A Comprehensive Empirical Analysis of TLS Handshake and Record Layer on IoT Platforms

Ramzi A. Nofal
Santa Clara University
Internet of Things Research Lab
Department of Computer Science and Engineering
Santa Clara, CA, USA
rnofal@scu.edu

Nam Tran
Santa Clara University
Internet of Things Research Lab
Department of Computer Science and Engineering
Santa Clara, CA, USA
nvtran@scu.edu

Carlos Garcia
Santa Clara University
Internet of Things Research Lab
Department of Computer Science and Engineering
Santa Clara, CA, USA
cdgarcia@scu.edu

Yuhong Liu
Santa Clara University
Internet of Things Research Lab
Department of Computer Science and Engineering
Santa Clara, CA, USA
yliu@scu.edu

Behnam Dezfouli
Santa Clara University
Internet of Things Research Lab
Department of Computer Science and Engineering
Santa Clara, CA, USA
bdezfouli@scu.edu

ABSTRACT

The Transport Layer Security (TLS) protocol has been considered as a promising approach to secure Internet of Things (IoT) applications. The different cipher suites offered by the TLS protocol play an essential role in determining communication security level. Each cipher suite encompasses a set of cryptographic algorithms, which can vary in terms of their resource consumption and significantly influence the lifetime of IoT devices. Based on these considerations, in this paper, we present a comprehensive study of the widely used cryptographic algorithms by annotating their source codes and running empirical measurements on two state-of-the-art, low-power wireless IoT platforms. Specifically, we present fine-grained resource consumption of the building blocks of the handshake and record layer algorithms and formulate tree structures that present various possible combinations of ciphers as well as individual functions. Depending on the parameters, a path is selected and traversed to calculate the corresponding resource impact. Our studies enable IoT developers to change cipher suite parameters and immediately observe the resource costs. Besides, these findings offer guidelines for choosing the most appropriate cipher suites for different application scenarios.

KEYWORDS

energy, encryption, wireless, elliptic curve, key exchange

ACM Reference Format:
Ramzi A. Nofal, Nam Tran, Carlos Garcia, Yuhong Liu, and Behnam Dezfouli. 2019. A Comprehensive Empirical Analysis of TLS Handshake and Record Layer on IoT Platforms. In *22nd Int'l ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM '19), November 25–29, 2019, Miami Beach, FL, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3345768.3355924

1 INTRODUCTION

The Internet of Things (IoT) is a system of interrelated computing/smart devices, such as smart homes, health-care devices as well as autonomous driving systems, that are provided with unique identifiers and the ability to transfer data over a network without requiring human-to-human or human-to-computer interactions [1]. Due to its wide application, some studies predict that by the end of 2020, nearly 50 billion smart, connected objects will exist [2]. It is projected in a white paper by Arm that a trillion new IoT devices will be produced between now and 2035 [3]. However, the lack of security has been recognized as one of the major issues that hinders the rapid adoption of IoT systems [4].

The Transport Layer Security (TLS) protocol, which provides authentication, data integrity, and encryption between two communication parties has been widely adopted for securing communications. Therefore, extensive studies have been recently proposed to apply TLS in IoT applications. Unfortunately, the high security of TLS comes at the cost of high computational and energy demands, due to the complexity of the cryptographic algorithms adopted by TLS. More importantly, due to the limited resources available to the IoT edge devices, achieving a certain security level while minimizing the resource consumption of TLS remains one of the foremost challenges of using TLS in IoT applications [5]. The need to strike a balance between security and resource consumption in IoT applications forms the basis of this work.

In this paper, we mainly focus on the two major layers of TLS: the *handshake protocol layer* and the *record layer*, as shown in Figure 1. The handshake protocol layer, which adopts *Public Key*

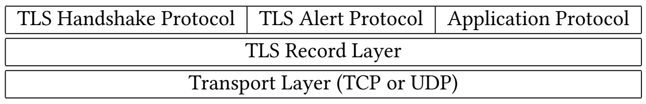


Figure 1: TLS Layers

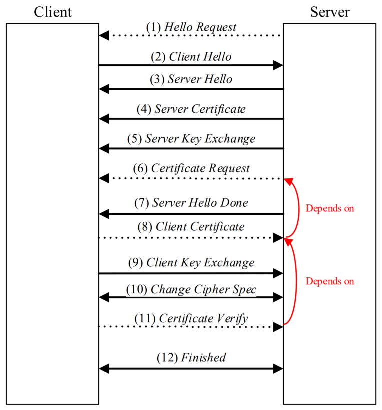


Figure 2: A visualization of the entire TLS handshake with dotted lines representing optional messages

Cryptography (PKC), allows the server and the client to authenticate each other and negotiate a set of cryptographic keys before the application layer transmits or receives its first byte of data. The record layer, which adopts *symmetric cryptography*, handles data fragmentation, encryption and decryption as well as sending and receiving TLS messages to and from the transport layer (TCP or UDP).

TLS has several advantages over other protocols. As mentioned, it is widely adopted, highly secure, and easily implementable. An observation by Google demonstrates that more than 25% of connections to its server use TLS [6]. Another report carried out by Mozilla reveals more than 50% of observed connections use TLS [7]. Also, using TLS extensions may help reduce the number of messages exchanged between client and server. On the other hand, the adoption of TLS presents some challenges. From the IoT point of view, the computing requirements of TLS may be too demanding considering resource-constrained IoT devices. One can argue that this problem can be partially overcome by substituting DTLS.However, IoT protocols such as MQTT, which implement information-centric networking, rely on connection-oriented protocols such as TCP and TLS to establish communication paths between publishers and subscribers.

Although much research has been conducted to measure the energy consumption of PKC and symmetric cryptography algorithms on IoT boards [5, 8], very few investigate the energy consumption of TLS protocol. Furthermore, the existing studies on TLS [9] (i) often lack generality by focusing on obsolete devices or a limited number of cipher suites, and (ii) are not sufficiently deep into each individual function, thereby severely limiting the capability to evaluate diverse cipher suites based on various application requirements.

This paper addresses the aforementioned concerns by evaluating the performance of cipher suites using two state-of-the-art and resource-constrained IoT edge devices. To accomplish this, we examine and annotate the source code of the cipher suites to enable these IoT boards to interface with our energy measurement platform. By analyzing the data collected from our testing platforms, this work brings an update to the field's understanding of the real world resource consumption impact from varying the parameters of TLS handshake and record layers. Consequently, our main contribution is a set of guidelines, embodied by a set of tree representations, to show all the possible configurations of a cipher suite. Depending on the usage parameters, different paths are traversed on the tree edges. This allows the IoT developer to modify the parameters relevant to the cipher suite as well as the certificate and immediately find the resource consumption impact. Moreover, these guidelines provide directions for selecting the most appropriate cipher suites according to different application scenarios.

The rest of this paper is organized as follows. Section 2 specifies the TLS handshake's messages, presents the algorithms associated with each message, and provides a brief overview of the record layer. In Section 3, the set of algorithms in a cipher suite are analyzed. The testing platforms, experimental procedure and design choices are presented in Section 4. Section 5 discusses experimental results and proposes guidelines for the most suitable cipher suite family depending on usage scenarios. Related works are discussed in Section 6. Finally, Section 7 concludes the paper and proposes future research directions.

2 TLS'S HANDSHAKE AND RECORD LAYERS

As discussed earlier, the TLS protocol is designed to establish a secure channel between a client and a server to provide information authenticity, confidentiality, and integrity. Specifically, its handshake layer adopts different PKC algorithms for identity authentication and allows the negotiation of a cipher suite, which consists of a set of cryptographic algorithms. This enables data confidentiality and integrity later in the record layer. Since the detailed understanding of the handshake procedure is essential for analysing resource consumption, we first summarize the phases of this procedure as follows. The entire process is presented in Figure 2.

(1) *Hello Request*: This message may be sent by the server at any time as a notification that the client should begin a new negotiation. The client may ignore this message or send *Client Hello* when convenient.
(2) *Client Hello*: This message is sent when the client first connects to the server or in response to a *Hello Request*. The record of this message contains the following fields: (i) TLS version, (ii) a random number, (iii) an optional Session ID to quickly resume a previous TLS connection and skip some steps of the TLS handshake, (iv) a list of cipher suites (specifies the key exchange algorithm, bulk encryption algorithm, MAC, and a Pseudo-Random Function (PRF)), (v) the compression method, which is often *null* to avoid CRIME attacks [10].
(3) *Server Hello*: This message is sent by the server in response to *Client Hello*. A Server Hello message includes: (i) TLS version, (ii) a random number, (iii) session ID: in case *Client Hello*'s session ID is not empty, the server looks into its session cache for a match, and (iv) the cipher suite selection chosen from the

client list, which should be the strongest suite supported by both sides.

(4) *Server Certificate*: The server must send a certificate message immediately after *Server Hello*. The certificate type must be appropriate for the selected cipher suite key exchange algorithm.
(5) *Server Key Exchange*: This message is sent immediately after the *Server Certificate* message. Only for Ephemeral Diffie-Hellman (DHE), Diffie-Hellman (DH), and RSA key exchange the server uses this message to specify the cryptographic parameters. In the case of Ephemeral Elliptic Curve Diffie-Hellman (ECDHE) the key exchange parameters reside in the *Server Certificate* message. If the key exchange is based on EC, the server specifies the curve name only. Both the server and the client can derive the curve parameters, such as prime p and generator G, based on the NIST standard [11]. The server also chooses a random private key a, computes $a * G$ as the public key, and saves it in `server_params`. In addition to this, the server also signs the data with its private key.
(6) *Certificate Request*: A server can optionally request a certificate from the client. This message, if sent, will immediately follow *Server Key Exchange* message. The message *Certificate Request* specifies the certificate types that the client may offer and a list of supported hashing/signature algorithms pairs that the server is able to verify, according to [12].
(7) *Server Hello Done*: This message is sent by the server to indicate the end of the *Server Hello* message.
(8) *Client Certificate*: This message is sent only if the server requests a certificate, which must be appropriate for the negotiated cipher suite's key exchange algorithm. The client signs the certificate. In this paper, we use either RSA or ECDSA as the signing algorithms.
(9) *Client Key Exchange*: This message is sent after receiving *Server Hello Done*. This message contains the client's DH public key, due to our choice of using ECDH or ECDHE as PKC. If the cipher does not indicate ephemeral key like ECDH, the message will be empty.
(10) *Change Cipher Spec*: This message is sent by both the client and server, as the final non-encrypted message, in order to notify the other party that subsequent messages are protected under the most recently negotiated cipher suite.
(11) *Certificate Verify*: This message is sent by the client only when *Client Certificate* is sent, in order to provide explicit verification for its own certificate. This message contains all the messages sent or received starting at the message *Client Hello* and up to but not including this message. This requires the client to buffer the messages or compute or buffer the hash of the previous messages.
(12) *Finished*: This message is sent immediately after *Change Cipher spec*, in order to verify that the key exchange and authentication processes have been successful. The *Finished* message is the first protected packet with the most recently negotiated algorithms, keys, and secrets. Before transmitting any encrypted data, both the client and the server generate several keys including the encryption keys, MAC keys and IV (nonce) using the master_secret as the seed for the PRF.

After the handshake phase is completed, the data transmitted between the client and server at the record layer will be secured by applying the symmetric cryptographic algorithms negotiated above in the listed steps. In the next section, we will investigate the algorithms and their individual functions, that construct a cipher suite, in detail.

3 CIPHER SUITE'S SET OF ALGORITHMS

A cipher suite is a set of cryptographic algorithms used for handshake and record layer, consisting of the following tasks: key exchange, signature for authentication, bulk encryption, and message integrity. For example, consider the following cipher suite: `ECDHE_RSA_WITH_AES_128_GCM_SHA256`.
This means that this cipher suite uses the following algorithms: Ephemeral Elliptic Curve Diffie-Hellman for key exchange, RSA for signature (verifying and signing), AES in GCM mode with 128-bit keys for bulk encryption/integrity, and SHA256 for key derivation (in the case of CBC, SHA256 is used as the hash function for HMAC). For IoT applications, in particular, ECC provides the same level of security using smaller key size compared to RSA. Therefore, we select an optimized version of ECC, called micro-ECC [13], to measure the resource consumption of ECDSA and ECDHE. In this section, we analyze the computations of each algorithm.

It should be noted that the negotiated hash function is used to implement HMAC - for non-AE (Authenticated Encryption) ciphers - and/or to implement PRF (via *P_hash*) to expand the secret keys [12]. The computations below occur on the client side:

(1) Key exchange: The computation used by EC requires inputs from the certificate, including the curve name, containing $[a, b]$ (public curve parameters), p (prime modulus specifying the size of the finite field), N (number of points on the curve), G (a generator), the order n and the co-factor h of the subgroup. Therefore the computations by ECDHE include the generation of a random number r chosen from $\{1, 2, ..., n-1\}$ (the private key) as well as the calculation of the public key $r * G$ and the shared key $r * s$, where s is the server's public key.
(2) Digital signature: The client certificate is signed based on the cryptographic parameters sent by the server in message *Client Certificate*. For RSA signing: the computation is $s = t^d \mod N$, where t is the hash value of the certificate content, d is the client's private key and N is the RSA modulus. For ECDSA signing, a random number k is generated from $\{1, 2, ..., n-1\}$, where n is the subgroup order. Then, the following steps occur: find the point $R = k * G$, find the value $r = R_x \mod n$ (where R_x is the x-coordinate of R) and calculate $s = k^{-1} * (t + r * d) \mod n$ (where d is the client private key and k^{-1} is the multiplicative inverse of k modulo n). The pair (r, s) is the signature.
(3) Signature verification: The signature verification is executed only on the hash value of the certificate content, regardless of algorithm. For RSA signature verification, the signature is represented by the value z. The computation is $s = t^{p_k} \mod N$, where t is the hash value and p_k is the CA's public key, and N is the RSA modulus. The last computation of this algorithm (negligible operation) is to compare if $s == z$. For ECDSA, the signature is represented by two values (r, s). In order to verify the signature, the CA's public key C_{pk} is

Table 1: The utilized IoT edge devices and their features

Device	BCM4343W (BCM)	CYW43907 (CYW)
MCU	ARM Cortex M4	ARM Cortex R4
Word Size	32-bit	32-bit
SRAM	128 KB	2 MB
Clock Frequency	100 MHz	320 MHz
WiFi Standards	802.11b/g/n	802.11b/g/n
On-chip Crypto Core	Not Available	Available

needed, which is hard-coded in our implementation. Let t be the hash value of the certificate content. The client then calculates $u_1 = s^{-1} * t \mod n$ and $u_2 = s^{-1} * r \mod n$, where s^{-1} is the multiplicative inverse of s modulo n. Then, the computation becomes $R = u_1 * G + u_2 * C_{pk}$. The last step is to validate if $r == R_x \mod n$, where R_x is the x coordinate of R.

(4) Message authentication: this is calculated for data integrity only on non-AE ciphers.

To achieve a comprehensive understanding of the resource consumption of the TLS protocol's cryptographic algorithms, in this work, we evaluate each computation discussed above. In the next section, we will discuss the evaluation methodology in detail.

4 METHODOLOGY

In this section, we present an overview of the testbed's components and the testing procedure. We also highlight the main performance measurement parameters.

4.1 IoT Edge Devices

The features of the two IoT edge devices employed in this work are summarized in Table 1. Our first testing platform, CYW43907 (CYW) [14], is an embedded wireless system-on-a-chip (SoC) manufactured by Cypress Semiconductor. Boasting the powerful ARM Cortex-R4 processor and an on-chip cryptography core, the CYW board is optimized for IoT computation-heavy applications and supports hardware-accelerated AES. The second platform, BCM4343W (BCM) [15], is an SoC built by Avnet. The BCM board provides less processing power from its ARM Cortex-M4 processor, lacking cryptographic hardware acceleration. It should be noted that the CYW board is more expensive than the BCM board, which justifies its superior specifications. The CYW board is better suited for computation-heavy tasks, thanks to its abundant computational power. In contrast, the more economical BCM platform is better for large-scale deployments. This contrast between devices allows us to generalize our experimental results of cipher suite performance and guidelines across both computation-heavy as well as scalable IoT scenarios.

4.2 Energy Measurement Tool

In this work, we adopt a powerful evaluation platform, EMPIOT, which is a fully software-controlled tool for the energy measurement of IoT devices [16]. EMPIOT is a shield board that is installed on top of a Raspberry Pi. The start-stop mechanism of EMPIOT energy measurements can be carefully controlled by utilizing the GPIO pins of the Raspberry Pi. The energy measurement accuracy

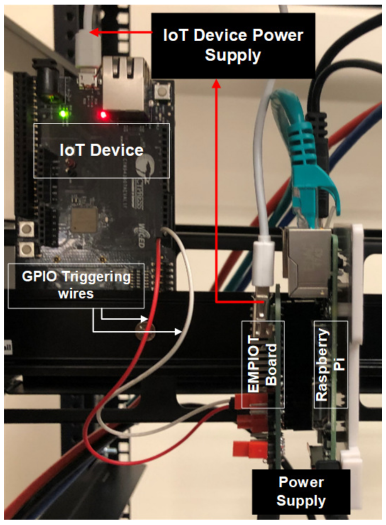


Figure 3: Components of the testbed used for performance evaluation

of EMPIOT is 0.4 μW. When measuring data from IoT devices using 802.15.4 and 802.11 wireless standards, the EMPIOT's energy measurement errors are less than 3%. When using 12-bit sampling resolution, this tool can stream 1000 samples per second. All energy and time measurements in this study have been carried out using this platform.

Figure 3 depicts the EMPIOT's components and connections with the IoT boards. The output signals from the IoT device under test to the GPIO pins of EMPIOT act as triggers to the measurement sequence. Once triggered by a positive edge signal, the EMPIOT measures the values of current (with precision of 100 μA) and voltage (with precision of 4 mV). Data measured using the EMPIOT is stored into a text file within the Raspberry Pi's on-board memory.

4.3 Cryptographic Algorithms

Cypress Semiconductor's WICED Studio version 6.1.0, the standard SDK for our testing platforms, includes a library of cryptographic algorithms [17]. To provide RSA, SHA256 and SHA384 functionalities, the WICED security library uses the *mbed TLS* free, open-source library[1]. For ECDSA, we again make use of a sample test program from the mbed TLS library that has been provided within WICED Studio. For the more optimized uECC-version ECDSA and uECC-version ECDHE, we use the implementations from the micro-ECC (μECC) library, also available in WICED Studio. Resistant to known side-channel attacks, the lightweight uECC implementations of ECDSA and ECDHE are more optimized and efficient than the standard implementations. Both uECC-version ECDSA and uECC-version ECDHE support five standard curves – secp160r1, secp192r1, secp224r1, secp256r1, and secp256k1, as well as 8, 32 and 64-bit architectures. Last but not least, we use AES-CBC and AES-GCM from the same open-source library within this software development kit. For all these implementations, we have modified

[1]https://tls.mbed.org/

and annotated the source codes to integrate fully with our energy measurement platform for resource consumption measurement.

4.4 Evaluation Process

In a single-thread configuration, we evaluate both the energy consumption and time duration for each individual algorithm (RSA, ECDSA, uECC-version ECDSA, uECC-version ECDHE, AES-CBC and AES-GCM) in our cipher suites. The evaluation follows three steps:

4.4.1 Initialization. In order to initialize an RSA experiment, the desired key size is set. Using the two common key sizes, 1024 and 2048 bits, we test RSA's hashing, encryption, decryption, signing and verification functions. Before each ECDSA experiment, the desired curve is set. Using three common curves, secp192r1, secp224r1 and secp256r1, we test ECDSA's hashing, random number generation, signing and verification operations. Similarly, the same setup for ECDSA is followed in the testing of uECC-version ECDSA. Lastly, for each uECC-version ECDHE experiment, using previously mentioned three common curves, we test uECC-version ECDHE's random number generation, public key generation, and secret calculation operations. To initialize SHA256 and SHA384 experiments, the input buffer length is set to 128 bytes. Last but not least, we test AES-CBC and AES-GCM across two buffer sizes (128 bytes and 512 bytes) as well as three key sizes (128 bits, 192 bits and 256 bits).

4.4.2 Starting Energy and Time Measurement. The first GPIO positive-edge trigger is enacted at time T_s, signaling EMPIOT to commence energy measurement. Depending on the specific combination of a particular experiment's parameters such as key size and curve size, the function being tested is repeated for N times using a `for` loop. Although the overhead of the `for` loop structure and the `if` statement is included, we find that the time and energy consumed by these operations are negligible.

4.4.3 Completing Energy and Time Measurement. Following the conclusion of the Nth encryption at time T_e, a second positive-edge GPIO pin is enacted, signaling EMPIOT to conclude energy measurement. Over an interval $[T_s, T_e]$, EMPIOT gathers 1000 samples per second of instantaneous current and voltage values. Each sample is also captured with a timestamp on it. Representing the total number of samples taken over the interval as M, the following equation is used to obtain total energy consumption E_{total} (J) over the interval: $E_{total} = \sum_{i=1}^{M} \frac{(I_i V_i + I_{i-1} V_{i-1})}{2}(t_i - t_{i-1})$ where I and V stand for current and voltage, respectively. The total energy is calculated using the trapezoidal rule. The sum of $I_i V_i$ and $I_{i-1} V_{i-1}$ represents the sum of the two bases of a trapezoid. The term $t_i - t_{i-1}$ stands for the height of the mentioned trapezoid. The area of the trapezoid represents the total energy consumed. For each specific combination of a particular experiment's parameters, the number of repetitions is denoted by N. We are only interested in the energy consumption per repetition, and so the average energy consumption per repetition with unit as joules (J) is calculated as follows: $\bar{E} = \frac{E_{total}}{N}$, where N is the number of repetitions. To acquire the total amount of time, the first sample's timestamp in seconds (s) is subtracted from the last sample's timestamp, as follows: $T_{total} = T_e - T_s$, where T_s is the time the first positive-edge trigger is called and T_e is the time the second positive-edge trigger is called, concluding data sampling. To acquire the average duration per repetition, we follow the same principle as in the previous energy consumption calculation: $\bar{T} = \frac{T_{total}}{N}$, where N is the number of repetitions.

The entire evaluation process above is repeated for all the tests outlined in the initialization step.

Table 2: Mean energy consumption (joule) per repetition

Platform	BCM			CYW		
Curve Type	192	224	256	192	224	256
ECDSA						
Signing	0.0405	0.0545	0.0817	0.0388	0.0515	0.0748
Verification	0.0788	0.1068	0.3831	0.0743	0.0992	0.1458
uECC-version ECDSA						
Signing	0.0197	0.0253	0.0451	0.0083	0.0111	0.0192
Verification	0.0206	0.0276	0.0502	0.0089	0.0120	0.0207

Table 3: Mean duration (second) per repetition

Platform	BCM			CYW		
Curve Type	192	224	256	192	224	256
ECDSA						
Signing	0.2080	0.2781	0.4193	0.0458	0.0610	0.0883
Verification	0.4037	0.5435	0.8257	0.0891	0.1187	0.1743
uECC-version ECDSA						
Signing	0.0343	0.0445	0.0813	0.0070	0.0098	0.0175
Verification	0.0361	0.0486	0.0900	0.0076	0.0106	0.0190

5 EXPERIMENTAL RESULTS

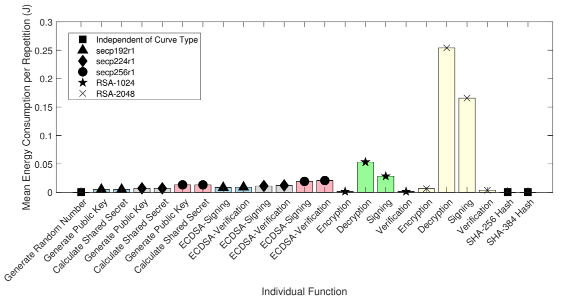
First, we discuss the performance difference between regular ECDSA and uECC-implemented ECDSA to support our decision to use the uECC implementations of ECDSA and ECDHE for all the measurements. As mentioned in Section 4, we test both standard ECDSA and uECC-implemented ECDSA. Tables 2 and 3 show the mean energy consumption and duration per repetition in joules (J) and seconds (s), respectively, for both ECDSA and uECC-implemented ECDSA's individual functions on the BCM and CYW boards. We notice that, on average, regular ECDSA's signing and verification operations consume approximately 2x to 8x more energy than uECC-implemented ECDSA's signing and verification operations. Hence, we decide to use uECC-implemented ECDSA and uECC-implemented ECDHE for the evaluations.

For the rest of this section, we present and evaluate performance results for the following four cipher suite families:

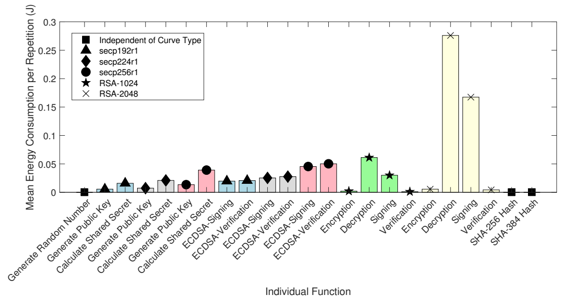
(1) `ECDHE_RSA_WITH_AES*`[2]`_SHA*`[3]
(2) `ECDHE_ECDSA_WITH_AES*_SHA*`
(3) `RSA_WITH_NULL_SHA*`
(4) `ECDHE_anon_WITH_AES*_SHA*`

Each family represents a set of cipher suites. As mentioned, each cipher suite consists of the following tasks: key exchange and authentication, with bulk encryption and message integrity. Cipher suites with `anon` authentication means the client does not need to

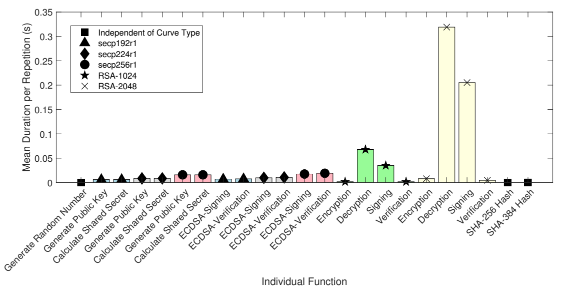
[2]AES* stands for AES_128(256)_CBC(GCM)
[3]SHA* stands for SHA256(384)



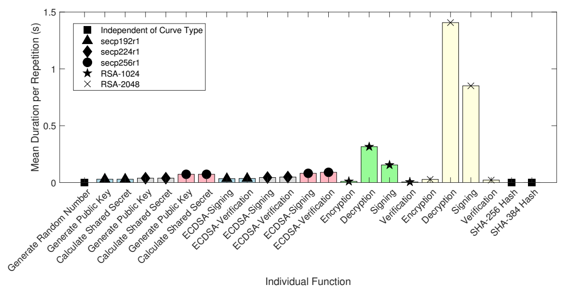
(a) Energy - CYW



(b) Energy - BCM



(c) Duration - CYW



(d) Duration - BCM

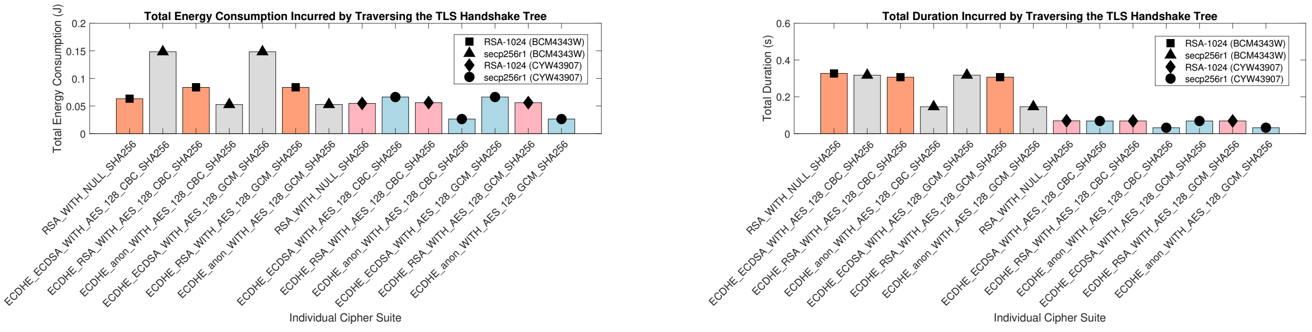
Figure 4: Mean energy consumption and duration of the elementary individual functions that construct any cipher suite from the four families, using three elliptic curves as well as RSA-1024 and RSA-2048 as parameters across two IoT platforms. We notice that as an individual function's curve size increases, that function's energy consumption also increases. Similarly, as RSA key size increases, RSA's individual functions demand more energy.

authenticate the servers. Cipher suites with NULL encryption means no encryption is required, yet message integrity is still needed. In particular, in Subsection 5.1, we present the energy consumption and duration of all the fine-grained building blocks that make up the four families above. Based on these building blocks, in Subsection 5.2, we select seven widely-used cipher suites from the four families as examples and discuss their energy consumption and duration in detail. In addition, we present two tree-like diagrams to illustrate all possible cipher suites within the four families. Finally, in Subsection 5.3, we provide a set of guidelines on selecting the most appropriate cipher suites for different application scenarios.

5.1 Resource Consumption of Individual Building Blocks

Figure 4 presents both energy consumption (J) and time duration (s) for all the building blocks of the cipher suites from the four listed families. Measurement parameters include three elliptic curves secp192r1, secp224r1 and secp256r1 as well as RSA-1024 and RSA-2048. Each subplot contains groups of bars, distinguished by the legend markers, representing the characteristics of each cipher suite's elementary individual functions. For instance, the markers show which elliptic curve a function belongs to, or if the function is a part of RSA-1024 or RSA-2048. These results show that in cipher suites using RSA, the signing is always more computationally expensive in terms of energy consumption (about 21x-46x) and duration (about 21x-45x) than verification for the same key size. We also notice that for cipher suites using ECDSA, the signing and the verification's energy consumption is closely matched for the same key size. A similar trend is noticed for execution duration.

Cipher suites using RSA and ECDSA can be compared with each other by referring to Table 4. This table lists comparable key sizes for symmetric and asymmetric-key cryptosystems, with both theoretical and industrial ECC key sizes, based on the most popular algorithms for attacking them [18]. For example, using industrial ECC 160-bit key size provides the same level of security as using RSA-1024. We actually use curve secp192r1 for ECDSA, which employs a 192-bit key size. With this comparison on the BCM platform testing RSA-1024 and ECDSA using curve secp192r1, the verification for ECDSA consumes almost 20x more energy than RSA's. In terms of duration, the verification for RSA is almost 6.5x quicker than ECDSA's. On the other hand, RSA's signing demands approximately 1.5x more energy compared to ECDSA's signing. Duration-wise, RSA's signing takes approximately 4.5x more compared to ECDSA's signing. Similarly, considering Table 4, we can compare RSA-2048 and ECDSA using curve secp224r1, which employs a 224-bit key size. For this case, the verification for ECDSA consumes about 6.4x more energy than RSA's verification while ECDSA's signing demands approximately 6.6x more energy than RSA's signing. In terms of execution time, the verification for ECDSA lasts about



(a) Energy (b) Duration

Figure 5: Total energy and duration of seven widely used cipher suites from the four families across the BCM and CYW platforms.

2.2x longer than RSA’s verification while RSA’s signing lasts approximately 19x longer than ECDSA’s signing. Furthermore, from the same figures, we notice that RSA’s decryption is more costly energy-wise and duration-wise than encryption across two keys.

Table 4: Comparable Key Sizes (in bits)

Symmetric	Theoretical ECC	Industrial ECC	DH/DSA/RSA
80	163	160	1024
112	233	224	2048
128	283	256	3072
192	409	384	7680
256	571	512	15360

For shared or premaster key generation using ECDHE, we notice that the larger the curve size, the greater the energy consumption of the function belonging to that curve. The same trend in energy consumption translates to duration. For shared or premaster key generation using ECDHE, we notice on the same figures that the larger the curve size is, the greater the execution duration of the function belonging to that curve. Random number generation’s energy consumption and execution duration are constant, as it is not associated with any curve. Since generating the public key and computing the shared secret depend on the curve size, certainly, increased security comes with a higher demand for energy and higher cost for duration. The behaviors described above for energy consumption and duration are detected on both of our IoT testing platforms.

5.2 Resource Consumption of Prevalent Cipher Suites

Based on the above analysis of the fine-grained individual functions, this subsection further explores the resource consumption of all possible cipher suites from the four families.

Since each cipher suite consists of several functions, to find the full energy consumption or duration of a cipher suite, we simply need to sum up all the functions’ energy consumption or duration data values. Figure 5 shows the aggregate energy and duration of seven widely-used cipher suites from the four families. Each bar represents the total energy consumption (a) or duration (b) of the entire set of algorithms that forms the cipher suite. Once again, it is obvious that the CYW platform outperforms the BCM board for both energy consumption and duration, thanks to the CYW’s on-chip cryptography core.

To provide in-depth guidelines on how each family should be selected, we present two tree-like diagrams to show every single function that constructs a TLS cipher suite. Figures 6 and 7 depict all possible sequences of functions in the handshake and the record layer, respectively, with different parameters such as key size, curve type and data size. It should be noted that in Figure 7, GCM, which is AEAD[4], combines encryption and integrity (the GMAC component of the GCM algorithm) together. In contrast, for CBC, the integrity is split from encryption/decryption.

In order to illustrate the impact of different sequences on resource consumption, two possible paths, shown by green and red lines in Figure 6, are compared. The red path does not include authentication and uses only RSA 1024-bit key exchange. The green path utilizes RSA-1024 authentication and uses ECDHE key exchange with the secp256r1 curve. Nevertheless, the green path is a more secure solution. Upon analysis, when using the CYW board,

[4]Authenticated Encryption with Associated Data

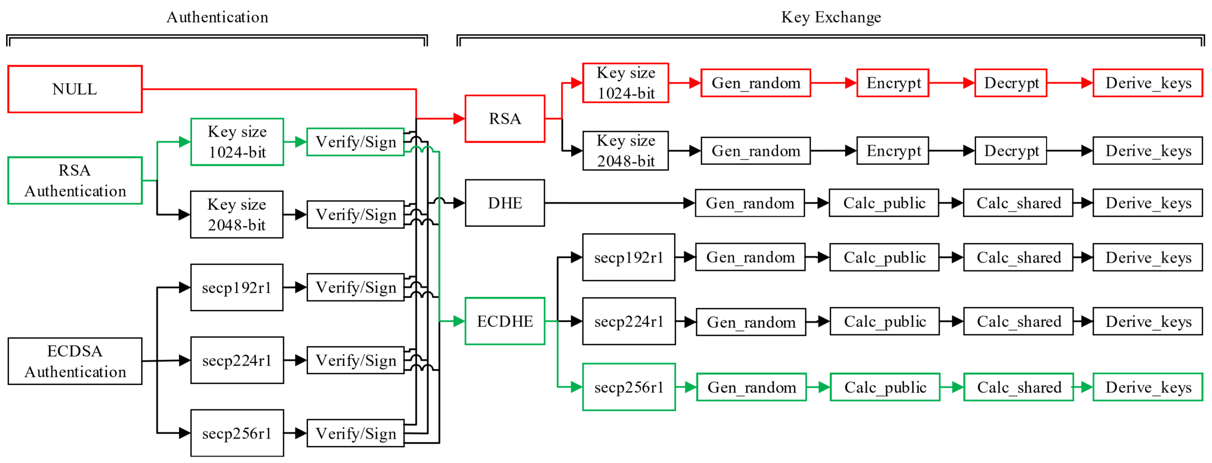


Figure 6: TLS Handshake Tree

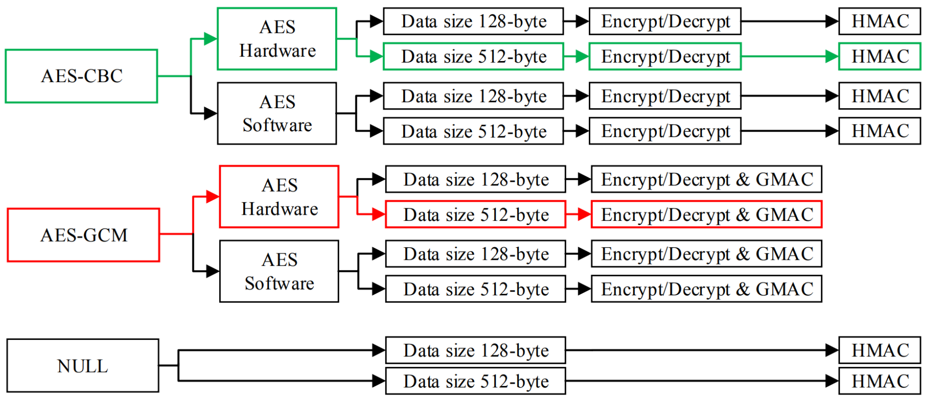


Figure 7: TLS Record Layer Tree

we notice that the red path uses 1.07x more energy and takes the same amount of time as the green path. In the case of the BCM platform, the energy consumption of red path is 0.43x of that of the green path, while the time taken by both paths are equal.

Now, to show resource consumption differences within the record layer, as shown in Figure 7, an example is provided. Representing two possible sequences, the green and red paths correspond to bulk ciphers AES-CBC and AES-GCM, respectively. Since it is known that symmetric key cryptography is way faster than asymmetric key cryptography, we may select any two arbitrary paths as examples. In Figure 7, using the CYW board, both paths use the hardware implementation of AES with a 128-bit key size and a 512-byte data size. Based on the data, the red path uses 1.36x more energy and takes 1.28x longer the green path. The same paths cannot be modeled on the BCM platform, as it does not support hardware acceleration for AES. Still, using the software implementation on the BCM to model these paths, we also see that AES-GCM has higher resource consumption than AES-CBC as it uses 1.79x more energy and takes 1.80x longer.

These comparisons provide an overview of the resource consumption incurred by different cipher suites for both handshake and record layer. This enables us to transition to providing specific guidelines on cipher suite selection in the next section.

5.3 Guidelines for Cipher Suite Selections

Based on our experimental results, we propose the following guidelines for selecting appropriate families depending on usage requirements.

Scenario 1: This scenario arises in applications requiring authenticity, confidentiality, and integrity. Such applications include healthcare (e.g., body area sensors) and smart homes (e.g., smart door locks, security systems), where confidentiality and privacy play a critical role. Family 1 and family 2 are most suitable for this scenario. Depending on application requirements, a longer (or shorter) key can be adopted to achieve higher (or lower) security. Assuming same key and signature size, family 2 consumes about 1.5x more energy and takes the same amount of time as family 1. Therefore, family 2 is recommended when both cipher suite families are available, while family 1 may be chosen in the absence of family 2.

Scenario 2: This scenario arises in applications requiring authenticity and integrity, but not necessarily confidentiality. Applications for this scenario may include smart city sensors that transmit non-confidential data but still require authenticity and integrity. Family 3 is most suitable for this scenario. If we add confidentiality to this family, for data size 512-byte, we may increase the energy consumption by approximately 1.001x. Though insignificant, if the data increases to 50KB, the energy consumption will increase by approximately 1.5x. Adding confidentiality may significantly increase energy consumption in the long run.

Scenario 3: This scenario arises in applications requiring confidentiality and integrity, but not necessarily authenticity. Family 4 is most suitable for this scenario. Even though this family is vulnerable to man-in-the-middle (MITM) attacks, it can still be used for closed-environment applications such as sensors on a military jet or in machine actuators in a factory's local network. Adding authenticity to this scenario (e.g., using RSA verification) will increase the energy consumption on the client side by only 1.05x. However, the overhead of managing and transmitting certificates increases dramatically.

This leads us to the culmination of this work. Based on energy and duration measurements, the right cipher suite family can be chosen to fulfill user requirements while achieving a desired balance between security and efficiency. Specifically, as future work, we recommend dynamic cipher suite adjustments to balance IoT devices' security and resource consumption, based on real-time network condition changes, such as changes to transmitted data type, traffic volume and wireless signal quality. When a change occurs, a new cipher suite negotiation may be launched accordingly.

6 RELATED WORK

There are many papers that analyze the power consumption of security protocols on resource-constrained platforms like IoT devices. Some works are conducted to analyze the impact of symmetric cryptography on the resource consumption of IoT devices [19–21]. In the study by Munoz et al. [19], the authors measure the duration and energy consumption of AES-CBC on both the CYW and BCM platforms. In [21], the authors compare hardware and software AES implementations on an FPGA. In [20], the authors test the impact of increasing AES key size on energy consumption.

In addition, many studies explore the resource consumption of PKC [22–25]. Potlapally et al. [5] study the Secure Sockets Layer (SSL) and the underlying PKC algorithms. The authors in [26] evaluate PKC's influence on four IoT devices mainly using the RSA and ECC families, but not Diffie-Hellman, which is the most widely used PKC family on IoT devices. These studies, however, ignore cipher suites that use ephemeral keys, a popular approach that provides higher security while requiring extra computations. Furthermore, although using customized EC library dedicated for resource-constrained devices is highly recommended, the existing studies often provide no information on their implementations for EC.

The following papers focus on the resource consumption of TLS. In [8], the authors provide a novel model of energy demand for end-to-end data communication. Their approach is to represent the energy consumption of each node in the network as a function. However, the specific information of the energy measurement process, particularly the data collection process and the types of tested IoT platforms, is absent. The authors in [27] evaluate the performance costs of TLS by using the same cipher suites that are used in our paper. Nevertheless, their study is not carried out using resource-constrained devices. In addition, for both [8] and [27], the authors do not analyze the fine-grained functions within the cipher suites, severely limiting their capability to evaluate diverse cipher suites based on various application requirements. While Gerez et al. investigate the energy and time consumption of TLS, they only focus on the handshake and ignore the record layer [28]. Moreover, their experiments are only carried out using three specific cipher suites on one IoT board, which greatly limits the generality of their results.

Our paper improves on these existing studies by (1) considering both the handshake and record layer of TLS, (2) measuring the resource consumption of each individual function, and (3) improving the results' generality by adopting two state-of-the-art resource-constrained IoT boards and evaluating various widely-used cipher suites. We also provided comprehensive tree diagrams to reflect all the possible paths of processes in the handshake and record layers. These representations not only facilitate flexible calculation of the resource consumption of different cipher combinations, but also provide guidelines for selecting the most appropriate cipher suite according to different application demands.

7 CONCLUSION

Due to the ever increasing number of IoT platforms that need to establish secure connections, we consider the three most important factors of IoT security: authenticity, confidentiality and integrity. Given the resource constraints and long term communication patterns of IoT devices, it is crucial to, depending on the application at hand, choose a proper cipher suite that minimizes resource consumption while ensuring the required security level.

In this paper, we presented a comprehensive study of the most widely-used cryptographic algorithms by annotating their source codes and running empirical measurements on two state-of-the-art IoT platforms (i.e. CYW and BCM). Also, we formulated the tree structures that cover various possible cipher combinations. We showed that by carefully choosing the right cipher suite family based on application requirements, energy consumption can be significantly reduced. For example, in use cases such as smart home and healthcare where high-level security is demanded, selecting `ECDHE_RSA_WITH_AES*_SHA*` can reduce energy consumption by 3x. Another example can be seen in smart city applications, where using `RSA_WITH_NULL_SHA*` results in savings up to 1.6x in the long run.

In this work, we did not consider the impact of real-time network condition variations such as the changes to transmitted data type, traffic volume and wireless signal quality, on the resource consumption. Although we focused on TLS 1.2, TLS 1.3 is already available, providing some changes with respect to TLS 1.2 [29]. Nevertheless, the migration towards TLS 1.3 will take a long time and hence, our analysis remains relevant for all current devices using TLS up to 1.2. The energy-duration study of TLS 1.3 is left as a future work.

ACKNOWLEDGMENT

This work has been partially supported by research grant CYP001 from Cypress Semiconductor Corporation.

REFERENCES

[1] Behnam Dezfouli, Marjan Radi, and Octav Chipara. REWIMO: A real-time and reliable low-power wireless mobile network. *ACM Transactions on Sensor Networks (TOSN)*, 13(3):17, 2017.

[2] Habib Ur Rehman, Muhammad Asif, and Mudassar Ahmad. Future applications and research challenges of iot. In *International Conference on Information and Communication Technologies (ICICT)*, pages 68–74. IEEE, 2017.

[3] One trillion new iot devices will be produced by 2035. URL https://learn.arm.com/route-to-trillion-devices.html.

[4] Jorge Granjal, Edmundo Monteiro, and Jorge Sá Silva. Security for the internet of things: a survey of existing protocols and open research issues. *IEEE Communications Surveys and Tutorials*, 17(3):1294–1312, 2015.

[5] Nachiketh R Potlapally, Srivaths Ravi, Anand Raghunathan, and Niraj K Jha. A study of the energy consumption characteristics of cryptographic algorithms and security protocols. *IEEE Transactions on mobile computing*, 5(2):128–143, 2006.

[6] URL https://transparencyreport.google.com/https/overview.

[7] Klint Finley. Half the web is now encrypted. that makes everyone safer, Jun 2017. URL https://www.wired.com/2017/01/half-web-now-encrypted-makes-everyone-safer/.

[8] Arcangelo Castiglione, Alfredo De Santis, Aniello Castiglione, Francesco Palmieri, and Ugo Fiore. An energy-aware framework for reliable and secure end-to-end ubiquitous data communications. In *5th International Conference on Intelligent Networking and Collaborative Systems (INCoS)*, pages 157–165. IEEE, 2013.

[9] Tim Dierks and Eric Rescorla. The transport layer security (tls) protocol version 1.2. 2008.

[10] Fitzgerald Shawn Sarkar, Pratik Guha. Attacks on ssl a comprehensive study of beast, crime, time, breach, lucky 13 and rc4 biases. *iSecPartners*, pages 1–23, 2013.

[11] Lily Chen and Dustin Moody. Elliptic Curve Cryptography, 2017. URL https://csrc.nist.gov/Projects/Elliptic-Curve-Cryptography.

[12] E. Rescorla T. Dierks. The transport layer security (tls) protocol version 1.2. https://tools.ietf.org/html/rfc5246, Aug 2008.

[13] Zhe Liu, Xinyi Huang, Zhi Hu, Muhammad Khurram Khan, Hwajeong Seo, and Lu Zhou. On emerging family of elliptic curves to secure internet of things: Ecc comes of age. *IEEE Transactions on Dependable and Secure Computing*, 14(3):237–248, 2017.

[14] Cypress Semiconductor. CYW943907AEVAL1F Evaluation Kit, 2018. URL http://www.cypress.com/documentation/development-kitsboards/cyw943907aeval1f-evaluation-kit.

[15] Avnet Inc. Avnet BCM4343W IoT Starter Kit, 2018. URL http://cloudconnectkits.org/product/avnet-bcm4343w-iot-starter-kit.

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

[17] Cypress Semiconductor. WICED Studio, Mar 2018. URL http://www.cypress.com/products/wiced-software.

[18] S. Blake-Wilson, N. Bolyard, V. Gupta, C. Hawk, and B. Moeller. Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS). https://tools.ietf.org/html/rfc4492, May 2006.

[19] Pedro Sanchez Munoz, Nam Tran, Brandon Craig, Behnam Dezfouli, and Yuhong Liu. Analyzing the resource utilization of aes encryption on iot devices. In *Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)*, pages 1–8, 2018.

[20] Diaa Salama Abd Elminaam, Hatem Mohamed Abdual-Kader, and Mohiy Mohamed Hadhoud. Evaluating the performance of symmetric encryption algorithms. *IJ Network Security*, 10(3):216–222, 2010.

[21] Anton Biasizzo, Marko Mali, and Frank Novak. Hardware implementation of aes algorithm. *Journal of Electrical Engineering*, 56(9-10):265–269, 2005.

[22] William Freeman and Ethan Miller. An experimental analysis of cryptographic overhead in performance-critical systems. In *7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 348–357. IEEE, 1999.

[23] Neil Daswani and Dan Boneh. Experimenting with electronic commerce on the palmpilot. In *International Conference on Financial Cryptography*, pages 1–16. Springer, 1999.

[24] Abdullah Almuhaideb, Mohammed Alhabeeb, Phu Dung Le, and Bala Srinivasan. Beyond fixed key size: Classifications toward a balance between security and performance. In *24th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, pages 1047–1053, 2010.

[25] George Apostolopoulos, Vinod Peris, Prashant Pradhan, and Debanjan Saha. Securing electronic commerce: reducing the ssl overhead. *IEEE Network*, 14(4):8–16, 2000.

[26] Krzysztof Piotrowski, Peter Langendoerfer, and Steffen Peter. How public key cryptography influences wireless sensor node lifetime. In *Proceedings of the fourth ACM workshop on Security of ad hoc and sensor networks*, pages 169–176, 2006.

[27] Lin-Shung Huang, Shrikant Adhikarla, Dan Boneh, and Collin Jackson. An experimental study of tls forward secrecy deployments. *IEEE Internet Computing*, 18(6):43–51, 2014.

[28] Alejandro Hernandez Gerez, Kavin Kamaraj, Ramzi Nofal, Yuhong Liu, and Behnam Dezfouli. Energy and processing demand analysis of tls protocol in internet of things applications. In *2018 IEEE International Workshop on Signal Processing Systems (SiPS)*, pages 312–317, 2018.

[29] E. Rescorla. The transport layer security (tls) protocol version 1.3. https://tools.ietf.org/html/rfc8446, Aug 2018.