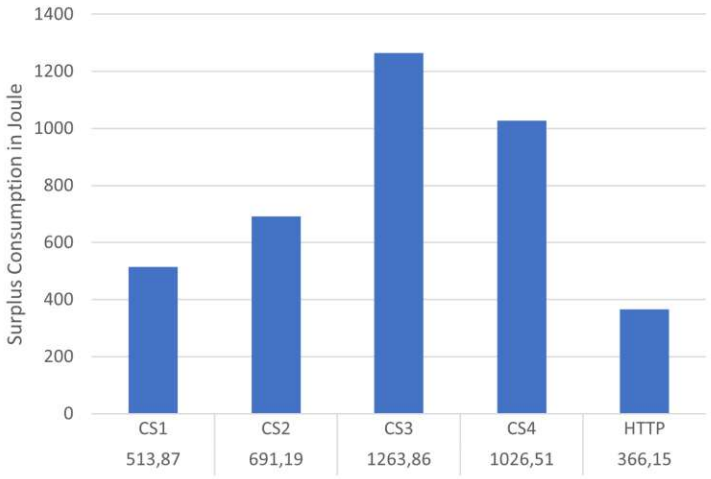


Figure 4.20: Energy consumption (in Joule) of the client-side test case for 10.000 GET requests

The figures indicate that using ECDSA instead of RSA for the TLS signature is beneficial for energy consumption. The most energy-consuming cipher suite is CS3. It can be stated that the consumption shows relatively similar proportional results when HTTP is compared to HTTPS. For the following evaluation, we analyze the energy consumption increase when HTTPS is used instead of HTTP. We evaluate the increase for the case of 10.000 HTTPS requests to smooth out the random power fluctuations. In this case, cipher suite CS1 consumed 40%, while CS2 consumed 89% more energy than HTTP. CS3 and CS4 saw an increased energy consumption by a factor of 3,45 and 2,80 respectively.

Our study confirmed, that even for the client-side, the energy consumption of ECDSA certificate verification considerably outweighs RSA's energy consumption for heavier

signature generation.

We further verified the same results as for the server-side: for all test cases, the ciphers which used ECDHE for the key exchange are more efficient than ciphers with Diffie Hellman (DHE). The only exception is the execution of 10 HTTPS requests, where cipher suite CS4 showed an unexpected energy consumption. This is caused by the small number of samples available to measure the energy consumption, since the execution time is very short. In general, DHE is computationally more expensive than ECDHE. Therefore, they are performing visibly worse in every test case, except the above-mentioned anomaly.

As a final remark, we observed that at the client-side, ECDHE-ECDSA performed the best, but CS1 with ECDHA-RSA performed well from 10.000 HTTPS calls up. The difference between the energy consumption between ECDSA and RSA is less noticeable than on the server-side. Potlapally et al. [27] observed at their research that this difference emerge from the asymmetric operations costs for signing and verification of RSA. ECDSA on the other hand uses nearly symmetrical costs.

4.4 Comparison of Server and Client Results

With the performed tests, we aimed to determine the impact of different cipher algorithms on the energy consumption of power-plugged devices. For this, we analyzed the transfer time, the amount of executed HTTPS/HTTP requests, the energy consumption and assessed different key exchange and authentication algorithms for secure communication over the network. To understand the differences between the cipher suites and how the usage of different algorithms (ECDSA/RSA, ECDHE/DHE, GCM/CCM) can impact the energy consumption, an analysis was carried out. In the following, we compare the achieved results between the energy consumption of the server and the consumption of the client-side.

Up until 100 HTTPS requests, due to the short execution time, the sampling rate of the EmonPi caused issues in comparing the energy consumption. For this reason, we used the evaluations from 100 HTTPS as our measurement reference and compared the energy consumption between the server and the client. For the evaluations, we used a 112 bits security level for RSA in contrast with the more secure security level of 192 bits for ECDHE. Even with this difference in mind, ECDHE is more efficient regarding the energy consumption at both, the server-side as well as the client side. There is a visible increase in energy consumption for all cipher suites on the client side. This is due to the fact that we used an SSH connection to start the HTTP traffic via curl on the Raspberry Pi. For the HTTP request, the client energy consumption increased on average by a factor of 5,28. The energy consumption of cipher suite CS1 increased by a factor of 3,09, for CS2 by a factor of 3,16 and for CS3 and CS4 by a factor of 1,34 and 1,24 respectively.

Our analysis showed that the handshake of RSA used more time than in ECDSA cipher. This result was expected, even in the case that for CS2 (ECDSA), where the symmetric key and the hash algorithm has a higher security level (AES128-GCM-SHA256 for CS1

and AES256-GCM-SHA384 for CS2). With the increasing amount of HTTPS requests, the confidence of the measured energy consumption values increases. For this reason, we evaluated the energy consumption of 10.000 HTTPS requests. In this case, the use of ECDHE can save up to 78% of the energy compared to DHE on the server-side, and up to 47% of energy on the client.

For the server the total energy consumption can be reduced for 70% if ECDSA (CS2) is used instead of RSA. For the client, the situation is different if we look at the consumption of 10.000 HTTPS requests for cipher suites CS1 and CS2. In this case, CS1 with ECDHE-RSA consumed 26% less energy than CS2. The performance of cipher suites CS3 and CS4 was consistently worse.

In closing, if we look back to the table 4.1, the ciphers can be ranked based on their energy consumption:

Ranking	Cipher Suite	Authentication	Key Exchange
1	CS2	**ECDSA**	**ECDHE**
2	CS1	**RSA**	**ECDHE**
3	CS4	RSA	DHE
4	CS3	RSA	DHE

Table 4.12: Ranking of cipher suites based on energy efficiency

As it can be seen in table 4.12, ciphers which use ECDSA as their authentication algorithm and ECDHE as the key exchange algorithm consume the lowest amount of energy. Ciphers which used RSA-ECDHE showed also good performance regarding their energy usage. The worst performing ciphers are ciphers which use DHE as their key exchange algorithm. An additional impact to the energy consumption of cipher suite CS3 is the usage of AES in CCM mode.

CHAPTER 5

Conclusion

In this final chapter, we summarize the contributions of this thesis and reflect on the findings. In addition, we present further ideas for extensions of the proposed measurement environment and propose further possible measurements.

In the first part of the thesis, we have examined a set of different measurement tools and presented a low-cost measurement environment for assessing the power consumption on the level of milliseconds. In order to find such an environment, we defined the most important requirements that have to be fulfilled. Our goal was to enable the measurements of the power consumption before the power supply of a device. We tried to cover a wider range of devices and offered measurements on different single-board computers, e.g a Raspberry Pi, Arduino or a notebook. With this, we presented one of the first setups for analyzing TLS on power-plugged devices. However, the presented measurement setup comes with some limitations. It was challenging to find the best-suited setup, since there are very few tools that could be considered for this kind of measurements due to the required precision.

Nevertheless, with the presented low-cost setup we offer a possibility to measure low values for power measurements. This setup is based on an open-source project and is therefore affordable and adaptable. Other devices, with a higher sampling rate and accuracy are available on the market, but the cost of these devices made them impractical for our use. Thus, the evaluation of the power consumption of our findings could be executed with a more powerful setup although at a higher cost. The used measuring setup is not optimal, but a middle way between the ability to measure small scale power changes and the cost factor.

After providing the background information and a literature review on TLS and the fundamentals of power measurements in chapter 2 as well as the construction of our measurement setup for evaluation in chapter 3, we presented our results of all parties involved in the usage of a TLS connection. We determined the impact of the usage of

different cipher suites regarding the power consumption on the server-side as well as the client-side.

With the selected ciphers we tried to cover the most available and recommended ciphers by the National Institute of Standards and Technology (NIST) and the National Security Agency BSI (Bundesamt für Sicherheit in der Informationstechnik) in Germany. We performed a power consumption analysis over this subset of cipher suites with a focus on the amount of executed HTTP/HTTPS requests, the used transfer time and the energy consumption. Looking at the presented results in chapter 4 the evaluation gives a good impression of the energy consumption of different cipher suites used.

One of the main metrics was the comparison of the power consumption of HTTP and HTTP over TLS. It has been shown that cipher CS2 (ECDHE-ECDSA) consumed less energy for every test case and outperforms all ciphers, that use RSA as their key exchange algorithm. The construction of ECDSA around elliptic curves and the use of shorter key length enables lower complexity, while achieving the same level of security as RSA. We conclude, that ECDHE for the key exchange is more energy efficient than Diffie Hellman (DHE). The energy consumption of cipher suite CS2 (ECDSA) on the server side is 27% smaller than CS1, 86% smaller than CS3 and 82% smaller than CS4.

On the server-side the total energy consumption can be reduced by 70% if ECDSA (CS2) is used instead of RSA. As for the client side, cipher CS1 with ECDHE-RSA consumed 26% less energy than CS2. The worst performing ciphers are ciphers which use DHE as their key exchange algorithm. An additional impact to the energy consumption of cipher suite CS3 is the usage of AES in CCM mode. The performance of cipher suites CS3 and CS4 was consistently worse.

This thesis can be used as a stepping stone for future work in this area. It would be of interest to evaluate a real-world network and determine which cipher suites are the most used to establish the encryption with TLS and use the work of this thesis to calculate the usage of energy in a real-world scenario. In addition, further analysis should include the measurement of power consumption of each step of the TLS connection workflow. It would be of interest to evaluate a more overall measurement of different security protocols and analyze further, what it would mean to reduce the power consumption of more than one security protocol.

These future development options show that there is still a long way to go, but the presented environment and the achieved results can be seen as another stepping stone in the evaluation of power consumption of security protocols. This work should trigger further research into this topic due to ever-growing relevance of power consumption and associated economical and ecological costs within IT.

List of Figures

List of Tables

Bibliography

[1] W. W. W. C. W3C, "The original http as defined in 1991," available at http://www.w3.org/Protocols/HTTP/AsImplemented.html. [Online; accessed 2020-04-01]. [Online]. Available: http://www.w3.org/Protocols/HTTP/AsImplemented.html

[2] J. Kohout, T. Komarek, P. cech, J. Bodnar, and J. Lokoc, "Learning communication patterns for malware discovery in https data," *Expert Systems with Applications*, vol. 101, 02 2018.

[3] G. Ouvrier, M. Laterman, M. Arlitt, and N. Carlsson, "Characterizing the https trust landscape: A passive view from the edge," *IEEE Communications Magazine*, vol. 55, no. 7, pp. 36–42, 2017.

[4] N. W. Group, "The transport layer security (tls) protocol version 1.2," available at https://tools.ietf.org/html/rfc5246. [Online; accessed 2020-04-01]. [Online]. Available: https://tools.ietf.org/html/rfc5246

[5] H. Lee, T. Malkin, and E. Nahum, "Cryptographic strength of ssl/tls servers: Current and recent practices," 01 2007, pp. 83–92.

[6] Q-Success, "Web technology surveys," available at https://w3techs.com/. [Online; accessed 2022-03-27]. [Online]. Available: https://w3techs.com/

[7] S. A. Chowdhury, V. Sapra, and A. Hindle, "Client-side energy efficiency of http/2 for web and mobile app developers," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1, 2016, pp. 529–540.

[8] G. Apostolopoulos, G. J. V. Peris, and D. Saha, "Transport layer security: how much does it really cost?" *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, pp. 717–725vol.2, 1999.

[9] I. Ayala, M. Pinilla, and L. Fuentes, "An energy efficiency study of web-based communication in android phones," *Scientific Programming*, vol. 2019, pp. 1–19, 04 2019.

[10] P. S. O'Brien, S. W. H. Young, K. Arlitsch, and K. Benedict, "Protecting privacy on the web: A study of https and google analytics implementation in academic library websites," *Online Information Review*, vol. 42, pp. 734–751, 2018.

[11] A. Ranjan, V. Kumar, and M. Hussain, "Security analysis of tls authentication," 11 2014.

[12] A. R. Ralph Holz, Johanna Amann and N. Vallina-Rodriguez, "The era of TLS 1.3: Measuring deployment and use with active and passive methods," *CoRR*, vol. abs/1907.12762, 2019. [Online]. Available: http://arxiv.org/abs/1907.12762

[13] N. I. of Standards and Technology, *Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations*. Independently published, 2019.

[14] N. Pohlmann, *Cyber-Sicherheit. Das Lehrbuch für Konzepte, Prinzipien, Mechanismen, Architekturen und Eigenschaften von Cyber-Sicherheitssystemen in der Digitalisierung*, 2019.

[15] M. Suárez-Albela, P. Fraga-Lamas, L. Castedo, and T. Fernández-Caramés, "Clock frequency impact on the performance of high-security cryptographic cipher suites for energy-efficient resource-constrained iot devices," *Sensors*, vol. 19, p. 15, 12 2018.

[16] R. A. Nofal, N. Tran, C. Garcia, Y. Liu, and B. Dezfouli, "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*, ser. MSWIM '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 61–70. [Online]. Available: https://doi.org/10.1145/3345768.3355924

[17] M. Suárez-Albela, T. M. Fernández-Caramés, P. Fraga-Lamas, and L. Castedo, "A practical evaluation of a high-security energy-efficient gateway for iot fog computing applications," *Sensors (Basel, Switzerland)*, vol. 17, 2017.

[18] P. Miranda, M. Siekkinen, and H. Waris, "Tls and energy consumption on a mobile device: A measurement study," *2011 IEEE Symposium on Computers and Communications (ISCC)*, pp. 983–989, 2011.

[19] I. Grigorik, *High Performance Browser Networking*. Shroff Publishers & Distr, 2014. [Online]. Available: https://books.google.at/books?id=IANHAQAACAAJ

[20] C. Coarfa, P. Druschel, and D. S. Wallach, "Performance analysis of tls web servers," *ACM Trans. Comput. Syst.*, vol. 24, no. 1, p. 39–69, Feb. 2006. [Online]. Available: https://doi.org/10.1145/1124153.1124155

[21] D. Berbecaru, "On measuring ssl-based secure data transfer with handheld devices," in *2005 2nd International Symposium on Wireless Communication Systems*, 2005, pp. 409–413.

[22] D. E. Simos, R. Kuhn, Y. Lei, and R. Kacker, "Combinatorial security testing course," in *Proceedings of the 5th Annual Symposium and Bootcamp on Hot Topics in the Science of Security*, ser. HoTSoS '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3190619.3190643

[23] Y. Mao, *Detailed Power Measurement with Arm Embedded Boards*. University of Maine, 2018. [Online]. Available: https://books.google.at/books?id=tqnKvQEACAAJ

[24] C. Chang, D. J. Nagel, and S. Muftic, "Assessment of energy consumption in wireless sensor networks: A case study for security algorithms," in *2007 IEEE International Conference on Mobile Adhoc and Sensor Systems*, 2007, pp. 1–6.

[25] P. Fuchs, J. Hribik, M. Hruškovic, B. Lojko, and R. Michálek, "Digital power and energy measurement," 01 2014.

[26] K. Aggarwal, C. Zhang, H. V. Campbell, A. Hindle, and E. Stroulia, "The power of system call traces: predicting the software energy consumption impact of changes," in *CASCON*, 2014.

[27] N. R. Potlapally, S. Ravi, A. Raghunathan, and N. K. Jha, "A study of the energy consumption characteristics of cryptographic algorithms and security protocols," *IEEE Transactions on Mobile Computing*, vol. 5, no. 2, pp. 128–143, 2006.

[28] D. Naylor, A. Finamore, I. Leontiadis, Y. Grunenberger, M. Mellia, M. M. Munafò, K. Papagiannaki, and P. Steenkiste, "The cost of the "s" in HTTPS," in *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies, CoNEXT 2014, Sydney, Australia, December 2-5, 2014*, A. Seneviratne, C. Diot, J. Kurose, A. Chaintreau, and L. Rizzo, Eds. ACM, 2014, pp. 133–140. [Online]. Available: https://doi.org/10.1145/2674005.2674991

[29] A. Gerez, K. Kamaraj, R. Nofal, Y. Liu, and B. Dezfouli, "Energy and processing demand analysis of tls protocol in internet of things applications," 10 2018, pp. 312–317.

[30] H. Bernstein, *Messen mit dem Oszilloskop: Praxisnahes Lernen mit einem PC-Simulationsprogramm*. Springer Fachmedien Wiesbaden, 2016. [Online]. Available: https://books.google.at/books?id=hIfJDQAAQBAJ

[31] D. Laverty, R. Best, P. Brogan, I. Al Khatib, L. Vanfretti, and D. Morrow, "The openpmu platform for open-source phasor measurements," *Instrumentation and Measurement, IEEE Transactions on*, vol. 62, pp. 701–709, 04 2013.

[32] I. Arbiter Systems®, "Precision gps timing and power measurement solutions," available at https://www.arbiter.com/. [Online; accessed 2020-04-01]. [Online]. Available: https://www.arbiter.com/

[33] D. L. et.al., "Open pmu project," available at https://sites.google.com/site/openpmu/pmu-fundamentals. [Online; accessed 2020-04-01]. [Online]. Available: https://sites.google.com/site/openpmu/pmu-fundamentals

[34] A. G. Phadke and T. Bi, "Phasor measurement units, wams, and their applications in protection and control of power systems," *Journal of Modern Power Systems and Clean Energy*, vol. 6, no. 4, pp. 619–629, 2018.

[35] C. Poole and I. Darwazeh, *Microwave Active Circuit Analysis and Design*, 11 2015.

[36] S. Grimnes and O. Martinsen, *Instrumentation and Measurements*, 12 2015, pp. 255–328.

[37] E. Products, "004656578 ena3," available at https://www.etigroup.eu/levels-2?view=ident&levelid=337&id=004656578. [Online; accessed 2020-04-01]. [Online]. Available: https://www.etigroup.eu/levels-2?view=ident&levelid=337&id=004656578

[38] D. Lancaster, *TTL Cookbook*, ser. Developer's Library. H. W. Sams, 1974.

[39] O. S. Project, "Openenergymonitor," available at https://openenergymonitor.org/. [Online; accessed 2020-04-01]. [Online]. Available: https://openenergymonitor.org/

[40] E. O. S. Project, "20a max clip-on current sensor ct," available at https://shop.openenergymonitor.com/20a-max-clip-on-current-sensor-ct/. [Online; accessed 2020-11-26]. [Online]. Available: https://shop.openenergymonitor.com/20a-max-clip-on-current-sensor-ct/

[41] w3tech, "Usage statistics of web servers," available at https://w3techs.com/technologies/overview/web_server. [Online; accessed 2020-04-01]. [Online]. Available: https://w3techs.com/technologies/overview/web_server

[42] O. S. Foundation, "Openssl cryptography and ssl/tls toolkit," available at https://www.openssl.org/. [Online; accessed 2020-04-01]. [Online]. Available: https://www.openssl.org/

[43] C. Site, "Rsa security llc," available at https://www.dnb.com/business-directory/company-profiles.rsa_security_llc.eb8af400e5cb5048394f0bb8edd01d66.html. [Online; accessed 2020-12-06]. [Online]. Available: https://www.dnb.com/business-directory/company-profiles.rsa_security_llc.eb8af400e5cb5048394f0bb8edd01d66.html