Energy and Processing Demand Analysis of TLS Protocol in Internet of Things Applications

Alejandro Hernandez Gerez*, Kavin Kamaraj†, Ramzi Nofal‡, Yuhong Liu§, and Behnam Dezfouli¶
Internet of Things Research Lab, Department of Computer Engineering, Santa Clara University, USA
*a3hernandez@scu.edu, †kkamaraj@scu.edu, ‡rnofal@scu.edu, §yliu@scu.edu, ¶bdezfouli@scu.edu

***Abstract*—Transport Layer Security (TLS) is the de-facto protocol for secure communication in Internet of Things (IoT) applications. However, the processing and energy demands of this protocol are two essential parameters that must be taken into account with respect to the resource-constraint nature of IoT devices. In this paper, we study the resource consumption of the TLS handshake using a testbed in which an IoT board (Cypress CYW43907) communicates with a Raspberry Pi server over an 802.11 wireless link. Although TLS supports a wide-array of encryption algorithms, we focus on the performance of TLS using three of the most popular and robust cipher suites. Our experiments show that ciphers using Elliptic Curve Diffie Hellman (ECDHE) key exchange are considerably more efficient than ciphers using Diffie Hellman (DHE). Furthermore, ECDSA signature verification consumes more time and energy than RSA signature verification given ECDHE key exchange. The studies of this paper help IoT designers choose an appropriate TLS cipher suite based on application demands, computational capabilities, and available energy resources.**

***Index Terms*—Security, Encryption, Computation, Wireless, Key Exchange.**

I. INTRODUCTION

IoT (Internet of Things) refers to a network of physical devices which are embedded with the ability to communicate and exchange data via Internet. IoT networks are significantly growing in number and are being used in various application domains such as factory automation, health-care, and smart agriculture [1]–[3]. As of 2018, there are approximately 9 billion IoT devices in place and experts forecast that this number will surge to 28.3 billion by 2020 [4], [5]. As information transfer across IoT networks is set to explode, most applications require secure data exchange. This necessitates the use of networking protocols that enable nodes to authenticate each other before exchanging information. Without proper authentication, a malicious node can act as either a server or a client and steal information from a node that is monitoring a classified process. Additionally, even after authentication, there must be a secure mode of communication between two parties to prevent data interception.

Transport Layer Security (TLS) protocol is designed to provide encryption, authentication, and data integrity for exchanging information over the Internet. The TLS protocol is composed of two phases [6]: The first phase is the *handshake*, which allows a client and a server to agree on TLS version and a cipher suite. A cipher suite encompasses the following encryption algorithms: server authentication algorithm, key exchange algorithm, bulk encryption algorithm, and measure digest algorithm. The agreement enables the two communicating parties to ultimately establish a shared session key. The client and server also have the option to authenticate each other using certificates provided by a trusted third-party certification authority (CA) [7]. The second phase of the protocol is the *record layer*, in which the shared session key is used to send encrypted messages between two nodes. As of today, TLS is the most widely used protocol for securing communication between IoT devices [8], [9]. This protocol is considered to be highly effective because of its use of both symmetric key and public key cryptography. Unfortunately, these advantages come at the cost of high computational and energy demands [10], [11].

Reducing the overhead of the TLS protocol is critical for multiple reasons [11]–[13]. First, as IoT devices are increasing by the billions, a few milli-joules saved from each TLS transaction can save millions of dollars in the big picture. Second, all networks are vulnerable to interference, timeouts and loss of connectivity; the overhead of the TLS handshake should not deter users from re-establishing lost connections between nodes. Third, understanding the overhead of TLS enables IoT designers to configure this protocol based on application requirements. For example, a user may intentionally deploy a light cipher (possibly compromising some aspects of security) when establishing a short-lived connection for exchanging trivial data. Furthermore, resource constraints of different IoT devices may necessitate using light ciphers regardless of the nature of data transfer. Although there are several studies focusing on the implementation of the TLS protocol on general purpose processing platforms and mobile devices, there are very few works that present the computational and energy costs of TLS on embedded IoT processors [8], [13]–[17].

In this paper, we analyze the performance of the TLS handshake using three popular and robust ciphers:

- C1: DHE_RSA_WITH_AES_256_CBC_SHA256
- C2: ECDHE_RSA_WITH_AES_256_CBC_SHA384
- C3: ECDHE_ECDSA_WITH_AES_256_CBC_SHA384

Our results show that ECDHE ciphers are considerably more efficient than DHE ciphers. We also conclude that an ECDHE cipher has lower resource consumption when using RSA encryption compared to ECDSA encryption. This is attributed to the fact that ECDSA certificate verification is longer and more computationally expensive. The results reported in this work help IoT designers choose TLS cipher suite based on

application demands as well as the computational capabilities of IoT devices.

The rest of the paper is organized as follows: In Section II, we explain the fifteen steps of the TLS protocol and the cipher suites used for our study. In Section III, we specify our hardware platform and experimental procedure. Section IV presents the analysis of collected results. Finally, Section V concludes the paper and provides future research directions.

II. Transport Layer Security (TLS)

The TLS protocol makes use of both symmetric key and public key cryptography to secure communication over a network. In symmetric key encryption, the client and server share a secret key that is used to encrypt or decrypt messages. However, this symmetric session key must somehow be communicated between the two nodes before secure communication can take place. In the TLS handshake, public key cryptography protects the exchange of the session key.

Unlike symmetric key cryptography, public key cryptography uses both a public and private key. In TLS, a public and private key pair is generated by the server. The server publishes the public key while keeping the private key to itself. The client will then use the public key to encrypt the symmetric session key which only the corresponding private key (owned by the server) can decrypt. However, this mandates the client to have a mechanism to bind the public key with the identity of the server. For this measure, the TLS protocol makes use of X.509 certificates that are signed by a certificate authority (CA) [18].

Encryption guarantees privacy but not necessarily the integrity of the data transferred between the client and the server. A malicious node can still alter the messages exchanged between nodes without detection. TLS preserves message integrity using a digest algorithm (one-way hash function) that outputs a unique digest for each input message. This mechanism ensures that any modification made to the message will also alter the digest. Therefore, a compromised message can be detected by identifying a mismatch between the digests computed by the sender and the receiver.

The TLS handshake phase consists of a total of 15 steps. At the start of the handshake, hello messages are exchanged between client and server to agree on a cipher suite and to exchange random values. Then, depending upon the cipher suite chosen, the client and server exchange the corresponding cryptographic parameters to agree on a pre-master secret. Afterwards, the client authenticates the server using a pre-installed certificate from a trusted CA; the server may optionally authenticate the client as well. Finally, a master secret is generated from the pre-master secret and the random values. The symmetric session key is derived from the master secret and is used to encrypt subsequent data exchange between the client and server in the record layer phase [19]. We present each step of the TLS handshake in detail as follows:

1) `client_hello`. The client sends this message to the server to establish an initial contact. This message, in particular, contains: (i) the TLS version that the client intends to use, (ii) a list of cipher suites supported by the client, (iii) a random number, (iv) compression method, and (v) a session ID. The server then checks its compatibility with the specified version of TLS and the list of ciphers specified in the message.
2) `server_hello`. If the server's TLS version and supported cipher suites are compatible with that of the client, this message is sent by the server in response to the client's hello message. This marks the completion of a successful negotiation.
3) `server_certificate`. The server sends to the client a certificate containing its public key. The client can authenticate the server by comparing the certificate it receives from the server to its pre-installed certificate. An authentication failure is raised in the case that the server's certificate does not match any of the certificates pre-installed by the client.
4) `server_key_exchange`. In this message, the server exchanges Diffie-Hellman cryptographic parameters (modulus, generator, newly-generated public key) with the client so that it can convey a pre-master secret. The resource consumption of this step can be attributed to the client verifying the signature of these parameters.
5) `certificate_request`. This message is sent by the server to request a certificate from the client in the case that the user has configured the server to require client authentication. In our study, the server sends a request to the client to adhere to the protocol, however, the server does not functionally verify the client.
6) `server_hello_done`. This message is sent by the server to indicate the end of the hello message exchange sequence. While this message is prepared and sent, the client is verifying the validity of server's certificate.
7) `client_certificate`. This message is sent by the client if the server has requested the client to send its certificate (i.e., if client authentication is required). Even if the client does not have a suitable certificate, it must send a certificate message that does not contain any certificate. In our study, the client is configured to send a blank certificate message to the server, thereby, this step is one of the fastest to execute.
8) `client_key_exchange`. In this message, the client sends to the server the shared secret along with its Diffie-Hellman public value (C1 uses DHE while C2 and C3 use ECDHE). The DH public value that is sent to the server is distinct for each handshake due to the ephemeral nature of the key exchange.
9) `certificate_verify`. This message is sent by the server to indicate that it has successfully verified the client's certificate. A digitally signed structure of all the handshake messages sent or received is included in this message.
10) `client_change_cipher_spec`. This message is sent by client to inform the server that subsequent data transfer will be protected by the newly negotiated ciphers.
11) `client_finished`. This message verifies that the client has successfully completed the authentication processes and key exchange.

12) `server_change_cipher_spec`. The server sends this message to notify the client that subsequent data transfer will be protected by the newly negotiated cipher and keys. The client then reacts by setting the session key parameters accordingly.
13) `server_finished`. This message verifies that the server has successfully completed the authentication processes and the key exchange.
14) `flush_buffers`. In this step, the temporary data that is a byproduct of the handshake process is deleted on both the client and server nodes.
15) `handshake_over`. This message marks the completion of the handshake phase and the start of the record layer phase.

The Internet Assigned Numbers Authority (IANA) has named over 300 cipher suites compatible with TLS in early 2016. BSI, a federal IT security agency in Germany, recommends using only 16 of those ciphers for TLS [7]. In this paper, we study the performance of the TLS protocol using three of the cipher suites that they have recommended. These are listed in Table I:

Note that all three ciphers use the same session key, AES_256_CBC. AES is a symmetric key block cipher algorithm that is used to encrypt data [20]. It is defined as a cryptographic standard by the NIST and is one of the most widely used encryption algorithms. Furthermore, it has a reputation for having high performance with relatively efficient resource consumption. In our study, AES-256 is used to encrypt and decrypt all information transfer in the record layer phase of the protocol. Hash functions SHA256 and SHA384 are used to create digital signatures of the data. As a one-way function, SHA easily authenticates messages and preserves message integrity. There are other variations for SHA (i.e., SHA224 and SHA512), but they are not available for use in mbed TLS because they do not offer added security or efficiency.

We have kept the session key and hash function type consistent among all the ciphers so that differences in resource consumption can be conclusively attributed to either the key exchange method or choice of public key encryption algorithm. The two key exchange methods used in this list of ciphers are DHE and ECDHE. They are both ephemeral key exchange methods meaning that a distinct DH public value is created in each handshake. As a result, they require more cryptographic operations than their non-ephemeral counterparts and offer more security. DHE uses modular arithmetic to compute the shared secret. In contrast, ECDHE uses elliptic curves to generate the secret; thereby, ECDHE is considerably more efficient than DHE. The key exchange method significantly impacts the establishment of the pre-master secret (Step 8) which is the most computationally expensive process in the TLS handshake. Aside from the key exchange method, our ciphers use either RSA or ECDSA public key encryption schemes. ECDSA signatures tend to be much smaller than RSA signatures given the same method of key exchange. However, RSA signature verification tends to be much faster than ECDSA verification [14].

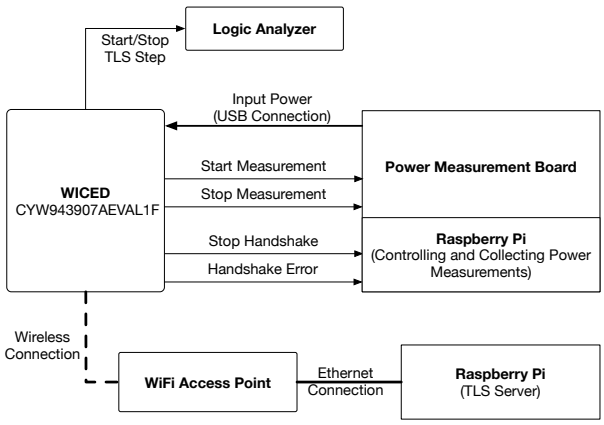


Fig. 1: The components and interconnection of the testbed.

III. Experimental Procedure

We run the TLS protocol using a Cypress CYW43907 IoT device as a client and a Raspberry Pi as a server. The client and server connect to an access point (router) through an 802.11 link. CYW43907 is an embedded wireless system-on-a-chip (SoC) that features an ARM Cortex-R4 32-bit RISC processor [21], [22]. The Cypress IoT device supports the WICED Development Platform which offers SDKs for system development. The TLS code used on the client and server is derived from the mbed TLS implementation [23]. Note that we perform all time and energy measurements on the client side.

For measuring the energy consumption of the TLS protocol we use EMPIOT (Energy Measurement Platform for IoT Devices) [24], that is connected to the client. The EMPIOT platform is composed of a shield installed on top of a Raspberry Pi (separate from the server). We measure processing time using a logic analyzer. Figure 1 shows our experimental setup for time and power measurement. When performing a TLS handshake, we can divide resource consumption into processing and transmission components. The former entails cryptographic operations such as symmetric/public key encryption, hashing, and digital signatures. The latter, on the other hand, consists of message exchange intervals over TCP. Our study focuses solely on measuring the processing components, and we have set markers in the mbed TLS code to demarcate the start and end of each step's processing duration.

The energy measurement platform, EMPIOT, is connected to the client as follows. First, EMPIOT powers the client via a USB connection and measures the bus voltage and current drawn by the client. Second, the tool detects the duration of each step by connecting to four pins toggled by the client's markers. We measure the computation and energy consumption of TLS steps 3, 4, 8, and 9, which are all longer than 1ms. We have observed that the processing time and energy consumption of the remaining steps are negligible and therefore these steps will not factor into our study.

The TLS handshake may occasionally fail due to network timeouts caused by packet loss over the wireless link. To

Alias	Cipher Suite	Authentication	Key Exchange	Encryption	Digest
C1	DHE-RSA-AES-256-SHA	RSA	DHE	AES-256-CBC	SHA256
C2	ECDHE-RSA-AES-256-SHA384	RSA	ECDHE	AES-256-CBC	SHA384
C3	ECDHE-ECDSA-AES-256-SHA384	ECDSA	ECDHE	AES-256-CBC	SHA384

TABLE I: Cipher suite configuration. The signature type corresponds with the longest signature available for each cipher suite.

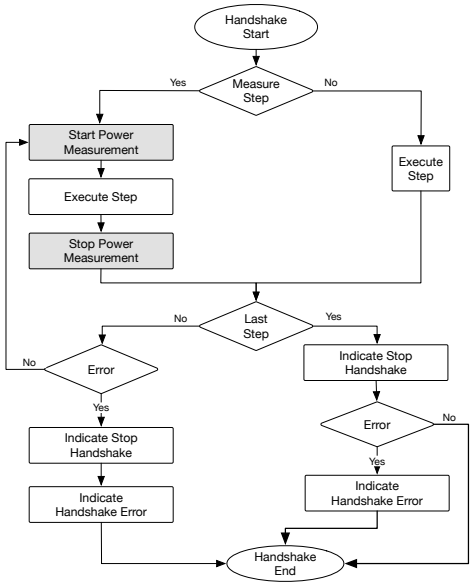
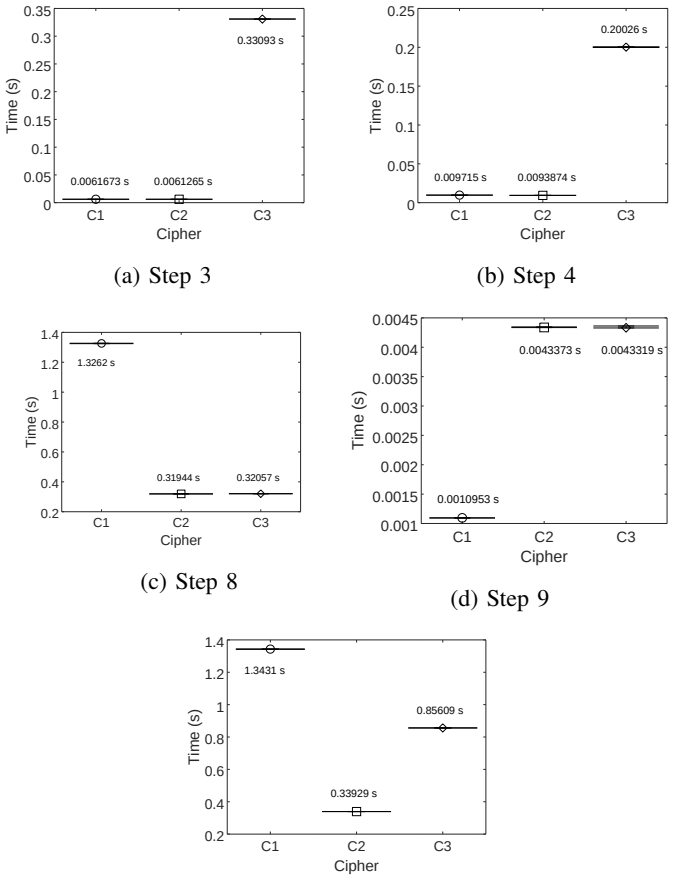


Fig. 2: The flow diagram of measurement methodology.



(a) Step 3

(b) Step 4

(c) Step 8

(d) Step 9

(e) Total

Fig. 3: Processing time of various steps. This figure confirms that client verification of an ECDSA certificate requires about 5200% longer processing compared to RSA. In addition, signature verification using ECDSA is around 2000% longer than RSA. Furthermore, key exchange using DHE shows 300% longer processing time than using ECDHE. In general, the processing overhead of C1 and C3 are 300% and 150% higher than C2, respectively.

ensure that the accuracy of our measurements is not affected by these error cases, we configure the client to notify EMPIOT if the handshake has failed. In this way, we can discard measurements collected from incomplete runs. The detail of the power measurement process is shown in Figure 2.

Our data collection procedure is as follows: We perform 1000 iterations of successful TLS handshakes for each cipher, i.e., C1, C2, C3. We measure the processing time and energy of steps 3,4,8, and 9 in each iteration.

IV. Results

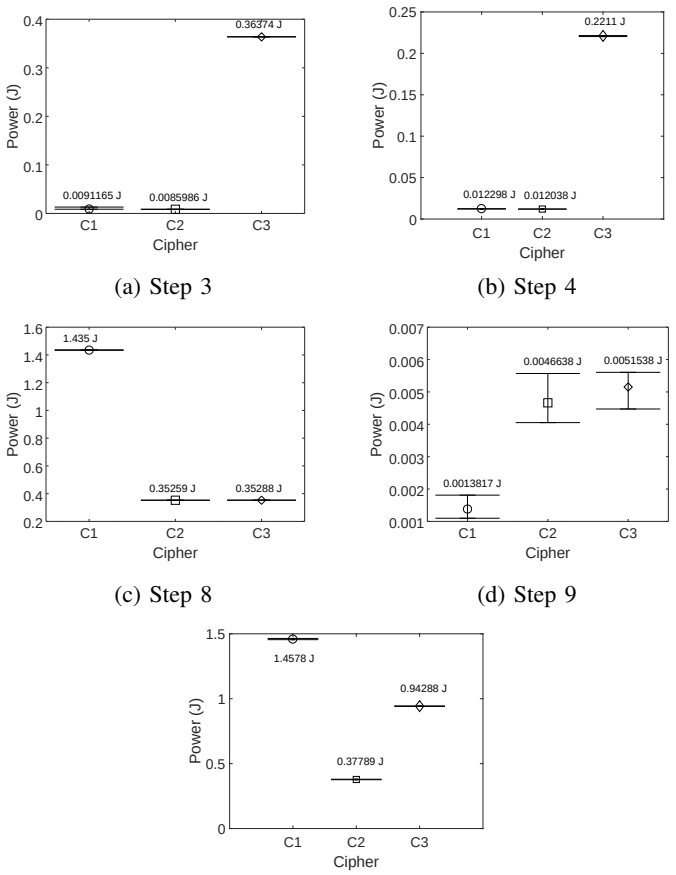
Figure 3 shows the processing time of each TLS step for the three cipher suites. Figure 4 shows the power consumption in Joules per TLS step for each cipher. Each marker denotes the median of 1000 iterations, and error bars show the higher and lower quartiles. The value of the median is shown on top of each error bar.

Analyzing Step 3. In Step 3 the server sends to the client a certificate with its public key and the client verifies this certificate. The results obtained from this step indicate that C3 has considerably higher values for both processing time and energy consumption compared to both C1 and C2. Specifically, client verification of an ECDSA certificate is heavier than client verification of an RSA certificate. This is a trend that has been generally identified and our results corroborate this trend.

Analyzing Step 4. Similar to Step 3, C3 has considerably higher values for resource consumption than both C1 and C2. This indicates that signature verification is more demanding when using ECDHE_ECDSA compared to the other ciphers.

Analyzing Step 8. C1 has considerably higher values for both processing time and energy consumption compared to both C2 and C3. We know that DHE key exchange is more computationally expensive than ECDHE and this trend is quantitatively proven. Compared to all other steps, Step 8 is clearly the heaviest, which confirms that key generation factors into most of the handshake's computational cost.

Analyzing Step 9. Note that our testbed does not functionally perform client authentication. In Step 7, the client sends a blank certificate that our server is configured to always accept. The client processes the server's default verification



(a) Step 3 (b) Step 4

(c) Step 8 (d) Step 9

(e) Total

Fig. 4: This figure confirms that client verification of an ECDSA certificate requires about 3900% more energy compared to RSA. In addition, signature verification using ECDSA requires around 1700% more energy verus RSA. Furthermore, key exchange using DHE shows 300% more energy compared with ECDHE. In general, the processing overhead of C1 and C3 are 300% and 150% higher than C2, respectively.

very quickly; the results indicate that C1 has the shortest processing time, while C2 and C3 also show relatively low values.

With the exception of Step 4 for C1, the upper and lower limit of all the error bars are very close to the median value. This indicates that the processing time and energy consumption measurements for each cipher are consistent throughout all the iterations. Because there are no major outlier values, the median value accurately represents the energy consumption of each step. The cumulative time and energy measurements show that C1 is the heaviest cipher, and that C3 is heavier than C2 (i.e., in the context of the TLS protocol).

V. Related Work

The studies of [6], [10] and [17] present a thorough overview of the TLS protocol and its broad scope of applications. However, these works do not present quantitative resource consumption analysis. Potlapally et al. [14] have conducted a thorough study on the energy consumption of the TLS handshake using various cryptographic protocols, but do not present time measurements nor consider ephemeral key exchange methods (i.e., DHE and ECDHE). Nevertheless, this is one of the very few works which analyzes resource consumption of individual steps during the TLS handshake (i.e., KeyExchange and Verification). Miranda et al. [13] have analyzed the energy consumption of TLS transactions between a Nokia N95 mobile device and several popular web services over WLAN and 3G interfaces. In their study, they present energy consumption measurements of the total overhead of TLS, inclusive of both cryptographic operations as well as message transmission costs. One limitation of this study is that it does not highlight the cost of the individual steps of the protocol. Another point to consider is that this study uses a mobile device as the client which is not apt in the context of IoT networks. Our analysis is unique in this regard and is beneficial for those who want to understand the computational costs of the different steps of the TLS handshake.

VI. Conclusion

We conclude that the order of the ciphers from least to greatest energy efficiency is the following: C1, C3, C2. On average, C1 consumes 55% more energy than C3, and C3 consumes 150% more than C2 per TLS handshake. Clearly, the most efficient cipher based on the results is C2 (i.e., ECDHE using RSA). However, when choosing between RSA and ECDSA encryption, there are two considerations: RSA generally has heavier signatures than ECDSA, and ECDSA requires more computation for certificate verification. Our study proves that the energy consumption of ECDSA certificate verification considerably outweighs RSA's energy consumption for heavier signature generation. We plan on expanding this study by extending our pool of ciphers, ranking the ciphers based on level of security, and including network overhead, using various types of IoT devices.

Acknowledgment

This work has been partially supported by a research grant from Cypress Semiconductor Corporation (Grant No. CYP-001).

References

[1] H. U. Rehman, M. Asif, and M. Ahmad, "Future applications and research challenges of iot," in *International Conference on Information and Communication Technologies (ICICT)*. IEEE, 2017, pp. 68–74.

[2] B. Dezfouli, M. Radi, and O. Chipara, "Mobility-aware real-time scheduling for low-power wireless networks," in *The 35th Annual IEEE International Conference on Computer Communications (INFOCOM 2016)*. IEEE, 2016, pp. 1–9.

[3] ——, "REWIMO: A real-time and reliable low-power wireless mobile network," *ACM Transactions on Sensor Networks (TOSN)*, vol. 13, no. 3, p. 17, 2017.

[4] D. Lund, C. MacGillivray, V. Turner, and M. Morales, "Worldwide and regional internet of things (iot) 2014–2020 forecast: A virtuous circle of proven value and demand," *International Data Corporation (IDC), Tech. Rep*, vol. 1, 2014.

[5] R. Behrens and A. Ahmed, "Internet of things: An end-to-end security layer," in *20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*. IEEE, 2017, pp. 146–149.

[6] R. Mzid, M. Boujelben, H. Youssef, and M. Abid, "Adapting tls handshake protocol for heterogenous ip-based wsn using identity based cryptography," in *International Conference on Communication in Wireless Environments and Ubiquitous Systems: New Challenges (ICWUS)*. IEEE, 2010, pp. 1–8.

[7] D. E. Simos, K. Kleine, A. G. Voyiatzis, R. Kuhn, and R. Kacker, "Tls cipher suites recommendations: A combinatorial coverage measurement approach," in *International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 2016, pp. 69–73.

[8] U. Banerjee, C. Juvekar, A. Wright, A. P. Chandrakasan *et al.*, "An energy-efficient reconfigurable dtls cryptographic engine for end-to-end security in iot applications," in *International Solid-State Circuits Conference (ISSCC)*. IEEE, 2018, pp. 42–44.

[9] N. J. Al Fardan and K. G. Paterson, "Lucky thirteen: Breaking the tls and dtls record protocols," in *IEEE Symposium on Security and Privacy (SP)*. IEEE, 2013, pp. 526–540.

[10] A. K. Ranjan, V. Kumar, and M. Hussain, "Security analysis of tls authentication," in *International Conference on Contemporary Computing and Informatics (IC3I)*. IEEE, 2014, pp. 1356–1360.

[11] C. Peng, Q. Zhang, and C. Tang, "Improved tls handshake protocols using identity-based cryptography," in *International Symposium on Information Engineering and Electronic Commerce (IEEC'09)*. IEEE, 2009, pp. 135–139.

[12] M. Atighetchi, N. Soule, P. Pal, J. Loyall, A. Sinclair, and R. Grant, "Safe configuration of tls connections," in *Conference on Communications and Network Security (CNS)*. IEEE, 2013, pp. 415–422.

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

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

[15] A. Emdadi, R. Karne, and A. Wijesinha, "Implementing the tls protocol on a bare pc," in *Second International Conference on Computer Research and Development*. IEEE, 2010, pp. 293–297.

[16] S. Gueron and V. Krasnov, "Fast prime field elliptic-curve cryptography with 256-bit primes," *Journal of Cryptographic Engineering*, vol. 5, no. 2, pp. 141–151, 2015.

[17] L.-S. Huang, S. Adhikarla, D. Boneh, and C. Jackson, "An experimental study of tls forward secrecy deployments," *IEEE Internet Computing*, vol. 18, no. 6, pp. 43–51, 2014.

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

[19] T. Dierks, "The transport layer security (tls) protocol version 1.2," 2008. [Online]. Available: https://tools.ietf.org/html/rfc5246

[20] J. Salowey, A. Choudhury, and D. McGrew, "Aes galois counter mode (gcm) cipher suites for tls," Tech. Rep., 2008.

[21] Cypress Semiconductor. CYW43907: IEEE 802.11 a/b/g/n SoC with an Embedded Applications Processor. [Online]. Available: http://www.cypress.com/file/298236/download

[22] ——. CYW943907AEVAL1F Evaluation Kit. [Online]. Available: http://www.cypress.com/documentation/development-kitsboards/cyw943907aeval1f-evaluation-kit

[23] ARM Limited, "mbed tls," https://github.com/ARMmbed/mbedtls, Mar. 2017.

[24] B. Dezfouli, I. Amirtharaj, and C.-C. Li, "EMPIOT: An Energy Measurement Platform for Wireless IoT Devices," *Journal of Network and Computer Applications*, 2018.