electronics

MDPI

Article

Lightweight Cryptography for the Encryption of Data Communication of IoT Devices

Ivan Sokol *, Peter Hubinský and Ľuboš Chovanec

Faculty of Electrical Engineering and Information Technology, Slovak Technical University in Bratislava, Ilkovičova 3, 81219 Bratislava, Slovakia; peter.hubinsky@stuba.sk (P.H.); lubos.chovanec@stuba.sk (Ľ.C.)
* Correspondence: ivan.sokol@stuba.sk

Abstract: We are at the beginning of the age of the Internet of things. Soon, we will be surrounded by smart homes, cities, and infrastructure. To achieve this vision, millions of devices will have to be able to communicate with each other. The demands for communication channels will increase significantly. An increasing amount of data will be transmitted with a requirement of minimal delay. The capacities of transmission systems can be quickly depleted. Building new communication channels is very time consuming but also financially demanding. To maximize existing infrastructure, we should pay attention today to the issue of transmitted data. One of the ways is to focus attention on reducing the volume of transmitted data. In this paper, we present a method of reducing the volume of data transmission between a server and an IoT device, focusing on the bandwidth, transmission security, and system resources of the IoT device. The required reduction is achieved by data compression and replacing the SSL/TLS cryptographic protocol with lightweight cryptography based on the Vernam cipher principle. The original SSL/TLS protocol is still used for device management needs only.

Keywords: internet of things (IoT); data compression; SSL/TLS cryptographic protocol; lightweight cryptography (LWC)

Citation: Sokol, I.; Hubinský, P.; Chovanec, Ľ. Lightweight Cryptography for the Encryption of Data Communication of IoT Devices. *Electronics* **2021**, *10*, 2567. https://doi.org/10.3390/electronics10212567

Academic Editors: Khaled Rabie, Xingwang Li, Waleed Ejaz and Laith Farhan

Received: 17 September 2021
Accepted: 19 October 2021
Published: 20 October 2021

1. Introduction

According to the Gartner [1] forecasts from 2017, 20 billion IoT devices will be connected to the Internet in 2020. In 2019, SAP [2] estimated the number of connected devices in 2020 was 30 billion, and its forecast predicted about 75 billion IoT devices in the global network by 2025. Such many numbers of devices bring increased requirements for the width of the transmission channel. The company AtomBeam [3] estimated that in 2025, the volume of data transmitted by IoT devices reaches 90 zettabytes. Today, the volume of transmitted data is at a level of 30 Zettabytes [3].

The way to deal with potential communication problems is to optimize data transmission. One of the ways to achieve this is to reduce the amount of transferred data.

In addition to reducing the volume of transmitted data, we will reduce the device's energy consumption of data processing processes. Energy saving plays an important role in battery-powered devices.

This paper focuses on the shortcomings of the SSL/TLS protocol and its replacement by lightweight cryptography (LWC) [4], which is more suitable for IoT technologies. The issue of lightweight cryptography is intensively addressed by The National Institute of Standards and Technology (NIST) [4].

We can summarize the shortcomings of the SSL/TLS cryptographic protocol from the point of view of IoT technology into the following points:

Inefficient communication—the volume of transmitted data during establishing a connection significantly exceeds the volume of data generated by the device. Depending on the protocol version, up to 4 kilobytes of data can be transferred during the process of establishing a connection. The size of the message transmitted by the device is at the level of hundreds of bytes.

The delay—which occurs when establishing a connection, can reach tens of milliseconds [5]. In adverse circumstances, this value can be up to 150 milliseconds.

Energy intensity—the transmitted data are encrypted with an algorithm appointed between the device and the remote server. Encryption places a high demand on system resources, and the used algorithms have high energy requirements.

2. Solution Design

Before we explain our proposed solution, we briefly look at the data processing method on the device side.

The message generated by the IoT device is often compressed in the first step before it is sent to the server with a target to reduce the amount of transmitting data. The device's algorithm encrypts the data to increase communication security in the next step (protection against SSL inspection). Subsequently, the sent data are encrypted by an algorithm appointed between the device and the server in the process of establishing a connection (handshake). As we can see in the case of encryption, the data are sent twice through a computationally and, therefore, energy-intensive process (double encryption).

The proposed solution offers to replace double data encryption in the communication process with simpler, less energy-efficient, and time-consuming algorithms while maintaining the security of data transmission.

3. Lightweight Cryptography

Using LWC for IoT devices reduces hardware requirements as well as power consumption. Today, there are a massive number of LWC algorithms. The authors of many professional articles [6–8] pay attention to them. For example, in the competition organized by NIST, 57 projects were entered. In March 2021, 10 finalists were selected. [9] Examining these algorithms, we find that most work on the cyclic cipher principle. We decided to go the other way.

One-Time Pad

Our proposal returns to an idea that is more than a hundred years old—Vernam's cipher. This cipher uses the one-time pad [10] method, which cannot be broken if the required conditions are met. In this case, the original message is encrypted with XOR and a key. For encryption to be secure, the key must meet the following necessary conditions:

1. The key must be a random string of characters;
2. The key must be at least as long as the original text;
3. The key must be secret;
4. The key must never be used again as a whole or in part.

The key must be a random string of characters

The key used to encrypt the transmitted message is generated by the application on the server side. Today, we can use a wide range of random chain generators for this purpose. As will be explained later in the Implementation section, the quality of the key does not play an important role.

The key must be at least as long as the original text

We can determine the minimum key length based on the size of the transmitted message. In most cases, the length of the message will be in the order of several hundred bytes. Therefore, a key with a length of 1024 bytes will be enough.

The key must be secret

The key is transferred from the server to the device via the encrypted SSL/TLS communication channel. Assuming the attacker fails to enter this communication, we can say that the requirement is met. The key is known only by the device and the application.

The key must never be used again as a whole or in part

This request poses a severe problem for us. If we do not want to change the key after each message transmission, we must find a way to solve this problem with available data. We will address this issue in the next section.

4. Data Preparation

The following steps aim to meet the fourth requirement, "The key must never be used again as a whole or in part", as closely as possible. We will encrypt the message with the key (Master key) that the device receives from the server during the initialization process. At this moment, we assume we have the required key.

The message is encrypted by the Master key using the XOR function. The following applies to the encrypted message:

$$message \oplus key = result \tag{1}$$

Without affecting the result, we can rewrite it in the form:

$$key \oplus message = result \tag{2}$$

To understand the proposed encryption process, we use a short message. For example:

This message will be encrypted.

We use the introduction of this paragraph as a Master key.

The aim of the following steps is to get as close as possible to meeting the fourth requirement.

The encrypted message is 31 characters long, and the Master key contains 96 characters. In this case, we can use 96 ways to encrypt the message. We must only change the initial position of the key's first character in the Master key for the goal.

The requirement for unbreakable encryption is still not met. In our example, part of the Master key is repeated in the encryption process every time. At the same time, we need to realize that we worked with a static message. In contrast, the message generated by the IoT device is dynamic.

In the explanation process, we work with a simple message in JSON format.

```
{
    "type": "light",
    "name": "corridor",
    "id": "154ded36",
    "status": "off",
    "date": "2021-08-23 09:41:15"
}
```

This message contains three static and two dynamic variables. Static variables have a constant value at each transfer; the dynamic variables can change their value.

As is seen in Figure 1, the data are compressed before sending. For comparison, we use the method described in the article "Internet of things—nonstandard data compression" [11]. A similar nonstandard data compression method used for IoT devices comes from the company AtomBeam [12].

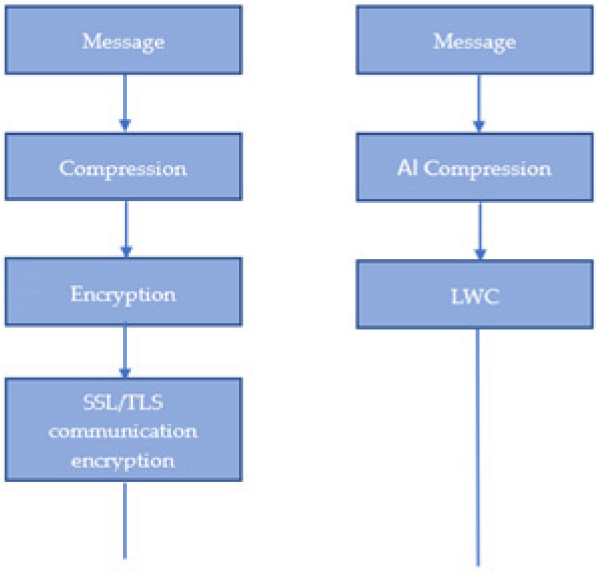

Figure 1. Standard SSL/TLS and suggested solution.

In this paper, we work with our algorithm. The result of the compression process on the application side is a device dictionary (see Table 1).

Table 1. Device dictionary.

Value	Subst. 1	Type	Add Data (Length)
"id": "12f8-0860-535d-f00b"	#ad04	fixed	0
"type": "light"	#11a7	fixed	0
"name": "corridor"	#4eac	fixed	0
"status": "on"	#230a	dynamic	0
"status": "off"	#ee01	dynamic	0
"date": "2021-08-23 09:41:15"	#c7aa	dynamic	4

Each variable always consists of a fixed part. This part of the variable in Table 1, named Subst.1, has a fixed length. In our case, the size of the substitution is 2 bytes. If a variable on the device side changes its value, this variable is transmitted in the form

Subst.1 {add data}

The length of the Add data depends on the type of dynamic variable. The method of defining the size of added data is described in a separate article [11]. In the case of a datum, 4 bytes are used to transmit the necessary information.

The message in compressed form is

ad04 11a7 4eac ee01 c7aa {4-byte data}

The original message in JSON format compressed by this algorithm reduces its size from about 105 characters to 14. The size of the resulting message is always constant. In this case, it is about 15% of the original size.

For comparison, we present the data structure of the message that SIEA [13] uses in the project monitoring system of energy efficiency.

```
{
    "RecTimestamp": "2021-09-1T01:02:03Z",
    "GPSCoordinates": [19.146192, 48.736277, 362.0],
    "LocalTemperature": −12.34,
    "BuildingId": 1234,
```

```
    "IntervalStart": "2021-09-11T01T00:00:00Z",
    "IntervalEnd": "2021-09-11T01T23:59:59Z",
    "EnergyFormId": 3561,
    "UnitCode": "kWh",
    "TotalConsumption": 150.1234,
    "HeatingConsumption": 10.1234,
    "HeatWaterConsumption": 20,
    "AirConditioningConsumption": 30,
    "ElectricityConsumption": 40,
    "OtherConsumption": 50
}
```


This report works with dynamic variables as a matter of priority (see Table 2).

Table 2. SIEA device dictionary.

Variable	Subst.1	Type	Add Data (Length)
BuildingId	#1e33	dynamic	8
RecTimestamp	#1d84	dynamic	4
IntervalStart	#d540	dynamic	4
IntervalEnd	#11a0	dynamic	4
LocalTemperature	#ad34	dynamic	4
GPSCoordinates0	#2aaa	dynamic	4
GPSCoordinates1	#e2ad	dynamic	4
GPSCoordinates2	#c56a	dynamic	4
EnergyFormId	#0e34	fixed	0
UnitCode	#445a	fixed	0
AirConditioningConsumption	#5aed	dynamic	8
ElectricityConsumption	#dde1	dynamic	8
HeatingConsumption	#36ee	dynamic	8
HeatWaterConsumption	#00a1	dynamic	8
OtherConsumption	#cc90	dynamic	8
TotalConsumption	#88a9	dynamic	8

The length of the message is around 440 characters. Based on the dictionary, we can rewrite it in the manner Subst.1 {add data} into a message with a constant length of 112 bytes, regardless of how the length of the original message changes. The data are reduced to 25% of their initial size.

The message in this form is almost useless for a potential attacker without knowledge of the dictionary. However, in the case of long-term monitoring of the communication, the attacker would be able to manipulate these data, albeit to a limited extent.

To eliminate the potential risk of misuse of transmitted data, we design two additional methods for data modification in the presented solution. For this reason, we designed two processes were named ***Alias*** and ***Shaker***. Process Alias runs on the application server. Shaker runs on the device side.

4.1. Alias

The purpose of this step extends the method of how the message writes in the compressed format. When choosing a suitable method, we focused on simple implementation as well as minimize energy consumption. We were inspired by the i18n [14] standard.

That led us to create the Alias Dictionary. Each variable can then be written through n substitutions (see Table 3).

Table 3. Alias dictionary.

Value	Subst. 1	Subst. 2	Subst. 3	Type	Add Data (Length)
"id": "12f8-0860-535d-f00b"	#ad04	#34e1	#1679	fixed	0
"type": "light"	#11a7	#aae4	#edc7	fixed	0
"name": "corridor"	#4eac	#001a	#98ef	fixed	0
"status": "on"	#230a	#2301	#e540	dynamic	0
"status": "off"	#ee01	#56ad	#bca3	dynamic	0
"date": "2021-08-23 09:41:15"	#c7aa	#65d0	#1aac	dynamic	4

Using the original dictionary, we had only one option to compose the sent message. If we use the Alias dictionary, the number of message creation options is defined by the formula

$$P_a = n^m \tag{3}$$

P_a the number of message generation options using the Alias method.
n number of aliases.
m number of message variables.

In our example, if we use three substitutions for each variable, we have 243 options for writing a message. The following examples still present the same message.

1679 aae4 001a 56ad 1aac {4-byte data}
34e1 edc7 4eac bca3 65d0 {4-byte data}

As we can see, the number of options for writing a message has an exponential course. More complex device can easily acquire thousands of message writing forms.

4.2. Shaker

Using a nonstandard compression method allowed us to eliminate the need to transfer the dictionary in the data transfer process. Only the compressed message is transmitted.

If we use standard methods for data compression, the final message is written in a strictly defined structure. The proposed Shaker method completely abandons this convention.

We send data in an imprecisely defined structure and in random order. Imagine that we insert individual Aliases into a cup and shake the cup (hence the name shaker). Consequently, we dump them out on the table and send in the order of how they fell out of the cup.

The number of ways for message generating is defined by a formula

$$P_s = m! \tag{4}$$

P_s the number of message generation options using the Shaker method.
m number of message variables.

For our five variables, we get 120 ways to compile the report.

The total number of the forms to write a message using the alias and shaker method is defined by the formula

$$P = P_a * P_s \tag{5}$$

P total number of message generation options using the Alias and Shaker method.

Now, we have disposed of 29,160 forms of the message.

In the encryption process, we work with two dynamic variables. Since formulas (1) and (2) apply, we can consider the generated message as the key and the used part of the

Master key as the message. At this point, we have come close enough to meet the condition: the key must never be used in whole or in part.

5. Message Encryption

The last task awaits us—encrypt the message. For this purpose, we will use the Master key and Secure key. The following line presents the part of the Master key.

a2:70:a2:d0:9f:42:ae:5b:97:5a:9f:94:04:70:ec:3f:59:d9:b7:cc:66:8b:7a:33:

We have a compressed message as the input in the form

34:e1:ed:c7:4e:ac:bc:a3:65:d0:01:01:01:01

In the first step, we generate the initial position of the sub-string from the Master key which will be used for message encryption. In the next step, the message is encoded by this sub-string.

34:e1:ed:c7:4e:ac:bc:a3:65:d0:01:01:01:01 message
97:5a:9f:94:04:70:ec:3f:59:d9:b7:cc:66:8b sub-string
a3:bb:72:53:4a:dc:50:9c:3c:09:b6:cd:67:8a result

We can decrypt an encrypted message as such by using the brute-force method only. The following formula gives the count of possibilities of decrypting this message

$$P_x = 256^{(L*m+D)} \tag{6}$$

P_x the number of possibilities for breaking message encryption using the brute-force method.
L length of the compressed message in bytes.
m number of message variables.
D number of bytes of dynamic variables.
In our example, we get the value

$$256^{(2*5+4)} = 5.19 * 10^{33} \tag{7}$$

If we want to encrypt this message, we must know the start position of the key. For this reason, we must supplement this information into the message (e.g., #01:ad). This information is placed at the beginning of the encrypted message.

01:ad:a3:bb:72:53:4a:dc:50:9c:3c:09:b6:cd:67:8a position + result

Now, we encrypt the message with the Secure key. The device generates this key in the initialization process.

01:ad:a3:bb:72:53:4a:dc:50:9c:3c:09:b6:cd:67:8a position + result
a4:a4:a4:a4:a4:a4:a4:a4:a4:a4:a4:a4:a4:a4:a4:a4 Secure key
5a:9a:70:bf:76:57:4e:d8:54:98:38:0d:b2:c9:63:8e result

We have the final form of the message, which we can send to the server. The brute-force resistance of this message increased in value

$$P_x = 256^{(L*m+D+O)} \tag{8}$$

P_x the number of possibilities for breaking message encryption using the brute-force method.
O two bytes that define the initial position of the character in the Master key.
In the numerical values

$$256^{(2*5+4+2)} = 3.40 * 10^{38} \tag{9}$$

6. Implementation

Security is an integral part of data processing. The final level of security is a question of the used algorithms and their implementation.

6.1. Initialization

The described method has been tested under laboratory conditions, as it is not supported by existing equipment. Communication takes place according to the following scenario:

——> The server creates an SSL communication channel;
——> The server sends a dictionary to the device;
——> The server sends Master key to the device;
<—- The device sends the Secure key to the server;
——> The server confirms Secure key to the device;
——> The server terminates SSL communication.

After successful initialization, the device can communicate with the application without building an encrypted connection.

6.2. Message Encryption

The encryption process is described in the ***Message encryption*** chapter. The message created in this way is sent to the server where it is decrypted.

6.3. Decrypting a Message

The application identifies the device and assigns it a valid Secure key, Master key, and device dictionary (identification of the device is not part of this article).

The Secure key was created during the initialization process and is only known to the device and the application. We use the Secure key for the message decrypt at the first level.

In the next step, the algorithm takes the first 2 bytes of the message. These 2 bytes determine the initial position of the character in the Master key. Based on this information, we decrypt the received message at the second level. The result of this operation is a message in compressed form. In the last step, we use the device dictionary to convert the message into the application's internal data format.

6.4. Security of Transmitted Data

The brute-force method is the only way to attack the transmitted data. For the attack to be successful, the data sent must meet the requirements of the application. The presented solution brings a four-layer data protection structure:

Unique identifier—a string that is associated with a device (UID). UID implementation is an application task and is therefore not part of this article.

Secure key—a unique string created during the initialization process by the device and sent to the server.

Master key—a random string generated on the application side and sent to the device during the initialization process.

Device dictionary—a dictionary created on the application side and represents a binary form of a compressed message.

From a security point of view, the last step of data decryption is important for us. This step is a reverse translation of compressed messages to the application form using the device dictionary. If the reverse translation does not match in just one case, we know that the data have been manipulated. An application can respond to such a message by ignoring the message or addressing the device using the standard method over an SSL channel. The dictionary and Master key will then be exchanged. After a successful exchange, the communication returns to the proposed mode.

7. Conclusions

The presented solution focuses on the transmission of short messages, which in the original JSON form, have a size of 100 bytes (up to 500 characters). Using a nonstandard compression method can reduce their size to 20% of their original size. By eliminating the SSL/TSL protocol from the data transfer process as described, reducing the total amount of data transferred to less than 10% of the original value is possible. The main part of the savings is data related to building an encrypted connection.

The expected mass emergence of IoT technologies will place ever-increasing demands on the width of the transmission channel, on the one hand, and the minimization of energy consumption, on the other hand. The described solution offers a way to solve potential problems. It deals with the reduction in transmitting data, the acceleration of communication, and the issue of energy consumption with priority on the device side. Significant attention is also paid to the security of transmitted data.

Author Contributions: Conceptualization, I.S.; Supervision, P.H.; Validation, Ľ.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Acknowledgments: This work has been supported by Grant VEGA 1/0754/19 of the Slovak Scientific Grant Agency.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Gartner. 8.4 Billion Connected Things Will Be in Use 2017 | Gartner. 2017. Available online: https://www.gartner.com/en/newsroom/press-releases/2017-02-07-gartner-says-8-billion-connected-things-will-be-in-use-in-2017-up-31-percent-from-2016 (accessed on 17 October 2021).
2. Mauchline, S.; Teerlink, M.; Manohar, S. IoT-Prognosen: 75 Milliarden vernetze Geräte | SAP News Center. SAP News Center. 2019. Available online: https://news.sap.com/germany/2019/10/iot-chance-moeglichkeiten/ (accessed on 17 October 2021).
3. StartEngine. AtomBeam Radically Efficient Software for IoT. 2021. Available online: https://www.startengine.com/atombeam/ (accessed on 17 October 2021).
4. Csrc.nist.gov. Lightweight Cryptography | CSRC. 2015. Available online: https://csrc.nist.gov/Projects/lightweight-cryptography (accessed on 17 October 2021).
5. Shamsher, U.; Jun, W.X. Evaluating Web-latency reducing Protocols in Mobile Invironments. Diva-portal.org. 2013. Available online: https://www.diva-portal.org/smash/get/diva2:829633/FULLTEXT01.pdf (accessed on 17 October 2021).
6. Thakor, V.A.; Razzaque, M.A.; Khandaker, M.R.A. Lightweight Cryptography Algorithms for Resource-Constrained IoT Devices: A Review, Comparison and Research Opportunities. ieeexplore.ieee.org. 2021. Available online: https://ieeexplore.ieee.org/abstract/document/9328432 (accessed on 17 October 2021).
7. Usman, M. Lightweight Encryption for the Low Powered IoT Devices. arXiv.org. 2020. Available online: https://arxiv.org/abs/2012.00193 (accessed on 17 October 2021).
8. Dutta, I.K.; Ghosh, B.; Bayoumi, M. Lightweight Cryptography for Internet of Insecure Things: A Survey. Ieeexplore.ieee.org. 2019. Available online: https://ieeexplore.ieee.org/document/8666557 (accessed on 17 October 2021).
9. Csrc.nist.gov. Lightweight Cryptography | Finalists. 2021. Available online: https://csrc.nist.gov/Projects/lightweight-cryptography/finalists (accessed on 17 October 2021).
10. Boneh, D. Online Cryptography Course. Crypto.stanford.edu. Available online: https://crypto.stanford.edu/~{}dabo/courses/OnlineCrypto/slides/02-stream-v2-annotated.pdf (accessed on 17 October 2021).
11. Sokol, I.; Hubinský, P. Internet of things-nonstandard data compression. *J. Electr. Eng.* **2020**, *71*, 281–285. Available online: 10.2478/jee-2020-0038 (accessed on 17 October 2021). [CrossRef]
12. AtomBeam. Compaction vs. Compression—AtomBeam. 2019. Available online: https://atombeamtech.com/compaction-vs-compression/ (accessed on 17 October 2021).
13. SIEA API. SIEA API. 2020. Available online: https://documenter.getpostman.com/view/2447371/SzRuWr8t?version=latest (accessed on 17 October 2021).
14. W3.org. Localization vs. Internationalization. 2005. Available online: https://www.w3.org/International/questions/qa-i18n (accessed on 17 October 2021).