TLS in IoT Environment

First A. Author, *Fellow, IEEE*, Second B. Author, and Third C. Author Jr., *Member, IEEE*

[[1]](#footnote-1)

***Abstract*—One of the most widely used communication security protocols worldwide is Transport Layer Security (TLS). Its fundamental objective is to offer a secure communication channel with authentication, confidentiality, integrity, and perfect forward secrecy security features (PFS). One of the many algorithms that are now available can be used to implement each security service. TLS is too computationally intensive for many Internet of Things (IoT) devices and was not intended for the limited context. It is a flexible protocol, though, and each security function may be set or disabled for a specific connection. The amount of computing resources used is decreased by skipping a security service or implementing it using a less expensive approach. A TLS configuration specifies a connection's security features. With the resource-constrained IoT devices, several of those setups are applicable. The emphasis of current study is Datagram TLS (DTLS), which is either bound to a particular protocol or necessitates the involvement of a third party entity. This makes it difficult to integrate with already-in-use installations. In this study, we thoroughly assess the TLS protocol and its security features. We provide a framework that software developers and security experts may use to determine the least expensive TLS configuration for the requirements and constraints of their environment. Utilizing two cost metrics—estimated CPU cycles from valgrind and execution time from PAPI—we assessed the TLS implementation of the mbedTLS library. We'll conclude by demonstrating how closely the estimated numbers match the actual ones.**

***Index Terms*—TLS, DTLS, SSL, IoT, Embedded Systems**

# I. INTRODUCTION

T

he number of IoT devices has sharply increased over the past several years, and this trend is anticipated to continue. The Internet of Things (IoT) is a network of interconnected devices that communicate with one another by exchanging data online. In actuality, it might be any device having a certain IP address and the capacity to send and receive data across a network. IoT devices come in a variety of shapes and sizes, but they are all constrained by the amount of memory, computing power, and energy they can use. However, this does not imply that these systems can only execute the easiest algorithms. IoT includes a variety of hardware configurations and many device types. Others have the means to employ public key cryptography, yet for some of them symmetric encryption is the only practical alternative. Temperature sensors, smart lightbulbs, and activity trackers are a few examples of IoT gadgets.

Even though inter-device communication has many advantages, it's crucial to make sure it's secure. For instance, you don't want other people to be able to see your password when you connect into your online banking account because this might result in the compromising of your account. If your account is compromised, your money might be taken by a malevolent party. Despite all the advantages that IoT technology offers, communication security is sometimes neglected and treated as an afterthought.

One of the most used protocols for securing communication is TLS. Many technologies, including Hypertext Transfer Protocol Secure, are powered by it (HTTPS) [1]. TLS provides authentication, confidentiality, privacy, integrity, replay protection, and perfect forward secrecy as security services. Every TLS connection does not have to use each of those services. The protocol resembles a framework in that you may enable certain security services for each connection, much as in a framework. Refusing superfluous services will result in lower resource utilisation, which will reduce execution time and power consumption. Due to the limited capabilities of the devices, this is particularly crucial in the context of the Internet of Things. While a device connects with an external service, for instance, confidentiality, integrity, and authentication are necessary, but the first security attribute is not as relevant when the device is downloading a firmware update. In the latter scenario, authentication and integrity would be sufficient.

Despite the fact that TLS was not created for the restricted IoT environment [2], It is a flexible protocol that may be tailored to a user's requirements. The protocol provides a number of security services, and a variety of methods can be applied to perform each of those services. The usage of TLS in the context of IoT is conceivable if such algorithms are correctly chosen.

Existing research either focuses on Datagram TLS (DTLS) optimization and only a portion of it can be used to TLS or calls for modifications to the fundamental structure of TLS, including the addition of third-party entities. Here, we wish to investigate TLS optimization in more detail. It is obvious that such a thing is necessary, particularly in light of the ongoing development of the TLS standard and the Constrained Application Protocol (CoAP) over TCP [3]. The standard does not examine TLS improvements, and since they would be advantageous for any IoT device utilising it in the future, this is a crucial subject to investigate.

The purpose of this work is to give a method for helping application developers who want to integrate secure communications in their applications to balance security and resource utilisation based on the requirements and constraints of the environment. We want to give a broad overview of the expenses associated with the TLS protocol overall and with each of its component pieces. This will make it possible to respond to queries like How much money would we save if we utilise authentication algorithm X rather than Y? Therefore, doing assessments on particular IoT devices or reviewing hardware-specific improvements falls outside the purview of this article.

A thorough cost analysis of TLS is required in order for us to succeed. The programmer will be able to select a configuration that satisfies his security needs and device limitations with the help of this information. The programmer may opt for an alternate configuration, maybe with a loss of some security services and a lower security level, or forego utilising (D)TLS completely if the hardware restrictions of the device do not enable it to satisfy the criteria. Therefore, this work is intended for developers and infosec specialists who want to enhance communication security in IoT systems.

In our study, we thoroughly assessed the implementation costs of TLS 1:2 in mbedTLS 2.7.0. One of the most well-liked TLS implementation libraries for embedded devices is mbedTLS. We assessed expenditures in terms of the anticipated time and CPU cycle use. The processor's registers were used to read the time values directly. By contrasting the estimates with time measurements received directly from the CPU registers, we will demonstrate in our research that the estimations do indeed reflect genuine values. At four distinct security levels, we thoroughly examined each and every one of the 161 TLS configurations offered by mbedTLS 2.7.0.

Two components make up a TLS connection: In the Handshake phase, the peers first create a secure communication channel; the Record phase then involves data exchange over that channel [4]. For two key reasons, we concentrated on the handshake portion of the protocol. Due of the intricate combinations of several alternative algorithms, it is the component with the most cost fluctuation. Second, it is the area that has received the least attention from previous research. Hash functions and symmetric encryption techniques are the major components of the Record phase. Existing work has already done a detailed analysis of their expenses.

We analysed the costs of the symmetric encryption techniques and hash functions even though our focus was on the Handshake. In order for the costs of the Record phase to be equal to the costs of the Handshake phase, we evaluated how much data must be shared between peers. We came at the conclusion that this range is between 560KB and 1:62MB for the client and between 830KB and 1:27MB for the server for the most often used setups on the internet. Because a Handshake will be performed by the device for every new connection, it only makes sense to concentrate on Handshake cost minimization if little data is being sent back and forth.

We produced numerous changes to the TLS 1:3 specification while working on this dissertation, and we were publicly acknowledged as contributors [5]. The paper defining TLS 1:3 has the author's name for this dissertation [6]. We have also contributed to DTLS 1:3 standard, but to a lesser level [7]. We discovered a security flaw and a standard non-conformity in the mbedTLS library's TLS implementation. We reported it, and a Common Vulnerabilities and Exposures (CVE) with the identifier CVE-2018-1000520 has been given to it [8]. This flaw affects the authentication portion of the TLS protocol, wherein certificates issued using the wrong algorithm were occasionally accepted. To be more exact, only Rivest-Shamir-Adleman (RSA)-signed certificates were permitted by ECDH(E)-RSA cipher-suites, not Elliptic Curve Digital Signature Algorithm (ECDSA)-signed ones. Additionally, we discovered a flaw in the mbedTLS test suite involving the usage of outdated SHA-1-signed certificates and provided a code patch for it [9] [10]. By enabling the automated collecting and analysis of metrics across a range of hardware and settings, we have also created a comprehensive set of tooling that can be used to further investigate the costs of TLS.

II. Related Work

The benefit of TLS [11] for end-to-end security in the Internet of Things is covered in this section. TLS was created to give end-to-end communication secrecy and integrity. Due to its widespread use in security solutions, IPSEC is also a noteworthy end-to-end security protocol. IPSEC can provide network node secrecy by depending on key exchange or pre-shared keys. By using the "Authentication Header," which offers end-to-end security comparable to TLS, it may also be used as an authentication protocol. In fact, this is the default technique in IPv6.

IPSEC is a great protection mechanism that may offer the same services as TLS or supplement other protocols to give superior security, although being restricted in practise to the creation of tunnels and Virtual Private Networks (VPN). Although end-to-end security is a criteria for secure communications, it is not the only one that must be met. The end-to-end security offered by TLS, according to Behringer, is often thought to be sufficient, but network security is equally important, necessitating the use of IPSEC and other protocols like it. Beyond end-to-end security, there are a number of duties that should be taken into account, including limiting harmful endpoint activity, monitoring and cryptographically isolating certain linkages, and IP spoofing prevention.

Some protocols, like Kerberos, don't offer authentication on their own; instead, they use SAML to convey permission choices made by other services. Additionally, applications must be created taking the protocol into account. Numerous other protocols that provide mutual authentication, such as TLS, do not completely protect participants' anonymity since they call for at least one of the parties to the transaction—typically the server—to reveal its identity.

This article discusses the present state of TLS and the PKI usage associated with it for IoT device access to internet services while being aware that there are other additional protocols that provide authentication in dispersed contexts and even provide complete anonymity. TLS is being analysed since it is the only end-to-end protocol that can be deemed to be universally approved. Because TLS normally calls for endpoint authentication, server privacy is not taken into account. Given that many of the connections in IoT contexts will be opportunistic, such those for service discovery or name resolution, which are required for accessing local computing resources, TLS is a great tool for creating secure connections. Although TLS requires PKI certificates to be handled by IoT devices, which might be resource-intensive, it offers good adaptability when used with PKI. TLS offers authentication, secrecy, and permits negotiation of practically all security parameters. Using user space libraries, the client application and the service may both actively take part in the negotiation. These and other factors have led to the adoption of TLS by a number of transport protocols.

Due to the fact that TLS was initially intended to operate on top of a TCP/IP stack, it is connection centric. TLS has been supplemented by variants that operate over SCTP and UDP while utilising the same security negotiation method. In addition, the TLS standard outlines a means for supporting additional functions. The ideal method for including additional capabilities that the protocol did not initially take into account is to use TLS extensions. Extensions enhance the negotiating capabilities of the protocol while maintaining backward compatibility by adding further information to the handshake messages. TLS endpoints disregard extensions they are unable to grasp in order to do that.

The bulk of the research in the field suggests solutions that are either dependent on a particular protocol, like CoAP, or call for the involvement of a third party, like the trust anchor in the case of the S3K system, or even both. [11]. This has two major problems. First off, it is difficult to apply a protocol-specific solution in a situation where that protocol does not support (D)TLS. Second, the need for a third party adds extra expense and complexity, which will be a major barrier to the technology's uptake. This is especially true for engineers working on personal or small-business projects, which, in the worst situation, makes communications unsafe.

DTLS is the subject of the study that does not depend on any one protocol or call for the involvement of a third party. Studying TLS optimization is crucial given the ongoing development of standards like CoAP over TCP and TLS [3]. The phrase "HTTP protocol for restricted devices" is frequently used to describe CoAP [12]. Since TLS improvements will be advantageous for every IoT device implementing CoAP over TCP and TLS in the future, this is a crucial topic to investigate [13].

# III. The TLS Protocol

TCP or another connection-oriented and dependable transport protocol, such as SSL, forms the foundation for the client-server protocol TLS [14]. Its fundamental objective is to maintain the two communicating peers' integrity and secrecy. While integrity indicates that a third party cannot change the data, confidentiality implies that a third party cannot access the data.

The TLS protocol is used by client-server applications to interact over networks in a way that guards against listening in and tampering. Applications can interact with each other with or without TLS (or SSL), therefore the client must ask the server to establish a TLS connection [15]. Using a separate port number for TLS sessions is one of the key strategies for doing this. While port 443 is frequently used for encrypted HTTPS transmission, port 80 is commonly utilised for unencrypted HTTP traffic. Another method is to use a protocol specific STARTTLS request to the server, such as when utilizing the mail or news protocols, to convert the connection to TLS.

Following their agreement to utilize TLS, the client and server handshakingly negotiate a stateful connection. The protocols create cypher parameters and a session-specific shared key during a handshake using an asymmetric cypher, after which all subsequent communication is encrypted with a symmetric cypher. The client and server agree on a number of parameters during this handshake that are used to establish the security of the connection: When a client requests a secure connection from a TLS-enabled server and offers a list of available cypher suites, the handshake starts (ciphers and hash functions). The server selects an encryption and hash function from this list that it also supports, notifying the client of its choice. A digital certificate is often then provided as identification by the server. The certificate includes the server name, the trusted certificate authority (CA) that attests to the certificate's authenticity, and the public encryption key for the server.

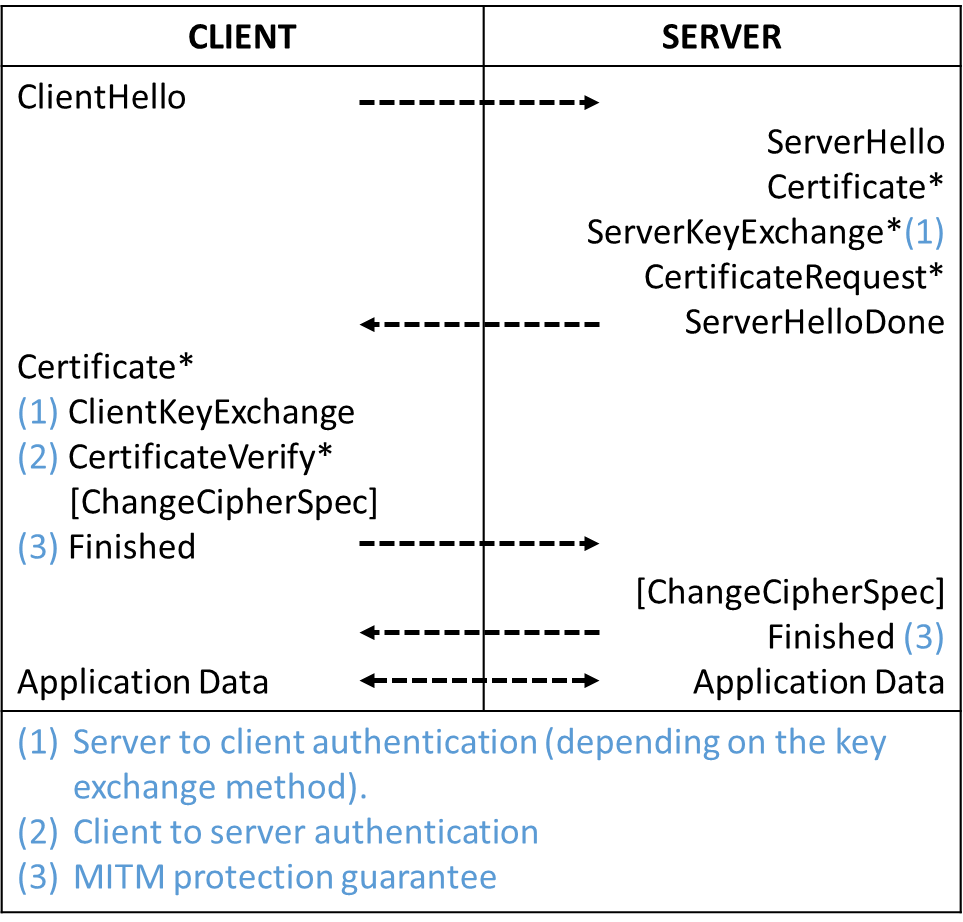
Before continuing, the client verifies the certificate's authenticity. The client either uses a random number (PreMasterSecret) to encrypt with the server's public key and send the result to the server (which only the server should be able to decrypt with its private key); both parties then use the random number to generate a unique session key for later encrypting and decrypting data during the session; or the client uses Diffie-Hellman key exchange to securely generate a random and unique session key.

The secured connection, which is encrypted and decrypted using the session key until the connection shuts, starts when the handshake is over. The TLS handshake will fail and the connection won't be established if any of the aforementioned stages are unsuccessful.

The OSI model and the TCP/IP paradigm's many layers do not all cleanly suit TLS and SSL. [8] TLS is supposedly above the transport layer since it operates "on top of some dependable transport protocol (e.g., TCP)". The presentation layer often performs the role of delivering encryption to upper levels. Although applications employing TLS must actively control starting TLS handshakes and managing exchanged authentication certificates, apps often utilise TLS as if it were a transport layer.

Connections between a client (like a web browser) and a server (like wikipedia.org) that are protected by TLS should have one or more of the following characteristics: The connection is private (or secure) because the transmitted data is encrypted using a symmetric-key technique. Based on a shared secret that was negotiated at the beginning of the session, the keys for this symmetric encryption are produced uniquely for each connection. Before the first byte of data is transferred, the server and client discuss the specifics of which encryption technique and cryptographic keys to employ (see below). The negotiation of a shared secret is dependable and secure (the negotiated secret cannot be accessed by an attacker who inserts himself in the midst of the connection or by eavesdroppers) (no attacker can modify the communications during the negotiation without being detected).

Public-key cryptography allows the communication between parties to be authenticated. For the server, this authentication is necessary; however, for the client, it is not. Because every message sent contains a message integrity check utilising a message authentication code to avoid undetected data loss or tampering during transmission, the connection is dependable. In addition to the aforementioned, careful TLS configuration can offer other privacy-related qualities including forward secrecy, which makes sure that any potential future key disclosure cannot be used to decrypt any previously recorded TLS interactions.



**Fig 1:** TLS 1:2 message flow for a full handshake

TLS has undergone multiple revisions to address security threats that have been developed in an effort to undermine some features of the communications security that it aims to provide. After possible security flaws were found, web browser developers often updated their software to address them (see the history of TLS/SSL support in web browsers).

TLS is positioned between the Transport and Application layers of the TCP/IP protocol stack. From the perspective of the application developer, it is made to make secure communications more convenient to set up and use. It is now just necessary for the developer to establish a "secure" connection (a socket), as opposed to a "regular" one.

There are two parts to a secure communication formed with TLS. The conversing peers negotiate the settings, such as the secret keys and the encryption technique, during the initial phase of communication. They communicate data that is cryptographically secured in the second phase while adhering to the terms that were previously agreed upon. The Handshake Protocol is used for the first phase, while the Record Protocol is used for the second. The client and the server exchange a variety of messages during the Handshake Protocol in order to accomplish their objectives. This message flow, which is shown in Fig 1, shows messages that rely on the context and are not always sent. The following security services are offered by TLS:

* Peer entity authentication ensures that a peer is speaking to a particular entity, such as www.google.com. Data origin (or integrity) authentication ensures the integrity of the data. This is accomplished by employing a Pre-Shared Key in conjunction with Asymmetrical Cryptography (AC), sometimes referred to as Public Key Cryptography (PKC), (e.g., RSA and DSA), or Symmetric Key Cryptography (PSK).
* Confidentiality: The information sent between the client and the server during communication is encrypted. Data encryption employs symmetric cryptography (e.g., AES).
* A peer may be certain that the data was not altered or fabricated thanks to integrity, also known as data origin authentication, which provides an assurance that the data is coming from the anticipated source. A peer may be certain, for instance, that the index.html file it received while connecting to www.google.com came from www.google.com and was not altered by an attacker (data integrity). Either a keyed Message Authentication Code (MAC) or an Authenticated Encryption With Associated Data (AEAD) cypher is used to accomplish this.
* A peer may be certain that a message has not been repeated thanks to replay protection, also known as freshness. The use of sequence numbers enables this. The sequence number for each TLS record is unique and continually increased. The sequence number is a direct input of the MAC function if a non-AEAD encryption is employed. A nonce produced from the sequence number is used as the cipher's input if an AEAD cypher is employed.
* Perfect forward secrecy (PFS) ensures that even if the long-term secret is disclosed, the confidentiality of previous exchanges is maintained.

Not every circumstance calls for the employment of every security service. TLS is similar to a framework that enables choosing which security services should be utilised for a communication session in this regard. For instance, the authenticity promise could not be fulfilled if certificate validation is ignored.

A connection's array of security services will vary depending on the TLS configuration being used. The key exchange procedure and the symmetric algorithm/hash function pair are specified by a TLS configuration. The security services that will be utilised in the connection are specified by the key exchange mechanism, together with the algorithms that will be employed to provide those services. The phrases "TLS configuration" and "cipher-suite" are used synonymously throughout this article. A connection will be created using the ECDHE-ECDSA key exchange and the combination of AES in CBC mode with 256-bit keys and the SHA-384 hash function will be utilised, for instance, if the TLS ECDHE ECDSA WITH AES 256 CBC SHA384 cipher-suite is used. ECDHE-ECDSA key exchange suggests that PFS will be provided by the Elliptic Curve Diffie-Hellman Ephemeral (ECDHE) technique and authentication by the ECDSA scheme.

# IV. Results and Data Analysis

The cost analysis of TLS and associated security services is presented in this section. Before presenting the time metrics collected using Performance Application Programming Interface, we will first analyse the estimated number of CPU cycles obtained with vallgrind/callgrind (PAPI). The estimations accurately represent the true values, as shown by a comparison of the two sets of findings.

1. *Methodology*

The expected number of CPU cycles and the time consumed served as our two cost indicators. Valgrind and callgrind were combined to get the approximate amount of CPU cycles. We utilised PAPI, which gets the time data straight from the processor's registers, to compute the time values. We calculated the estimated number of CPU cycles (CEst) using the kcachegrind formula: CEst is calculated using the formula Ir + 10 \* Bm + 10 \* L1m + 100 \* L2m, where Ir represents the total number of instruction fetches, Bm represents the total number of incorrectly anticipated branches, and L1 and L2 represent the total number of L1 and L2 cache misses, respectively.

With PAPI, we disabled Intel Turbo Boost and set the processor's speed to the lowest possible frequency of 800Mhz in order to maintain the metrics consistently and approximate the conditions to those of an IoT environment. We profiled the virtual time elapsed, which is the CPU time required to execute the process in its entirety and excludes time slices consumed by other processes or blocked time (such as waiting for I/O).

We conducted the assessments at four distinct degrees of security: low, regular, high, and extremely high. We used 1024 bit RSA/Diffie-Hellman (DH)/Digital Signature Algorithm (DSA) keys at the low security level and 256 bit Elliptic Curve Cryptography (ECC) keys at the normal, high, and very high security levels, respectively. At the low security level, those keys were 2048 bit and 224 bit. At the high security level, they were 4096 bit and 384 bit. At the very high security level, they were 8192 bit and 512. The most popular configuration of the internet at the time this paper was prepared served as our foundation for the typical security level. Only the client authenticated themselves to the server on each instance, and depending on the cipher-suite, the server's certificate was signed using a 2048-bit RSA key or a 256-bit ECC key.

To automate the procedures of metric collecting and analysis, we created tools. We established a client-server connection using each of the TLS setups during the metric collection phase. With our second piece of gear, the output metrics from the Handshake and Record phases were saved to disc for post-analysis.

The Intel(R) Core(TM) i7-4700HQ CPU @ 2.40GHz processor was used for all experiments. We didn't get any data about standard IoT processors. Despite this, the numbers offered are still important. The measurements would keep a comparable percentage if they were gathered on an IoT processor, hence the results and analysis offered would still be valid. Additionally, it is feasible to utilise callgrind on an IoT processor to gather data for more precise and device-specific CPU cycle estimation (either manually or automatically using our tools). For PAPI, the same is true.

1. *Authentication Costs In TLS*

There are two methods for doing authentication in TLS: either a PSK or asymmetric cryptography. There are two options for the algorithm when asymmetric cryptography is used: RSA or ECDSA. With PSK authentication, the secret used for mutual authentication is already in the possession of both peers. The Pseudo-Random Function (PRF) then uses this secret as an input to create the keying material. If the finished message's integrity check is successful, mutual authentication is achieved. This is only feasible if the PSKs used as inputs to the PRF by the peers were identical, which can only occur if they did. As a result, without taking any explicit authentication steps, the client and server authenticate one another. Because of this, we may assume that PSK authentication has a cost of 0.

The characteristics of RSA and ECDSA differ, and their prices change depending on the security level being used. The price is displayed as the projected number of CPU cycles, with M denoting millions of cycles. These two graphs' comparison indicates various variations between the two methods. First off, the cost rise for ECDSA is logarithmic as opposed to exponential for RSA. This is a result of the mathematical processes that each algorithm's foundational algorithm uses. For ECDSA, this process entails multiplying a scalar by a point on the elliptic curve, but for RSA it entails modular exponentiation. The second distinction is that ECDSA is less expensive for creating new signatures than RSA is for signature verification across all security levels. The third and last distinction is that, while the overall cost (i.e., the sum of the costs associated with creating and validating signatures) is cheaper for RSA at the low and standard security levels, it is lower for ECDSA at the high and very high security levels.

The appropriate algorithm to apply will depend on the circumstances, and the solution to this issue is not simple. For instance, RSA would be the least expensive option in the case of a confined client and an unconstrained server. However, if the server is the limited node, then ECDSA would be the least expensive method. The least expensive option for the low and regular security levels would be RSA, whereas ECDSA would be used for the other levels if both nodes were limited. ECDSA is the method to utilise if the goal is to divide the costs among the peers as evenly as feasible.

We will now examine the costs of this security service for each of the key exchange techniques for the client and the server after examining the prices of the authentication algorithms that can be utilised. As we've already shown, the authentication cost for PSK cipher-suites is zero for both peers. The ECDSA algorithm is used by ECDSA-based cipher-suites to generate and validate signatures. RSA is also used for those reasons by cypher suites that are RSA-based. It is difficult to discuss the cost of authentication in TLS without mentioning PFS. If a PFS-enabled cipher-suite is utilised, the Server Key Exchange message is an extra piece of data that is authenticated in all non-PSK cipher-suites. The public (EC)DH parameters have been hashed and signed in this message. This affects both the client's and the server's costs associated with creating and verifying signatures. That additional fee applies to any non-PSK key exchange techniques that start with either ECDHE or DHE. This explains why ECDHE cipher-suites are more expensive on the client side than ECDH ones.



**Fig 2:** normal security level decision tree for the cheapest key exchange

It is clear from the graphs that various key exchange techniques may be categorised according to authentication costs. These groups are: RSA-PSK, ECDH-RSA, ECDHE-RSA, DHE-RSA, RSA, RSA-PSK, ECHD-ECDSA, and ECHDE-ECDSA for the client. For the server, those groups are: 1 - PSK, ECDHE-PSK, DHE-PSK, ECDH-RSA, ECDH-ECDSA, 2 - ECHDE-ECDSA, 3 - ECDHE- RSA, DHE-RSA and 4 - RSA, RSA-PSK. The cost of authentication is the same within of any group, regardless of security level. Group numbers are arranged in decreasing order of expense, starting with Group 1, followed by Group 2, and so on. All key exchange techniques belonging to the same group conduct the same set of activities in order to give authentication.

1. *PFS Costs In TLS*

Using either the DH algorithm or its ECC cousin, the Elliptic Curve Diffie-Hellman, is one of two ways to achieve PFS in TLS (ECDH). The same fundamental tasks are carried out sequentially by each peer in both algorithms: creating a public/private keypair, exchanging the public values, and determining the shared secret.

Each peer in ECDH and DH performs two fundamental operations: first, they create a public/private ECC keypair, and then they create the shared secret. A 2D (x; y) coordinate on the curve will be the ECDH shared secret as a consequence. TLS discards the value of y and uses x as the premaster secret instead. The shared secret that results in DH is a scalar, which serves as the premaster secret.

Since the private key in ECDH and DH operations is merely a randomly generated integer, computing it is inexpensive. The calculation of the public key and shared secret is the most expensive since it calls for multiplying a scalar by a point on the elliptic curve for each. The cost of calculating the shared secret and the public key for ECDH and DH is relatively comparable.

One significant distinction between ECDH and DH is that, as opposed to the latter, which experiences exponential cost growth. This is a result of the mathematical procedures that underlie each algorithm. The least expensive solution depends not just on the security level but also on whether we were optimising for the generation or verification of signatures in RSA and ECDSA because the cost of those processes varies. The overall cost for ECDH and DH is roughly equally split between keypair creation and shared secret generation. As a result, we may decide by just comparing the Total cost. When using a low security level, DH is the least expensive option; when using a regular or higher security level, ECDH is. The formulae and forms of the trendlines also show the logarithmic and exponential characteristics of ECDH and DH, respectively.

After examining the prices of the PFS-capable algorithms, we will now examine the costs of this security service for each cipher-suite utilised by the client and the server. Since both employ the ECDH method, the ECDH key exchange is nonetheless closely tied to the ECDHE one even if it does not provide PFS. In our assessed situation, where only the server authenticates to the client, the increased costs for ECDH cipher-suites are considerable, and there is no cost difference between ECDHE and ECDH cipher-suites for the client.

1. *TLS Handshake Costs*

After examining the prices of TLS's authentication and PFS security services, we will now talk about the Handshake's overall cost. According to study, there are ten different key exchange techniques, some of which have the same prices. We refer to two or more key exchange techniques as belonging to the same cost category if their costs are comparable. If a single key exchange method's costs differ considerably from those of the others, a cost group may also contain it. On the client, there are six cost groups that can be found: PSK, RSA, RSA-PSK, ECDHE-PSK, ECDHE-RSA, ECDH-RSA, ECDH-ECDSA, DHE-PSK, and DHE-RSA. On the server, there are seven cost groups that can be found: PSK, ECDH-RSA, ECDH-ECDSA, ECHDE-PSK, ECDHE-ECDSA, RSA

The categories are shown in order of increasing expense. The PSK key exchange is by far the least expensive option for both the client and the server. The peer and security level determine which key exchange technique is the most costly. DHE-RSA, which is true beginning from a regular security level for the client and a low security level for the server, would be the solution if we had to pick one alternative to hold the title of most costly. Similar trends are observed with DHE-PSK, notably for the customer. Once more, the benefit of ECC is seen in how much less expensive key exchanges using ECDH(E) and/or ECDSA are than those using DHE and RSA. This results from a logarithmic cost rise for ECC vs an exponential cost increase for non-ECC.

Our investigation led us to construct a formula that divides the TLS Handshake's expenses into separate charges: HandshakeCost = AuthCost + PFSCost + AdditionalCosts + TLSOverhead. The price of the PSK key exchange is the TLS Overhead. The price of authentication is called the Auth Cost. The PFS cost is the PFS cost. The extra expenses incurred by the peers when producing and interpreting TLS messages that aren't part of the PSK Handshake are known as Additional Costs. All RSA and ECDSA cipher-suites include the Certificate message, which accounts for the majority of those costs. The client must parse the der-encoded certificate into internal fields while performing various checks along the way, such as the Not Valid Before/After fields, and the server must post the Certificate message to the Record layer.

It is not easy to choose the cheapest TLS setup for one's requirements and constraints. We created a set of decision trees that streamline this procedure as a result. The decision tree for the standard security level is shown in Fig 2. The key exchange techniques are laid out at the terminal nodes from least costly to most expensive. The cost variations among the cipher-suites that are part of the same cost group are negligibly tiny and may be disregarded. Cipher-suites are arranged to reduce client/server expenses if the decision is to optimise for the client/server. The cypher suite is arranged to reduce total costs, which is the sum of the expenses for the client and the server, if the decision is made to optimise for both.

1. *Confidentiality and Integrity Costs In TLS*

We will now examine the cost of the secrecy and integrity services after examining the costs of authentication, PFS, and the Handshake. Due to the fact that these services are provided together as a part of the cipher-suite in TLS, it is not necessary to examine them separately. The Record phase, which follows the Handshake phase in establishing a secure communication channel between two peers, is when data is sent while being protected by confidentiality and integrity guarantees (if any are relevant to the cipher-suite).

In mbedTLS 2.7.0, there are 26 different symmetric encryption algorithm/hash function pairings available. There are a total of four hashing algorithms: MD5, SHA1, SHA256, and SHA384. In terms of the projected number of CPU cycles, we profiled the expenses of each of the 26 distinct pairs. As anticipated, their price increases linearly as the number of encrypted bytes increases. AEAD encryption algorithms (AES with GCM and CCM modes) are the least expensive block cypher algorithms, according to the examination of the data acquired, whereas 3DES with SHA is the most expensive combination of an encryption algorithm with a MAC function. The cost of CAMELLIA algorithms is higher than that of AES.

The most expensive hash algorithm, according to an examination of its costs, is SHA-256, whereas the least expensive is MD5. It's crucial to keep in mind, though, that SHA-1 and MD5 are currently regarded as being unsafe and open to a variety of attacks. Because of this, the two hash algorithms that are recommended are SHA-256 and SHA-384, the latter of which is the least costly. The majority of the cost for both scenarios—AEAD methods and non-AEAD algorithms paired with a hash function—is associated with data encryption and decryption.

Only when there is little data being exchanged does it make sense to aggressively optimise the handshake. But what does a modest amount of data actually mean? We analysed the costs of data encryption using the AES-128-GCM (low and standard security levels) and AES-256-GCM (high and very high security levels) in order to respond to that issue. These algorithms were chosen because they were among the least expensive to offer the necessary security level and were recommended by browsers like Google Chrome 67. For certain techniques, the price of encryption and decryption is comparable. The following calculations, for the cost of encryption using AES-128-GCM and AES-256-GCM, respectively, are the results of our analysis: In both cases, NumCC is equal to 104 \* NumBytes + 22680 and 105 \* NumBytes + 22740 (R2 = 1). The amount of CPU cycles is NumCC, while the number of encrypted bytes is NumBytes. The formulae demonstrate that AES-256-GCM is somewhat more expensive than AES-128-GCM. This is to be expected given the first one's higher key size.

The obtained formulae can be used to determine whether the cost of executing the handshake equates to the cost of encrypting data. Less than 2 MB of data must be transferred in 85% of client situations in order for the Record phase costs to equal the Handshake costs. That ratio for the server is 72.5 percent. Currently, the most used TLS setups are ECDHE-ECDSA and ECDHE-RSA at the usual security level. The client only has to submit 560KB for the second key exchange and 1:62MB for the first key exchange for those cipher-suites. These figures are 830KB and 1; 27MB for the server, respectively.

1. *PAPI Time Measurements and Comparisons with Estimates*

The previous analysis was done with the estimated amount of CPU cycles acquired from valgrind / callgrind. We wanted to know how accurate their projections were. For this reason, we utilised papi to gather time metrics. The iron law of processor performance states that the time spent by a programme execution is proportional to the number of instructions (I), the average number of cycles per instruction (CPI ) and the amount of time per each processor cycle (CT): I x CPI x CT = CPUTime. It is possible to estimate the amount of CPU cycles as follows: CPUCycles = I x CPI. As a result, the programme execution time may be calculated using the formula CPUTime = CPUCycles \* CT, where CT is a constant. As a result, we anticipate that the time-based graphs will have lesser numbers and a comparable appearance. The outcomes from PAPI were in line with what we had anticipated.

Not only do the graphs produced by PAPI analysis for the Handshake, authentication, and PFS resemble those produced by valgrind estimates extremely closely (with the obvious exception of the y-axis numbers), but the ratio between each connected set of operations is also quite comparable. To ensure that all of the ratios are positive, we purposely divided the most expensive operation by the least expensive one. The outcomes are displayed using a logarithmic scale. The PAPI ratios and the valgrind ratios are quite comparable. Particularly for the ECDH, DH, and ECDSA, this is valid. For DH, there is no difference; for ECDH, valgrind's ratio is 0:96 percent bigger at the high and very high security levels; and for ECDSA, there is only a difference at the low security level, where valgrind's ratio is 9:2 percent larger than PAPI's. Operations that employ RSA show the highest difference in ratio, but as the security level rises, this disparity becomes less. In actuality, this is the overarching pattern in every instance. For the low security level, Valgrind's RSA has a sign/verify ratio that is 24:87 percent bigger than PAPI's, but only 3:6 percent larger for the very high security level. Following this pattern, the encrypt/decrypt ratio for RSA in callgrind is 19:41% greater than PAPI's for low security and 7:3% higher for strong security.

V. Conclusion

The examination of the expenses of the TLS protocol reported in this dissertation was the most thorough and in-depth one available to date. Software developers and security experts may utilise the data offered here to make educated judgments regarding the security/cost trade-offs that are unique to the environment.

The expenses of TLS at the low, medium, and high levels were examined. We looked into and compared the prices of each method that makes it possible for TLS to provide security services at a low level. We came to the conclusion that the peer whose expenses we wished to decrease, as well as the key size, affected the choice of the cheapest method. We looked at the intermediate level security service costs for authentication and PFS. Regarding the cost of each security service in terms of CPU cycles and time, we provided an answer. On a broad scale, we examined the price of the handshake as a whole. Its price was broken down in relation to the price of the security services. As a consequence, we arrived with a formula that breaks down the TLS Handshake's expenses into their component elements.

Although the costs of the Handshake were the main focus of our work, we also considered the costs of the security services for secrecy and integrity. In order to determine when the costs of the Handshake equate with the costs of secrecy and integrity, the asymmetric encryption techniques were profiled. According to our study, less than 1:7MB of data must be sent between the peers in a normal internet arrangement for that to occur. Due to this, we came to the conclusion that it only made sense to extensively optimise the Handshake if there was little data being sent.

While working on this thesis, we examined TLS at a level that had never been done before and made a contribution to the international security community in the following ways:

* Contributing to the DTLS protocol version 1:3 specification, as well as, to a lesser extent, the TLS protocol version 1:3 specification.
* Discovering and reporting a security flaw in mbedTLS, which was given the CVE identification CVE-2018-1000520.

We utilised the TLS implementation of the mbedTLS library, one of the most well-liked TLS implementation libraries for embedded devices, for the evaluation. Two cost metrics were employed. First, we thoroughly examined TLS by counting the anticipated CPU cycles that callgrind provided. Then, by contrasting the estimates with time metrics received directly from the processor's registers, we demonstrated that the estimations are close to the actual values. The outcomes shown here were produced using a potent, contemporary computer. Despite this, they remain important when taking into account the expenses of limited IoT devices. Even while the costs will vary in absolute terms depending on the device, they will still be distributed similarly and follow a similar pattern. Additionally, any machine may be profiled using the established tools, providing device-specific cost data.

In our effort, we gathered and examined a sizable number of callgrind-derived metrics. Callgrind only provided estimates of the CPU cycles consumed, but we later demonstrated that these estimates accurately represented actual CPU cycle usage by contrasting them with PAPI's timing findings. It's crucial to keep in mind that those measurements were acquired using a general-purpose computer. The environment on an IoT device is still anticipated to be substantially different due to variables such a lower clock frequency, RAM, and cache capacity, despite the fact that we set the CPU frequency and deactivated various hardware optimizations. Consequently, gathering stats from an IoT device would be interesting.

Limited power is another trait of many IoT devices (e.g. using battery as a power source). It would thus be worthwhile to examine the cost of TLS in terms of power consumption. This would also make it possible to draw inferences, such as: Using the TLS configuration X would result in a Y-day reduction in device battery life.

# References

|  |  |
| --- | --- |
| [1] | C. Bayılmışa, M. A. Eblemeb, Ü. Çavuşoğluc, K. Küçükd and A. Sevin, "A survey on communication protocols and performance evaluations for Internet of Things," *Digital Communications and Networks,* 2022. |
| [2] | J. d. Hoz, J. Saldana, J. Fernández-Navajas, J. Ruiz-Mas, R. G. Rodríguez and F. d. J. M. Luna, "SSH as an Alternative to TLS in IoT Environments using HTTP," *2018 Global Internet of Things Summit (GIoTS),* 2018. |
| [3] | J.-H. Jung, M. Gohar and S.-J. Koh, "CoAP-Based Streaming Control for IoT Applications," *Electronics ,* vol. 9, no. 8, 2020. |
| [4] | S. Chen, S. Jero, M. Jagielski, A. Boldyreva and C. Nita-Rotaru, "Secure Communication Channel Establishment: TLS 1.3 (over TCP Fast Open) versus QUIC," *Journal of Cryptology,* 2021. |
| [5] | "TLS 1.3 Draft Specifications," [Online]. Available: https://github.com/tlswg/tls13-spec. [Accessed 06 Nov 2022]. |
| [6] | E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," *Internet Engineering Task Force (IETF), Request for Comments: 8446,* 2018. |
| [7] | "IETF TLS Working Group draft of [DTLS 1.3]," [Online]. Available: https://github.com/tlswg/dtls13-spec. [Accessed 06 Nov 2022]. |
| [8] | NIST, "National Vulnerability Database," [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2018-1000520. [Accessed 06 Nov 2022]. |
| [9] | "ssl\_server.c and ssl\_client1.c are using an SHA-1 signed certificate," [Online]. Available: https://github.com/Mbed-TLS/mbedtls/issues/1519. [Accessed 03 Nov 2022]. |
| [10] | "Mbed TLS," [Online]. Available: https://github.com/Mbed-TLS/mbedtls. [Accessed 04 Nov 2022]. |
| [11] | M. Goworko and J. Wytrębowicz, "A Secure Communication System for Constrained IoT Devices—Experiences and Recommendations," *Sensors,* vol. 21, no. 20, 2021. |
| [12] | A. Bhattacharjya, X. Zhong, J. Wang and X. Li, "CoAP—Application Layer Connection-Less Lightweight Protocol for the Internet of Things (IoT) and CoAP-IPSEC Security with DTLS Supporting CoAP," *Digital Twin Technologies and Smart Cities,* p. 151–175, 2019. |
| [13] | R. Herrero, "Analysis of the constrained application protocol over quick UDP internet connection transport," *Internet of Things,* vol. 12, 2020. |
| [14] | P. Kumar, "QUIC (Quick UDP Internet Connections)-A Quick Study," *arXiv preprint arXiv:2010.03059,* 2020. |
| [15] | L. Hughes, "SSL and TLS," in *Pro Active Directory Certificate Services*, Springer, 2022, pp. 155-175. |

1. [↑](#footnote-ref-1)