A New Light Weight Transport Method for Secured Transmission of Data for IoT

Sainandan Bayya Venkata
Infinera India Pvt Ltd.
Bangalore, India
svenkata@infinera.com

Prabhakara Yellai
Cisco Systems Pvt. Ltd.
Bangalore, India
pyellai@cisco.com

Gaurav D Verma, Andhavarapu Lokesh,
Adithya K S and Siva Sankara Sai Sanagapati*
Department of Physics,
Sri Sathya Sai Institute of Higher Learning
Prasanthi Nilayam, India
*sivasankarasai@sssihl.edu.in

***Abstract*—With increase in deployment of the Internet of Things (IoT) devices, there has been an increasing concern regarding security issues with these IoT devices. The limited processing capabilities of these devices pose a serious limitation for the application of conventional security algorithms in these devices. This paper discusses a new Light Weight Transport Method (LWTM) which uses existing Advanced Encryption Standard-Counter (AES_CTR) and Advanced Encryption Standard-Cipher Block Chaining (AES_CBC) algorithms in a way to reduce computational time drastically for IoT applications involving large data. This method can be beneficial in many applications where the data is not entirely sensitive but at the same time the security cannot be compromised.**

***Keywords*—IoT, LWTM, AES_CTR, AES_CBC, Light Weight Algorithm**

I. Introduction

The IoT constitutes a system of interrelated physical objects with ability to communicate with each other and also with their external environment. These physical objects use embedded systems having limited power budget, limited processing capability and limited Random Access Memory (RAM). IoT has various applications in a number of areas including but not limited to health care, smart homes, smart cities, smart transport systems, smart manufacturing and other industries. With increase in the usage of Internet, the factor of security and privacy in conjunction with limited resource availability shall define the actual success of various IoT products. The security model for IoT must deliver the services such as confidentiality, integrity, authentication, Intrusion Detection System and Replay Protection to maximum possible extent.

Security solutions as shown in Table I exist at various layers in the IP stack with each solution having pros and cons. One of the many challenges these solutions need to address is regarding the limited processing capabilities of IoT devices.

Various authors have attempted header compression and optimization techniques to incorporate light weight security methods in IoT. For example, Lithe: Lightweight Secure CoAP for the Internet of Things [1] integrates traditional Datagram Transport Layer Security (DTLS) with CoAP for the IoT and in SVELTE: Real-time Intrusion Detection in the Internet of Things [2], the authors propose a system which can detect about 95% of the various active attacks like spoofing, packet altering, DoS attacks, etc. Most of the IoT research is confined in developing new Light weight Algorithms or reducing the header load, but the making the whole process light weight has not been explored enough.

TABLE I: Security Solutions

IoT Layer	IoT protocol	Security Protocol
Application	CoAP	User-Defined(CoAP/MQTT)
Transport	UDP	DTLS
Network	IPv6,RPL	IPsec, RPL
6LoWPAN	6LoWPAN	None
Data-link	IEEE 802.15.4	802.15.4 security

This paper is one such attempt to make the transport process light weight. In this paper we propose a Light Weight Transport Method (LWTM) which uses the existing Light Weight Crypto (LWC) algorithms like AES_CTR and AES_CBC [3] to optimize data encryption process without compromising the inherent security.

The proposed method works entirely at software level. It can be implemented in IoT devices which do not need hardware assistance and in which not all data need confidentiality. The applications like medical devices which scan and send the data to any desired system can derive benefits from this algorithm.

In one set of applications where the entire data does not need confidentiality but consists of parts of the data that needs encryption, the proposed LWTM algorithm can be used. In another set of applications where access to the entire clear data is required to be able to meaningfully decipher the data, the proposed LWTM algorithm can be used. In these, not encrypting the entire data without compromising on the confidentiality could save nearly 5 to 10 times of CPU time compared to full data encryption. Full encryption of data in these applications leads to unnecessary CPU computation and power drain.

Section II discusses the proposed method: it's working, algorithms used and how it is implemented. Section III shows the experimental results including full encryption, partial encryption and comparative analysis. Finally, we conclude our work in section IV.

II. Proposed Method

Instead of adding one more new Light Weight Crypto (LWC) algorithm to the list of already existing ones, a new

transport method is proposed which can use the existing LWC algorithms for encryption while making the whole encryption process cheaper in terms of CPU foot print and memory without compromising the inherent security. This method termed as Light Weight Transport Method (LWTM) uses an existing LWC, but controls the size and position of the encrypted data in the message. The position of the encrypted data in the message could either be random or be determined by the application.

A. *Working of LWTM*

The LWTM algorithm is divided into two planes,viz., control plane and data plane. The information such as key generation parameters, positions of the encrypted data blocks in the message, size of the encrypted data blocks are exchanged between the IoT device and the server over the control plane. This control plane uses a secured transport channel such as TLS or DTLS[4]. The same parameters are used by the server to decrypt the partially encrypted data. The data is encrypted as per the encrypted block size and encrypted block position and sent over unsecured transport such as TCP/UDP. A simple illustration of the LWTM algorithm is in Fig. 1.

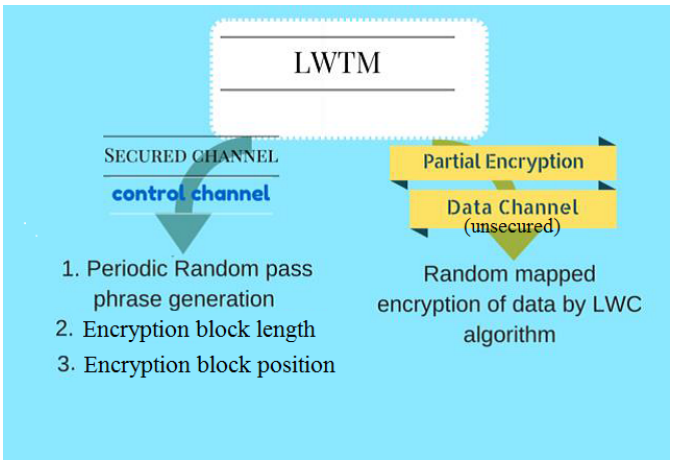

Fig. 1: Proposed Light Weight Transport Method.

Both the Data channel and the Control channel should be part of the IoT sensor device. Instead of encrypting the entire data, the LWTM adopts a distributed partial encryption model. The sender has the ability to encrypt a part of the data for every message segment. The encryption block size and the encryption block position can be changed as frequently as desired by the application or LWTM layer and communicated to the server. The algorithm starts with generating the encryption algorithm specific parameters such as keys, encryption block size, position of the encryption block by the client and communicating these to the server over a secured channel.

Then the data is fragmented into blocks of message of desired block size. Random numbers are generated according to the block size. These are mapped to the data byte locations. The Data channel and the control channel can be the same or different at layer 1. For instance, the secured control channel can use a 3G medium where the control parameters are exchanged safely whereas the data channel may be over insecured local wifi to send/receive data. Here, the IoT device

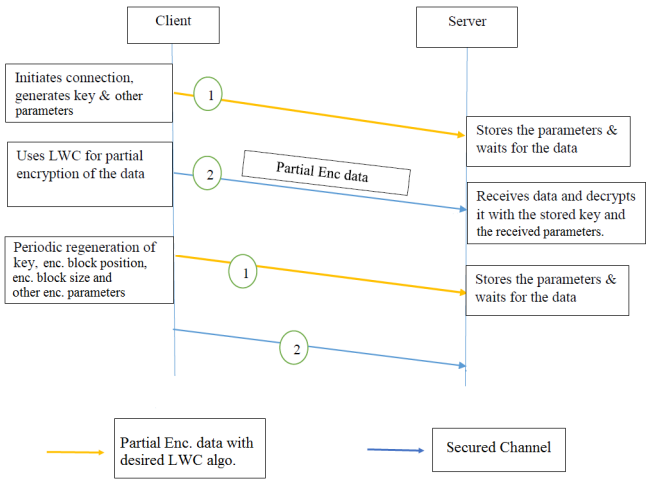

Fig. 2: Working of LWTM

and the server are connected to each other via two different physical mediums where only one of the two mediums is secure. In this case the control channel and the data channel are different at layer 1. If the IoT device and the server are connected via a single insecure medium on the other hand, then the control channel and the data channel are same at layer 1. In this case, the control parameters are encrypted and sent which ensures confidentiality. Stepwise functioning of this algorithm is described in Fig. 2.

A session is defined as the period over which the encryption parameters such as key, encryption block size, encryption block position remain the same. So, at the end of a session, these parameters parameters are changed and communicated to the server. Depending on the level of security required we can tune the session time. Shorter sessions lead to more frequent changes in these parameters leading to increased security compared to longer sessions.

At LWTM layer, the application data is fragmented into segments of *segment_size*. This *segment_size* depends on the Maximum Transmission Unit (MTU) of the data-link layer. If the application has not specified the encrypted block size and encrypted block position, the LWTM determines these parameters ensuring the uniform distribution of encrypted data blocks of desired overall amount of data to be encrypted through all the fragments of the message. Consider for example a 16 kilobyte message from an application to be sent over ethernet is fragmented into 12 segments of approximately 1366 bytes each. Consider further that 10% of the message needs to be encrypted. So the LWTM encrypts 136 bytes of data in each segment to uniformly encrypt data. The LWTM distributes this 136 bytes into two or more blocks of different sizes at random positions in each fragment. This process of partial encryption can be visualized in Fig. 3.

B. *Choice of Algorithm*

There are many well-known symmetric light weight crypto algorithms such as Tiny Encryption Algorithm (TEA) [5],

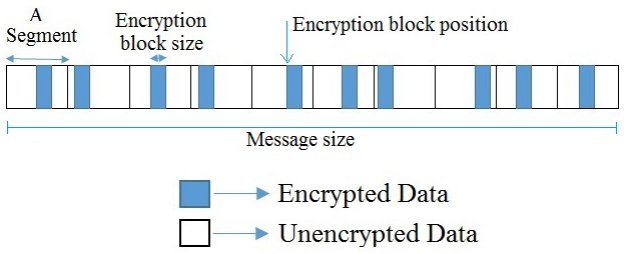

Fig. 3: Random block-encryption of a message in segments

Scalable Encryption Algorithm (SEA) [6] and PRESENT Algorithm [7] which can be considered for use. Using asymmetric key cryptography such as Elliptic Curve Cryptography (ECC) [8] on the other hand becomes a heavy process for IoT devices. The light weight crypto algorithms used in this LWTM process are AES_CTR and AES_CBC. It is advantageous to use these since the hardware of IEEE 802.15.4 also supports them. Experiments have been conducted to show that these processes are more efficient than several other available options [9]. IEEE 802.15.4 standard currently supports pre-shared key for communication. Several other advantages of choosing AES_CTR and AES_CBC are that these are symmetric key ciphers, light weight and more secured when compared to others. Typically, the CPU memory on these IoT devices is a scarce resource. Hence, implementation of AES_CTR and AES_CBC will not cause memory footprint overloads. However, the proposed LWTM algorithm is independent of the specific encryption algorithm used by the IoT devices.

C. Implementation of LWTM

The LWTM was implemented using OpenSSL security toolkit [10]. Various cryptographic operations were performed using the EVP library available in OpenSSL. The code for the LWTM method was written in C with the OpenSSL toolkit and socket programming was added to connect client and server machines.

Specifications of the client and server used in the experiment to implement this algorithm is shown in the Table II.

TABLE II: System configuration

Parameter	Value
Processor	IntelI CoreI i5-4570 CPU @ 3.20GHz
Number of Cores	4
Core Speed	3469.5 MHz

The client and the server were made to run on two different machines with same configuration and cache size. AES_CTR and AES_CBC were used in the implementation. In this implementation, a test file of size 426.6 MB was divided into small message blocks and then encrypted or decrypted according to the LWTM algorithm. The experiment was conducted by taking various values for parameters like *message_size*, encryption block length and encryption block position. We executed the program with various values for parameters like *message_size*, encryption block length and encryption block position. The *message_size* considered were 16 kB, 32 kB, 64 kB, 128 kB and 256 kB with encryption block size of each of these messages being 1 kB, 2 kB, 4 kB, 8 kB, 16 kB, 32 kB, 64 kB and 128 kB whichever possible. Appropriate random numbers were generated corresponding to each of the above *message_size* and encryption block length for positioning the encrypted blocks.

III. Results and Analysis

A. Full Encryption/Decryption

Table III is obtained by complete encryption of 436.6 MB test file after sending it from client to server and decrypting at the server side. The values shown in the table are averaged over 10 readings. This table is plotted in Fig. 4. This forms basis on which comparison between complete encryption and partial encryption can be done. It also shows that TLS with any encryption mode consumes maximum amount of CPU time for encryption and decryption. Therefore, TLS cannot be used in CPU constrained devices. The other two algorithms AES_CTR, AES_CBC are light weight algorithms and most of the IEEE 802.15.4 6LoWPAN devices come in handy with these pre-built algorithms. Hence, AES_CTR and AES_CBC were chosen as default light weight crypto algorithms. The difference in the CPU time between AES_CTR and AES_CBC is due to the design differences between the algorithms.

TABLE III: Time taken for full encryption/decryption

CPU Time (sec)	Encryption			Decryption		
	TLS	*CTR*	*CBC*	*TLS*	*CTR*	*CBC*
Minimum	2.2612	1.4145	2.0110	2.1037	1.4267	1.4002
Maximum	3.0849	1.4156	2.0232	2.7585	1.4339	1.4059
Average	2.6684	1.4150	2.0154	2.4186	1.4324	1.4026

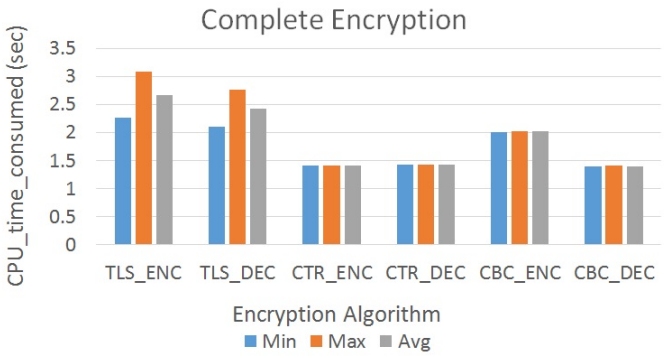

Fig. 4: Time Taken for full encryption/decryption using various crytpo algorithms

B. Partial Encryption/Decryption

Table IV to Table VII and Fig. 5 to Fig. 8 show the results of conducting the experiment for *encryption_length* 1 kB and 4 kB with *message_size* varying as 16 kB, 32 kB, 64 kB, 128 kB and 256 kB. The values shown are average of 10 readings with *encryption_length* 1 kB.

We infer by observing Fig. 5 to Fig. 8 that for AES_CTR, the time taken for encryption and decryption for a given

TABLE IV: Time taken for encryption with encryption_length 1 kB

CPU time (sec)	Encryption Mode: A⇒AES_CTR, B⇒AES_CBC									
	16 kB		*32 kB*		*64 kB*		*128 kB*		*256 kB*	
	A	**B**	**A**	**B**	**A**	**B**	**A**	**B**	**A**	**B**
Min	0.0 260	0.0 662	0.0 135	0.0 331	0.0 069	0.0 165	0.0 035	0.0 083	0.0 017	0.0 041
Max	0.0 276	0.0 665	0.0 141	0.0 334	0.0 071	0.0 170	0.0 034	0.0 083	0.0 018	0.0 042
Avg.	0.0 267	0.0 663	0.0 138	0.0 332	0.0 070	0.0 167	0.0 034	0.0 083	0.0 017	0.0 042

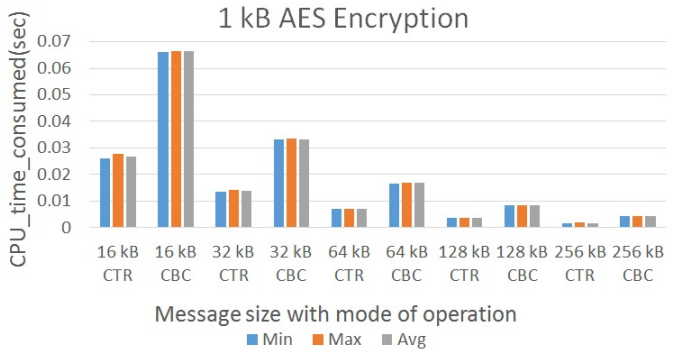

Fig. 5: Minimum, Maximum and Average time taken for encryption with encryption_length 1 kB using AES_CTR and AES_CBC for various *message_size*

TABLE V: Time taken for decryption with encryption_length 1 kB

CPU Time (sec)	Encryption Mode: A AES_CTR, B AES_CBC									
	16 kB		*32 kB*		*64 kB*		*128 kB*		*256 kB*	
	A	***B***	***A***	***B***	***A***	***B***	***A***	***B***	***A***	***B***
Min	0.0 274	0.0 314	0.0 109	0.0 137	0.0 071	0.0 081	0.0 035	0.0 039	0.0 018	0.0 020
Max	0.0 294	0.0 319	0.0 114	0.0 164	0.0 073	0.0 086	0.0 039	0.0 042	0.0 019	0.0 021
Avg.	0.0 283	0.0 316	0.0 112	0.0 153	0.0 071	0.0 083	0.0 037	0.0 040	0.0 018	0.0 021

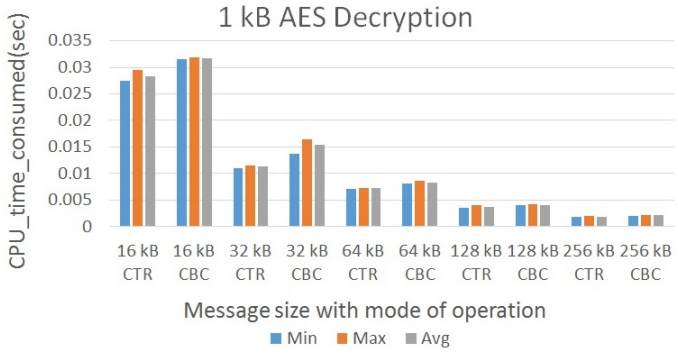

Fig. 6: Minimum, Maximum and Average time taken for decryption with *encryption_length* 1 kB using AES_CTR and AES_CBC for various *message_size*

encryption_length is almost the same. Whereas, for AES_CTR, decryption takes less time than encryption because of the chain mode of operation in CBC. In each of these graphs the CPU time falls proportionately as the *message_size* increases for a given encryption. This is expected since the net amount of data encrypted/decrypted decreases as the message size increases. For example, in Table VIII and Fig. 9, encryption with *message_size* 16 kB takes more time compared to *message_size* of 128 kB or 256 kB. This is because with 16 kB *message_size* and 4 kB *encryption_length*, as much as one fourth of the entire data has to be encrypted/decrypted. We anticipate the CPU time taken to become half as *message_size* doubles and in fact this is seen in the experimental values obtained by applying the algorithm. Tables and Graphs corresponding to *encryption_length* 8 kB, 16 kB, 32 kB, 64 kB and 128 kB were also obtained and were found to follow similar trend as seen in Table IV to Table VII and Fig. 5 to Fig. 8.

C. Comparative Analysis

It is found that the AES_CTR encryption algorithm without the influence of proposed model gives 0.215sec for all the *message_size* and when we use AES_CTR with the proposed algorithm the encryption time varies with *encryption_length* and *message_size*. It can also be seen from Table VIII and Fig. 9 that when *encryption_length* is half of the *message_size*, the time taken for encryption is exactly half or slightly more than the time taken for encryption of entire *message_size*. In the same way, the encryption time is proportional to the ratio of *encryption_length* to *message_size*, which confirms that the proposed algorithm is on good terms. Table IX and Fig. 10 show similar results corresponding to AES_CBC encryption algorithm.

TABLE VI: Time taken for encryption with encryption_length 4 kB

CPU Time (sec)	Encryption Mode: A AES_CTR, B AES_CBC									
	16 kB		*32 kB*		*64 kB*		*128 kB*		*256 kB*	
	A	***B***	***A***	***B***	***A***	***B***	***A***	***B***	***A***	***B***
Min	0.0 704	0.2 361	0.0 353	0.1 184	0.0 175	0.0 591	0.0 089	0.0 295	0.0 026	0.0 147
Max	0.0 725	0.2 371	0.0 368	0.1 185	0.0 179	0.0 594	0.0 094	0.0 296	0.0 028	0.0 149
Avg.	0.0 715	0.2 367	0.0 359	0.1 184	0.0 177	0.0 593	0.0 091	0.0 295	0.0 027	0.0 148

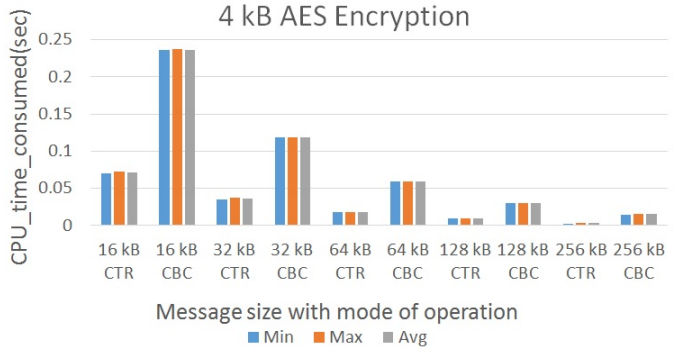

Fig. 7: Minimum, Maximum and Average time taken for encryption with *encrytption_length* 4 kB using AES_CTR and AES_CBC for various *message_size*

A simple calculation can be done to determine the percentage of encryption and the percentage of CPU saving from the data generated in this experiment. This is done in Table X with two message size values of 16 kB and 32 kB. The same

TABLE VII: Time taken for decryption with encryption_length 4kB

CPU Time (sec)	Encryption Mode: A AES_CTR, B AES_CBC									
	16 kB		*32 kB*		*64 kB*		*128 kB*		*256 kB*	
	A	*B*	*A*	*B*	*A*	*B*	*A*	*B*	*A*	*B*
Min	0.0 722	0.0 716	0.0 361	0.0 360	0.0 180	0.0 091	0.0 088	0.0 027	0.0 027	0.0 044
Max	0.0 748	0.0 736	0.0 367	0.0 364	0.0 184	0.0 183	0.00 093	0.0 091	0.0 091	0.0 047
Avg.	0.0 738	0.0 729	0.0 364	0.0 363	0.0 182	0.0 181	0.0 092	0.0 089	0.0 028	0.0 045

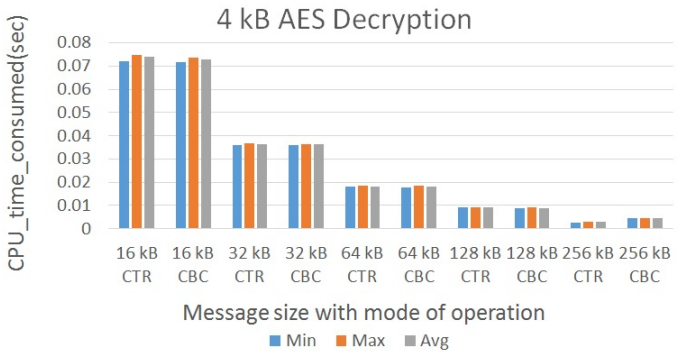

Fig. 8: Minimum, Maximum and Average time taken for decryption with *encryption_length* 4 kB using AES_CTR and AES_CBC for various *message_size*

is shown for all the values used in this experiment in Figure 11.

The CPU consumption and the security aspects are the key parameters which need to be optimized for any given IoT application. Fig. 11 shows the wide range of possibilities available for the user to choose with each choice leading to saving of a specific amount of computational time. We may heuristically claim that 25 % of encryption leading to 75 % CPU saving shall be a good choice for any application in general. This optimum percentage of encryption would be distributed over the entire data and hence depending on the link layer fragmentation each layer 2 packet would have some percentage of encrypted data. This shall make sure that the packet going out of link layer would have some part encrypted. However, if the user wishes to have very high CPU saving time say by choosing 256 kB *message_size* with 1 kB *encryption_length*, user would still be secure without having very high visibility. This is because, due to the link layer impact at least one block would be encrypted for every 1500 bytes i.e., MTU size of Ethernet. From this, we can say that the proposed LWTM has achieved its objective of saving CPU consumption without compromising on security.

IV. Conclusion

The objective of the paper is to analyze the existing light weight crypto algorithms, how data is processed in other layers of the IoT device and to come up with a secured Light Weight Transport Method for IoT. It is shown that the proposed algorithm can be used effectively to optimize the parameters like CPU consumption, fragmentation of the data and latency.

The proposed algorithm has two channels. One is control channel and the other is data channel. Before encryption, the

TABLE VIII: Comparison of full and partial encryption in AES_CTR mode

Seg. size ↓	*Full*	*1 kB*	*2 kB*	*4 kB*	*8 kB*	*16 kB*	*32 kB*	*64 kB*	*128 kB*
16 kB	0.215	0.0 267	0.0 414	0.0 715	0.1 322	NA	NA	NA	NA
32 kB	0.215	0.0 138	0.0 209	0.0 359	0.0 660	0.1 256	NA	NA	NA
64 kB	0.215	0.0 070	0.0 108	0.0 177	0.0 329	0.0 632	0.1 231	NA	NA
128 kB	0.215	0.0 034	0.0 053	0.0 091	0.0 164	0.0 315	0.0 613	0.1 204	NA
256 kB	0.215	0.0 017	0.0 026	0.0 027	0.0 082	0.0 158	0.0 306	0.0 609	0.1 207

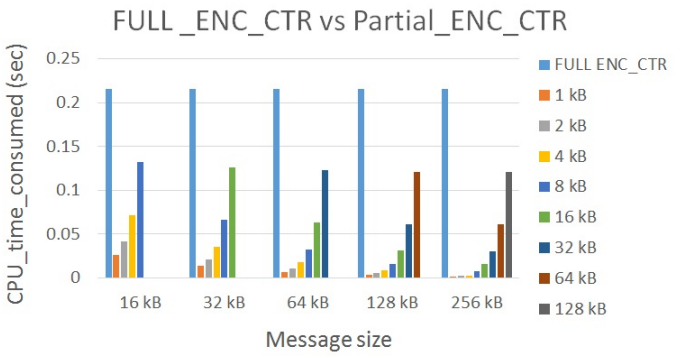

Fig. 9: Comparison between full encryption and partial encryption with the proposed method in AES_CTR mode

parameters like random passphrase for key generation, random numbers, *encryption_length*, and *message_size* are generated by the algorithm on the client side and the same are transmitted to the server over control channel in a secured manner using DTLS or any other desired security suite. The data is encrypted as per the parameters and sent through data channel using TCP/UDP. Server uses these parameters to decrypt the partially encrypted data from the data channel. Although only a part of the data has been encrypted, for many applications this would suffice. The algorithm can be modelled in a way that assures encryption of specific sensitive parts of data. Also, partial encryption alters the statistics of the entire data making it less prone to attacks based on statistics or data-frequency analysis.

Hence, the security concern of IoT has been elegantly addressed in this proposed method as the adversary may not know the important encryption parameters such as random numbers, *encryption_length*, random passphrase, etc. and therefore cannot differentiate between the encrypted and the unencrypted plain text. A random part of each segment in the data to be transmitted being encrypted in the manner explained in this paper can thus solve the security issues with less computational capabilities in specific IoT devices.

The future work involves developing an Application Program Interface (API) for the IoT device with Contiki OS [11] or any other platform which directly implements the algorithm into the required environment without using any other libraries. This API can be made to have its own functions to bind, connect and forward the traffic. Users willing to implement LWTM can directly use the API.

TABLE IX: Comparison of full and partial encryption in AES_CBC mode

Seg. size ↓	*Full*	*1 kB*	*2 kB*	*4 kB*	*8 kB*	*16 kB*	*32 kB*	*64 kB*	*128 kB*
16 kB	0.815	0.0 663	0.1 232	0.2 367	0.4 623	NA	NA	NA	NA
32 kB	0.815	0.0 332	0.0 616	0.1 184	0.2 315	0.4 584	NA	NA	NA
64 kB	0.815	0.0 167	0.0 308	0.0 593	0.1 160	0.2 290	0.4 550	NA	NA
128 kB	0.815	0.0 083	0.0 155	0.0 295	0.0 579	0.1 142	0.2 274	0.4 552	NA
256 kB	0.815	0.0 042	0.0 077	0.0 148	0.0 290	0.0 572	0.1 136	0.2 271	0.4 542

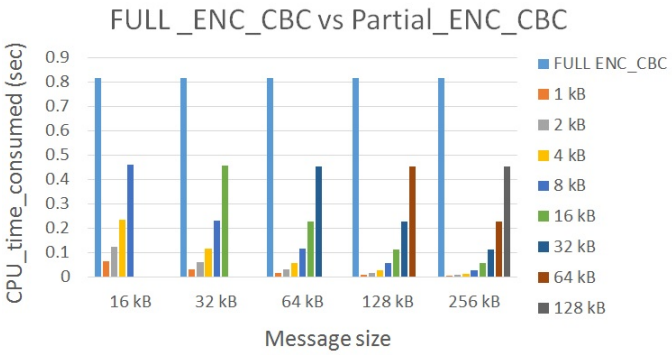

Fig. 10: Comparison between full encryption and partial encryption with the proposed method in AES_CBC mode

TABLE X: Percentage of Encryption and CPU savings using LWTM

Seg. size	*Percentage of*	*Encryption Length*				
		1 kB	**2 kB**	**4 kB**	**8 kB**	**16 kB**
16 kB	Encryption	6.25	12.5	25	50	NA
	CPU saving	93.75	87.5	75	50	NA
32 kB	Encryption	3.11	6.22	12.5	25	50
	CPU saving	96.8	93.7	87.5	75	50

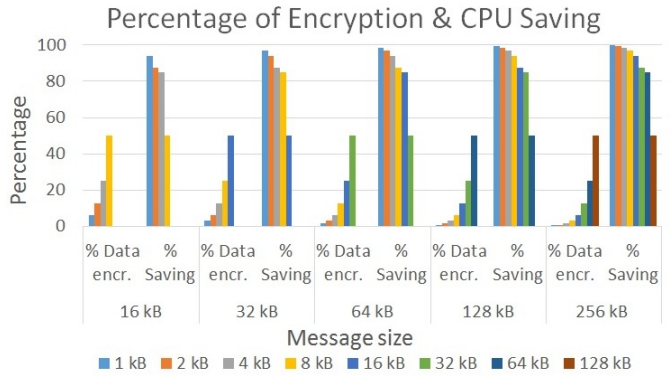

Fig. 11: A graphical visualisation of percentage encryption and CPU saving using LWTM

REFERENCES

[1] Raza, S., Shafagh, H., Hewage, K., Hummen, R., & Voigt, T. (2013). Lithe: Lightweight secure CoAP for the internet of things. IEEE Sensors Journal, 13(10), 3711-3720.

[2] Raza, S., Wallgren, L., & Voigt, T. (2013). SVELTE: Real-time intrusion detection in the Internet of Things. Ad hoc networks, 11(8), 2661-2674.

[3] Lipmaa, H., Wagner, D., & Rogaway, P. (2000). Comments to NIST concerning AES modes of operation: CTR-mode encryption.

[4] Raza, S., Trabalza, D., & Voigt, T. (2012, May). 6LoWPAN compressed DTLS for CoAP. In 2012 IEEE 8th International Conference on Distributed Computing in Sensor Systems (pp. 287-289). IEEE.

[5] Wheeler, D., & Needham, R. (1995). TEA, a tiny encryption algorithm. In Fast Software Encryption (pp. 363-366). Springer Berlin/Heidelberg.

[6] Standaert, F. X., Piret, G., Gershenfeld, N., & Quisquater, J. J. (2006, April). SEA: A scalable encryption algorithm for small embedded applications. In International Conference on Smart Card Research and Advanced Applications (pp. 222-236). Springer Berlin Heidelberg.

[7] Bogdanov, A., Knudsen, L. R., Leander, G., Paar, C., Poschmann, A., Robshaw, M. J., ... & Vikkelsoe, C. (2007, September). PRESENT: An ultra-lightweight block cipher. In International Workshop on Cryptographic Hardware and Embedded Systems (pp. 450-466). Springer Berlin Heidelberg.

[8] Koblitz, N. (1987). Elliptic curve cryptosystems. Mathematics of computation, 48(177), 203-209.

[9] https://www.cryptopp.com/benchmarks.html

[10] OpenSSL, "OpenSSL About," p. 1, 2013. [Online],Available:https://www.openssl.org/support

[11] Dunkels, A., Gronvall, B., & Voigt, T. (2004, November). Contiki-a lightweight and flexible operating system for tiny networked sensors. In Local Computer Networks, 2004. 29th Annual IEEE International Conference on (pp. 455-462). IEEE.