Implementing Fully Homomorphic Encryption with BFV Scheme in Internet of Things Network

Gede Satya Adi Dharma, 13217016
Program Studi Sistem dan Teknologi Informasi/Informatika/Teknik Elektro
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): satyaadidharma2000@gmail.com

and almost all devices must be physically held and controlled in order to use it. However, that would be a problem if the devices's position is out of range. This problem can be solved by controlling the devices using the internet network so the device can be controlled by simply pressing a button provided or doing a similar little action. A devices that can connect to the internet network are called IoT device. In sending and receiving data by an IoT device, the security must be guaranteed so the data can be maintained and not used by irresponsible people. This experiment will show about implementing Fully Homomorphic Encryption on an IoT device.

Keywords—IoT, Security, Internet, Encryption

I. INTRODUCTION

IoT devices are pieces of hardware that can transmit data over the internet or other networks. By using internet networks and storage data provider, such as cloud, the data are accessible. All this data on IoT device must be safe and the transmission process must be fast so that the users of IoT device are comfortable in using it.

The reason why data transmission must be fast and safe is because from 2005 to 2019, the internet users have increased by 10% [1]. This growth is closely related to traffic from internet networks in this world as the traffic is also experiencing growth per year. In addition, the potential for growth is also increasing.

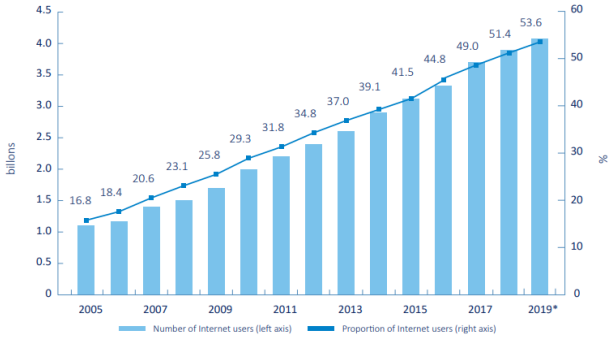

Fig. 1. The increases of internet users every year [1]

As more users look to operate faster while improving the data security, a solution is needed which can secure data but still have a good performance. Encryption, when used properly, can provide an additional security above basic protection data. In encryption algorithms, a key will be used to perform an encryption and also decryption. However, when the encryption algorithm is used to process the data in the form of numeric, the process requires a decryption before the data can be processed.

In this paper, a fully homomorphic encryption (FHE) scheme is used, where this scheme can process data in the form of numeric without having to do the decryption first. The data transactions on IoT device use a socket on the gateway. The time limit of the encryption and decryption process will be seen with data length of 2^n with various n.

This paper is structured of several chapters. Chapter I discusses the introduction of this experiment, chapter II contains various kinds of knowledge related to this experiment in the form of a literature, chapter III contains a design that proposed for this experiment, chapter IV is the result of the experiment, and chapter V is the conclusion of this experiment.

II. PREMILINARIES

This section will discuss some knowledge related to an IoT device and a method for securing existing data in the cloud as well as fully homomorphic encryption scheme.

A. *IoT (Internet of Things)*

IoT is a concept to expand the benefits of internet connectivity that is connected continuously [2]. Basically, IoT refers to an object that can be uniquely identified as a virtual representation in an internet-based structure. An IoT device will have a certain scheme of how a device can send and receive data. The schematic of an IoT device in sending and receiving data is shown in Figure 2.

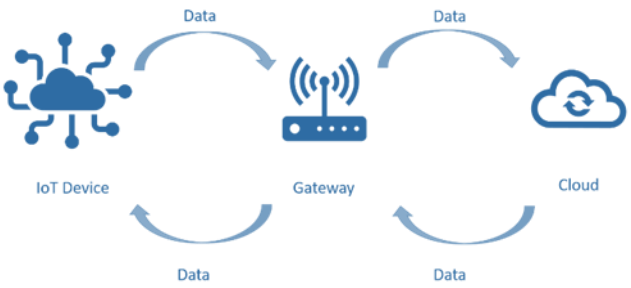

Fig. 2. IoT device scheme

From Figure 2, it can be seen that the data that sent by an IoT device will go through a gateway first before sending it to the cloud, as well as when an IoT device wants to receive data.

The role of the gateway here is as a data bridge between the cloud and IoT device. Before IoT device receives a data, gateway will process the data according to the request, such as filter data, delete data, and add additional data. A gateway is needed to carry out this process because an IoT device cannot perform complex processes but only carry out normal processes.

In order for secure the data, the data must be encrypted before sending it to the cloud so that when a person want to take or use the data to commit a crime then that person will only get encrypted data that won't be able to understand.

B. *Data Communication on Cloud*

Data communication in internet network requires a certain protocol so that data communication can take place. One of the most frequently used is the Hyper Text Transfer Protocol (HTTP) [3]. HTTP has been used for the basis of data communication on the world wide web (WWW) so that it can also used in a cloud.

HTTP has several delivery methods in communicating data, namely OPTIONS, GET, HEAD, POST, PUT, TRACE, DELETE, and CONECT. In every HTTP message, it consists a request from the client to the server and then the client will get a response from the server.

In order for guarantee its security, the data must be in an encrypted state before doing the sending process so that the data communication on the cloud remains safe. The schematic of data communication in the cloud is shown in fig. 3.

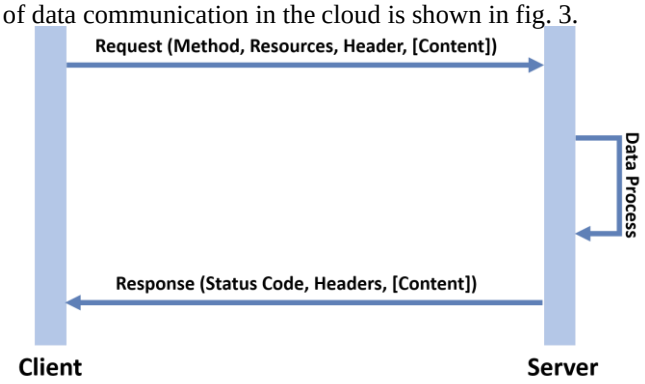

Fig. 3. Schematic illustration of the HTTP protocol [4]

HTTP has several delivery methods in communicating data, namely OPTIONS, GET, HEAD, POST, PUT, TRACE, DELETE, and CONECT. In every HTTP message, it consists a request from the client to the server and then the client will get a response from the server.

Figure 3 describes a client that sends a request to the server using the Application Interface Protocol (API). The request contains a delivery method, header, and message. The request that has been given by the client to the server will be processed by the server. The server will send a response containing the status, header, and message depending on the result process of the request from the client. The status from the response sent by the server used to tell the client how the request was given. This status is closely related to the success or failure of the request sent by the client to the server. This response can later be used by clients for various things. For example, in case of IoT devices, an IoT device can show this data to users.

C. *Message Queuing Telemetry Transport (MQTT)*

MQTT is a lightweight network protocol that commonly used to send messages between one device and another. The protocol used in MQTT is usually over TCP / IP, but all network protocols that can be used for two-way connections can also use MQTT [5].

In data communication, MQTT protocol requires two types of network entities, namely a broker and the number of clients. MQTT broker is a software that runs on a computer or cloud which acts like a post office. MQTT doesn't use the address of the recipient, but use a subject which is commonly referred as "Topic" so any client who wants to receive a messages must subscribe the topic. By this method, a broker can send a message to several clients at once.

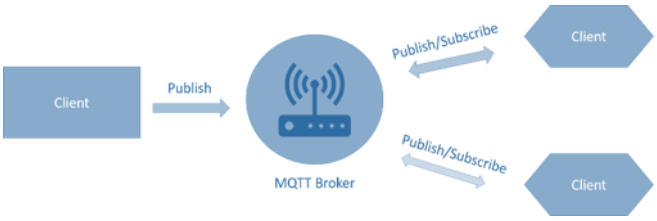

Fig. 4. MQTT Broker

MQTT is the best choice for network protocols in IoT devices because MQTT is lightweight network protocol that do not use too many methods as an IoT device usually do not use many methods as methods in HTTP.

D. *Gateway*

Gateway is part of networking which can be in form of hardware or software that can connect data between one device and another. This gateway is different from router because it uses various protocols to connect many networks [6].

In an IoT device, gateway is useful not only as a bridge in data transactions. The uses of the Gateway in the IoT device are as follows:

- Facilitating with devices that are not connected to the internet.
- Data caching, queuing, and filtering.
- Additional security, because it can control user access and manage network security.
- Configure the device.
- To diagnose a system

Apart from all the things mentioned above, a gateway can perform complex processes of the data. Therefore, without a gateway, an IoT device will be difficult to communicate with cloud and won't be able to perform complex data processing.

E. *Homomorphic Encryption*

Gentry first introduced an encryption algorithm, namely homomorphic encryption, in 2009 [7]. Homomorphic encryption is an asymmetric encryption technique where the algorithm consists of three main algorithms, that is key generation, encryption, and decryption.

Homomorphic encryption has its own strength when compared to other encryption algorithms, as it can perform operations on data without having a decryption process first. When the data that has been processed by homomorphic encryption is decrypted, the result is the same as the processing result of another encryption that requires a decryption proccess.

The scheme can be illustrated as follows, for example there is an encrypted message *E(m)*, where *m* is an original message, and a certain operation is performed, namely a function *f*, the encrypted message becomes *Enc(f(m))*. When the encrypted message is decrypted, the result of the message will be *f(m)* so *Enc(f(m)) = f (Enc(m))*.

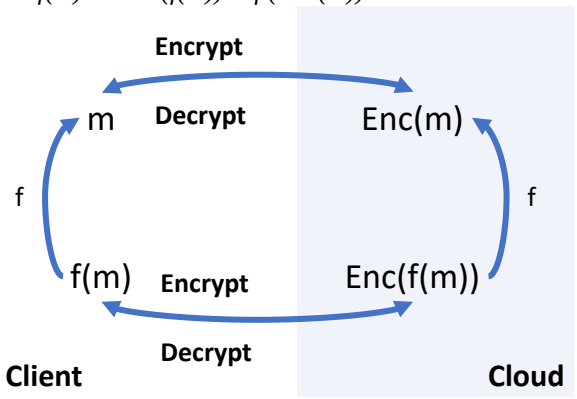

Fig. 5. Homomorphic Encryption Scheme [4]

Figure 5 shows that all encryption and decryption processes are carried out on the client, while the cloud only accepts encrypted data and send data in encrypted forms. It works like that so that if a person who is not responsible finds a security gap in cloud and gets the data, they won't be able to do anything with the data because that person will only get an encrypted data.

Fan and Vercautreren created homomorphic encryption algorithm based on a scheme that called Learning with Error (LWE) with a ring-LWE (RLWE) variant. The LWE operates using an integer, while RLWE operates using a polynomial ring [8]. The LWE scheme is shown in fig. 3.

Fig. 6. LWE Scheme

Fig. 6 shows that to make a public key for encryption process, an operation is performed first. The operation performed by multiplying the random number with a secret key and then adding it with a random error to get a public key.

Before LWE, key generation only used a random number which was basically difficult to crack. However, as time goes by and technology develops, random numbers can be solved by computers using the brute force network method. The LWE scheme uses an error value in key generation so that if one key and random numbers are to be known, it will be still difficult to find the other key because of an error stored in the key creation process. In addition, the error value is used in the encryption process so that the results of the encrypted data will be more difficult to solve even by a computer.

Homomorphic encryption has the potential to be applied in an IoT device because it can process data in an encrypted state so that the data will be processed faster without any decryption process and also add a additional layer of security. Moreover, homomorphic encryption is very suitable for this application because it can handle data traffic congestion on an internet network as an IoT device requires a real time connection where the processing must be done quickly.

F. *Securing Cloud Using Homomorphic Encryption*

Almost all cloud services that exist today requires high performance to transmit data. However, the cloud service is not only require high performance, but also require a high level of security. As internet users, choosing a right cloud service that have two requirements above is a must so that the data can be secure and processed quickly.

There are two threats in maintaining data security in the cloud, namely data that is in cloud and when the data is being transmitted from/to the cloud. The best way to secure both threats is to use an encryption technique and a data communication that meets the standards. Data in the cloud can be secured first by using encryption before being sent to the cloud as mention in [7] and [8]. Meanwhile, the data that is being transmitted can be secured by using a data communication that meets the standards as mention in [9] and [10].

The homomorphic encryption that used in this experiment have the following functions:

- *ParamGen*, for specifies the security parameters used for encryption
- *PubKeyGen*, for generate public key
- *SecKeyGen*, for generate secret key
- *Encrypt*, to encrypt the original data or plaintext
- *PolyAdd*, to perform addition operations on encrypted data or chipertext
- *PolyMul*, to perform multiplication operations on encrypted data or chipertext
- *Decrypt*, to decrypt the encrypted data or ciphertext

Homomorphic encryption is a suitable choice for securing a cloud while still paying attention to the performance of the cloud in transmitting the data.

G. *Network Socket*

A network socket is a software structure connected to a computer network that can serve as an endpoint for sending and receiving data over the network. Socket structure together with its property are using Application Programming Interface

(API) in its network architecture [11]. Due to the standardization of the TCP / IP protocol in the development process of the Internet, usually the provisions that exist on sockets depends on the existing protocols on the Internet.

Sockets are usually used when create applications or create programs that requires a real-time data. Socket always connect to the client where the client sends data and also send the data to the another clients. The socket scheme is shown in fig. 7.

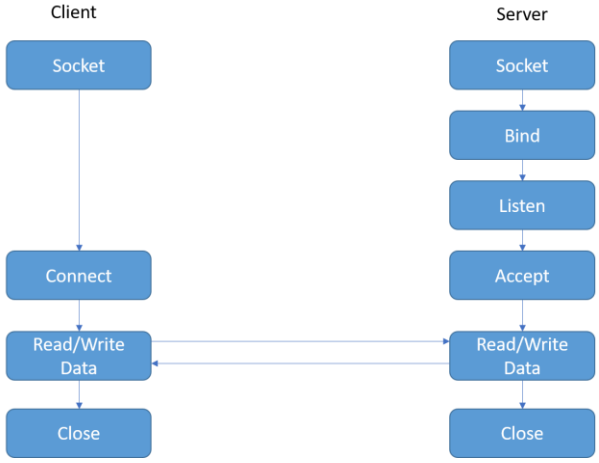

Fig. 7. Skema Socket

III. Fully Homomorphic Encryption for IoT Device

This section will proposed a design for IoT devices that will be integrated into a system that able to perform data retrieval, processing and, sending data between one hardware device to another. The design requires system gateway architecture, IoT network infrastructure, and IoT device design.

To check the readiness of the system that can work with many devices at once, a gateway with nice performance is needed to become an MQTT broker for the IoT system. The performance of IoT devices can be tested by hardware virtualization using internet bot which will do what the hardware would do to test the performance.

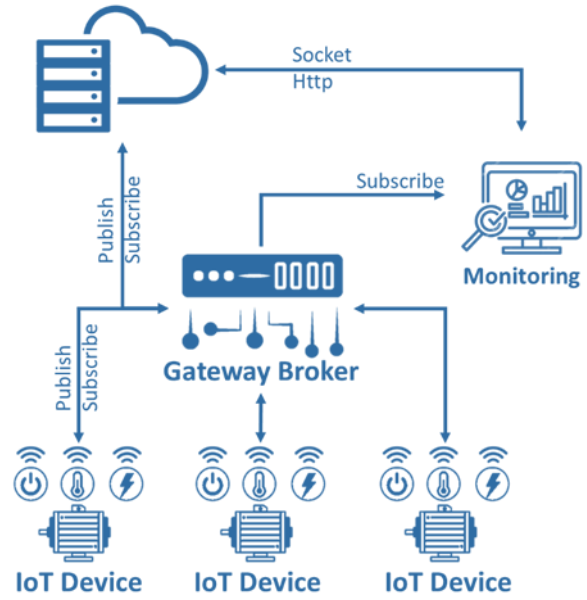

Fig. 8. System Infrastructure

Fig. 8 shows the IoT infrastructure system that we build. There are four entities that exist in this system infrastructure:

- IoT Device, implemented using the ESP8266 Wemos which will do a publish to the broker
- Gateway Broker, implemented using Raspberry Pi 3B+
- Monitoring, frontend interface that can view incoming data to a broker gateway
- Cloud, it can publish and subscribe to the broker with MQTT

A. *Gateway Infrastructure Design*

Broker is used to regulate the flow of data from the device to other components in the IoT system. For the implementation, the choice fell into MochaJs. The sensors and devices will send data to the topic of broker, and queued for encryption. If the data queue exceeds the capacity, it will be saved in the logging section and a warning will appear to the users.

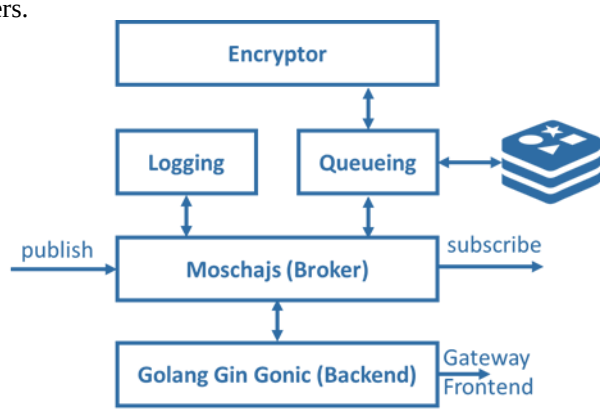

Fig. 9. Gateway Infrastructure

Queue data is stored in Redist. The reason for choosing redist is because redist allows to store data in volatile storage such as RAM, so that if there are things that are not desired, such as a system down, the data will be erased so that it is safe from irresponsible people.

After the data is encrypted, it will be stored to the broker. The topic has been subscriber by various components, such as databases and interfaces, so that the data will be forwarded to the database and user interface.

B. IoT Network Infratructure

The IoT network infrastructure is divided into two main parts, namely the getter data event and setter data event. The Getter data event used to get encrypted monitoring data at the gateway. Meanwhile, setter data event used to adjust the configuration of the monitoring tool, such as whether the tool is active or not.

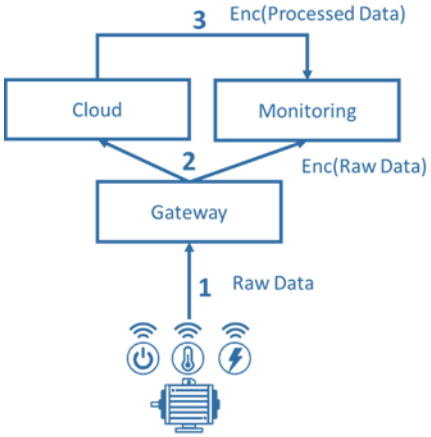

Fig. 10. Getter Data Event

The getter data event process start in a monitoring tool. Monitoring tool will send a raw, unencrypted data to the gateway. The gateway will encrypt the data and forward the data to the cloud for storage and further processing. Another gateway function is to send encrypted raw data to the monitoring interface. Data processing in the cloud including get average, minimum, and maximum data from sensors. The data that has been processed in the cloud will be displayed on the interface along with the raw data from the gateway.

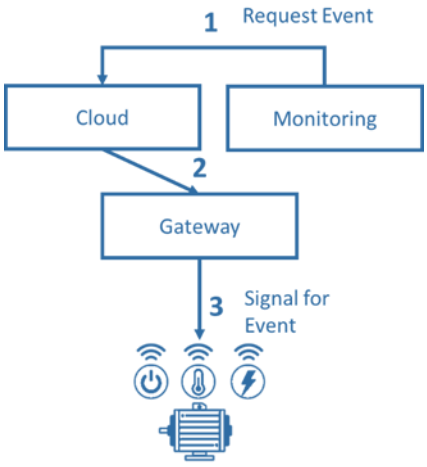

Fig. 11. Setter Data Event

The setter data event start in a user. User will changes the state of the device through the user interface. After the changes are processed by the cloud, the changes will be sent to the gateway and the gateway will forward it to the device. After the device received it, the state of the tool will change according to the user's wishes.

C. IoT Devices Design

The IoT device consists of a Wemos D1 Mini, Oled Display and MAX30102 Sensor which can read heart rate. IoT scheme is designed to ensure that each cable is connected properly and has the ability to communicate using the MQTT protocol.

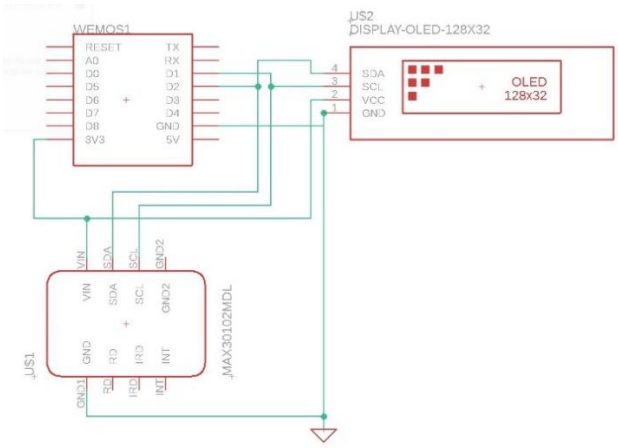

Fig. 12. IoT Device Schematics

The results of this scheme are tested using a breadboard and have proven to be running well. After receive a good results, a PCB is designed to implement the IoT device. PCB sized in 4.2 x 6.4 cm. After that, the soldering and testing stage are carried out to see if the system is still running well.

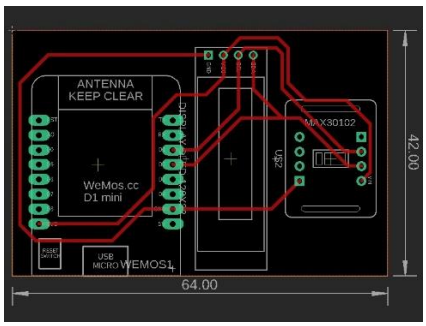

Fig. 13. PCB Design

IV. Experimental Results

We will see the implementation of the hardware that we designed where the hardware can detect the heart rate value and displayed it on the OLED Display. The hardware can also detect oxygen levels and patient temperature and the data will be sent directly to the gateway. We will look at the performance of the systems we design and build.

After that we also do benchmarking on the gateway that we have built. The stress test will be carried out on the designed gateway. The test will see how suitable the IoT device can be combined into IoT environment. It appears in the form of how may IoT devices that need to publish data every few second can be connected to the gateway.

The stress test is useful for seeing the performance of the gateway we choose and seeing the server capabilities with the specifications we have determined, whether it has the ability to become a gateway node for a large IoT infrastructure or not.

A. *Hardware Implementations*

We have tested the IoT device that we designed before. The result of the hardware implementation is that the device can display heart rate on OLED Display, the data is sent every 10 seconds to the gateway for encryption and the encrypted data is forwarded to the cloud. The Wemos D1 Mini microcontroller sends its data via wifi to the gateway using the MQTT protocol to get better efficiency. The performance results shows that MAX30102 sensor takes about 10 seconds to get the appropriate value.

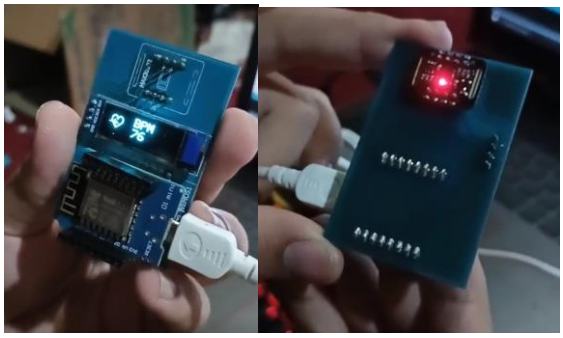

Fig. 14. Implementations

B. *Keygen Event*

We use virtualization to see how the system performs computation of key generation algorithm. We use 1000 virtual devices that will send a request to generate a key with polynomial degrees 4, 8, 16, 32, and 64. The request will be send every t ms/data.

1) Keygen Hitting

From the data shown below, the keygen request with polynomial degrees of 4 has the ability to handle data requests up to 12.5 ms for each data, and the threshold value will get worse as the polynomial degree value is increasing. When the polynomial degree is 64, it can only handle 1000 requests in 50 ms per hit.

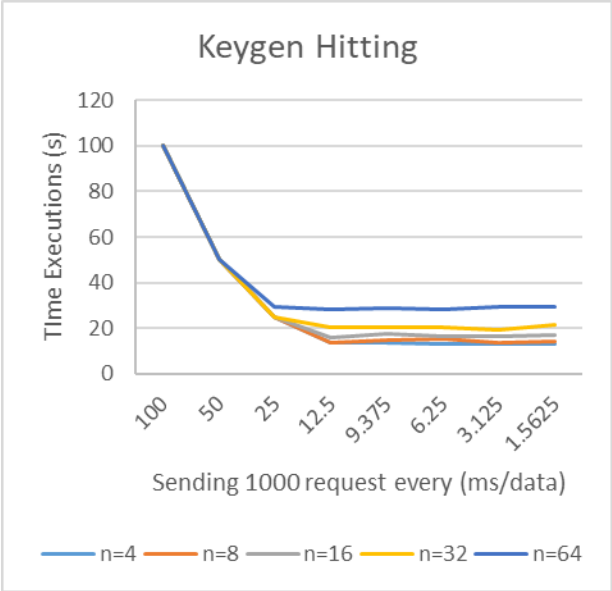

Fig. 15. Keygen Hitting Performance

2) Time Execution in Gateway

From the threshold that was previously obtained, we also have to see the time execution of gateway (Raspberry PI 3B+) as we do 1000 computation request. From the graph shown below, when we use small polynomial degrees, execution at the gateway is not too heavy and does not require a long queue, but when the polynomial degree increases, the process will stop and it will take a long time to do this computation process.

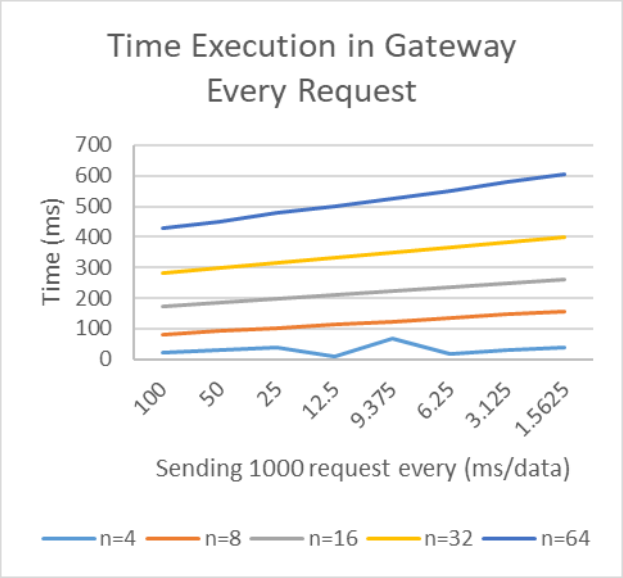

Fig. 16. Time Execution in Gateway

3) Scoring Performance

We will see the scoring that will be done by the gateway. The gateway should be able to process requests so that there is no queue system and all requests work properly. From Fig.17, it can be seen that when a score is below 1, no data is queued at the gateway before sending it to the cloud. When we use the smaller polynomial degree values, we also find that the ability to process a request from 1000 data every second is faster and more secure than when we using a larger polynomial.

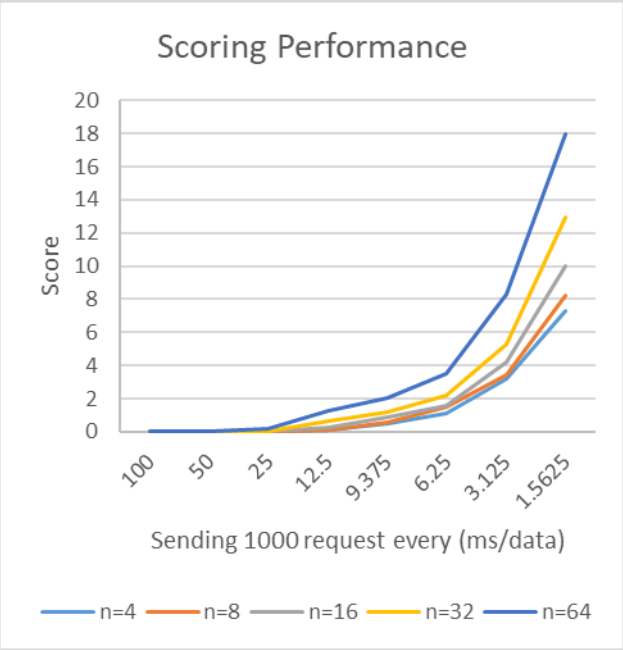

Fig. 17. Scoring Performance

C. Data Encryption

We also use virtualization to see how the system performs computation of encryption algorithm. We use 1000 virtual devices that will send a request to encrypt the data with polynomial degrees 4, 16, 64, and 256.

1) Encrypt Hitting

From the data shown below, the encryption request with polynomial degrees of 4 has the ability to handle 1000 requests in 6.25 ms, and the threshold value will get worse as the polynomial degree value is increasing as it can only handle 1000 requests in 500 ms per hit.

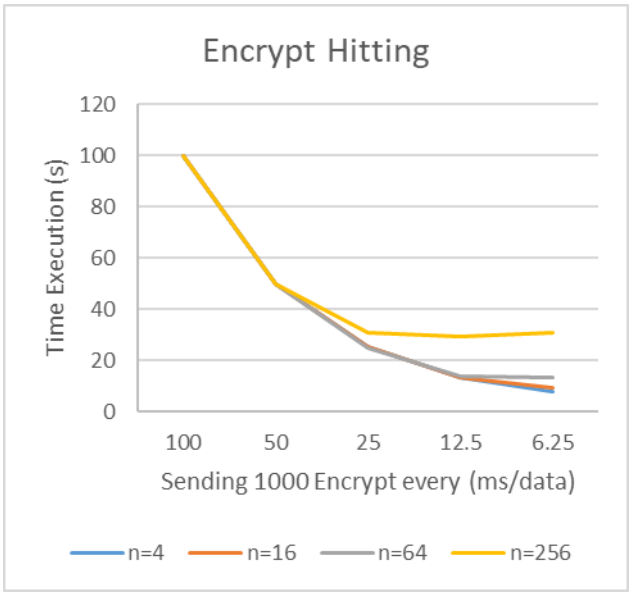

Fig. 18. Ecnryption Hitting Performance

2) Time Execution in Gateway

From the threshold that was previously obtained, we also have to see the time execution of gateway (Raspberry PI 3B+) as we do 1000 computation request. From the graph shown below, It can be seen that the higher of polynomial degree used greatly affects the ability of the gateway to perform computations. When we use the polynomial degree 64, there is also an increase in the execution time which makes the gateway throttling.

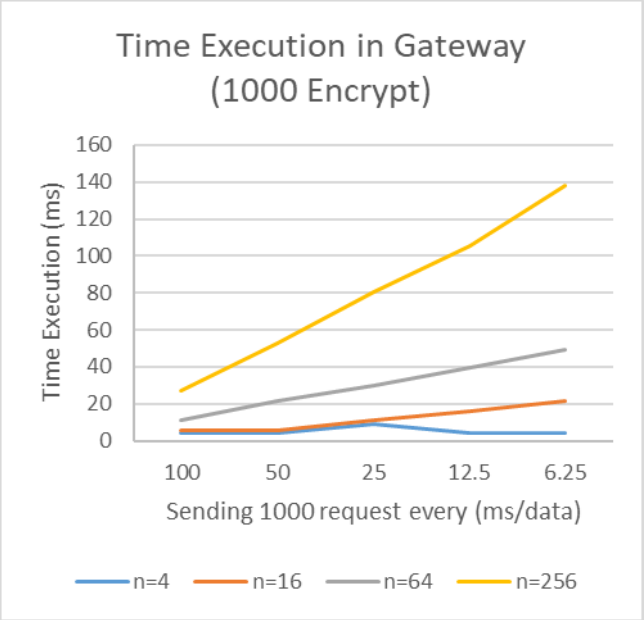

Fig. 19. Time Execution in Gateway

3) Scoring Performance

We will see the scoring that will be done by the gateway. The gateway should be able to process requests so that there is no queue system and all requests work properly. From the graph shown below, it can be seen that if we use polynomial degree 256, the gateway is only capable of reaching 50 ms per request, same as the previous experiment. But if we use the smaller polynomials degree, the gateway is able to compute up to 12.5 ms per 1000 request.

Fig. 20. Scoring Encryption

D. Data Decryption

Same as previous experiment, we use virtualization to see how the system performs computation of decryption algorithm. We use 1000 virtual devices that will send a request to encrypt the data using MQTT with polynomial degrees 4, 16, 64, and 256.

1) Decryption Hitting

From the data shown below, the decryption hitting request is slightly different from encryption hitting request. In contrast to encryption, system is still able to compute decryption with higher polynomial degrees. There is no significant difference between 64 polynomial degree and 16 polynomial degree, but the performance is less stable when using 256 polynomials degree.

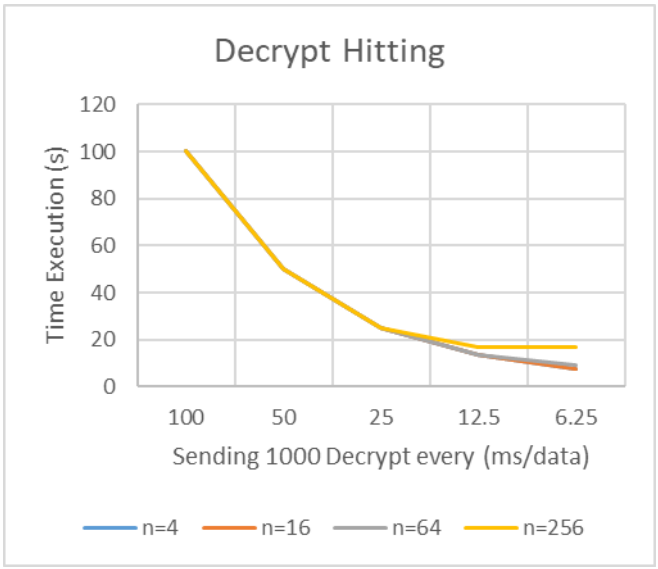

Fig. 21. Decryption Hitting Performance

2) Time Execution in Gateway

From the hitting performance above, we also see the processing time of the internal gateway for each request. From the graph in Figure 22, it can be seen that the performance of decryption process with polynomial degree 4, 16, and 64 are fast enough, where the gateway still able to do computations. But when the polynomial degree rises to 256, the gateway still able to do computation but with a long time.

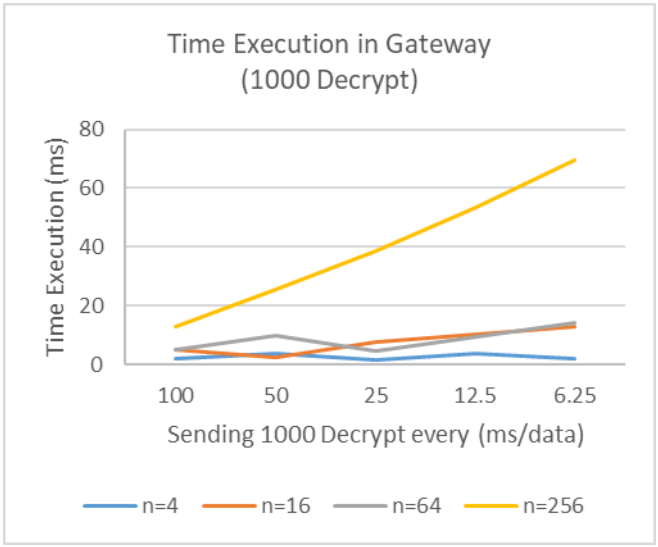

Fig. 22. Time Execution in Gateway

3) Scoring Performance

Then we also see the performance of the gateway's ability to perform this decryption to see which polynomial degree can be used to hit the requests. From the graph in Fig.23, it can be seen that for every 1000 requests, any polynomial degree can handle the request in 25ms. But when the hit increases to 12.5ms per 1000 requests, the polynomial degree that can still perform a decryption process is polynomial degree 4, 16 and 64. When it reaches to 6.25ms per 1000 requests, the polynomial degree that can still perform a decryption process is polynomial degree 4 and 16.

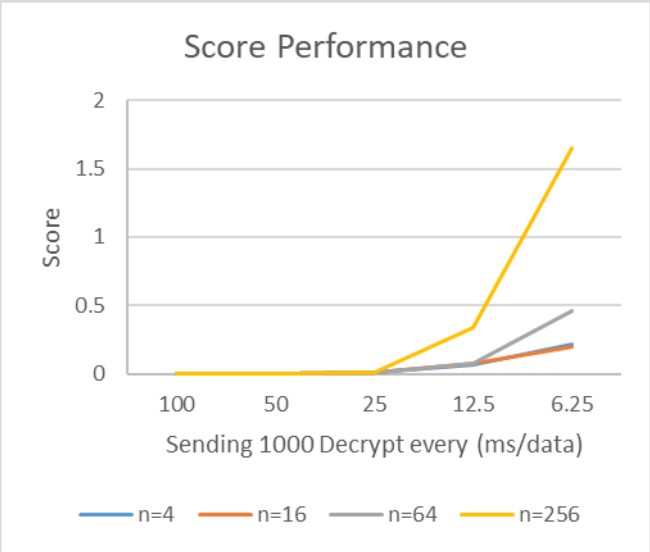

Fig. 23. Scoring Decryption

V. CONCLUSSIONS

Data integrity is important these days because data leakage from a device or system will cause a great losses regardless of what system we are talking about. One of the ways to improve the integrity of the data is by using encryption algorithms.

We implement this encryption algorithm using a fully homomorphic encryption with a *Brakerski / Fan*-Vercauteren scheme (FHE-BFV) to maintain data integrity on an IoT device. This IoT device sends its data using the MQTT protocol and will be encrypted at the gateway of the system that we build.

Kita juga mengimplementasikan alat IoT untuk melakukan pembacaan terhadap hearth rate yang kemudian datanya di transmisikan menggunakan wifi menggunakan sebuah protocol MQTT. Performa yang didapatkan juga kita lakukan virtualisasi untuk melihat kinerja dari gateway yang kita desain untuk melakukan semua algoritma kriptografi. Dari semua performance yang kita test jika kita menggunakan polynomial degree yang kecil yaitu 4 gateway masih dapat memperlihatkan performance yang tinggi tetapi pada saat kita memilih polynomial degree 64 hitting hanya mendapatkan performa pengiriman data tiap 12.5ms saja tiap 1000 datanya. Untuk nilai polynomial degree yang lebih tinggi performa hitting gateway pun harus semakin pelan.

We also implemented an IoT device that read heart rate that can be sent using WiFi with MQTT protocol. We also using virtualization to see the performance of the gateway that we designed to perform all encryption algorithms. Of all the performances we tested, the results shows that if we use a small polynomial degree, such as 4, the gateway still shows a high performance. But, if we choose a higher polynomial degrees, such as 64, the gateway shows a decreasing of performance as we get time execution equal to 12.5ms for every 1000 data. For higher polynomial degree, the gateway performance is shown to be decreasing.

REFERENCES

[1] International Telecomunication Union, "Measuring Digital Development Facts and Figures," *ITUPublications*, pp. 1-15, 2019.

[2] Technopedia, "Technopedia," Technopedia, 27 November 2020. [Online]. Available: https://www.techopedia.com/definition/28247/internet-of-things-iot. [Accessed 29 April 2021].

[3] R. T. Fielding, J. Gettys, J. C. Mogul, H. F. Nielsen, L. Masinter, P. J. Leach and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1," RFC2616, 2009.

[4] I. Syafalni, "Cloud Security Implementation using Homomorphic Encryption," *IEEE International Conference on Communication, Networks and Satellite (Comnetsat)*, pp. 341-345, 2020.

[5] OASIS, "MQTT Version 5.0," *OASIS Standard*, pp. 1-3, 07 March 2019.

[6] IDG Network World, Network World, IDG Network Worl Inc, 1997.

[7] C. Gentry, "A fully homomorphic encryption scheme," Stanford University, 2009.

[8] J. Fan and F. Vercauteren, "Somewhat Practical Fully Homomorphic Encryption," in *IACR Cryptology ePrint Archive*, 2012.

[9] M. Brener, W. Dai, S. Halevi, K. Han, A. Jalali, M. Kim, K. Laine, A. Malozemoff, P. Paillier, Y. Polyakov, K. Rohloff, E. Savas and B. Sunar, "A Standard API for RLWE-Based Encryption," in *Academic Consortium to Advance Secure Computation*, California, 2017.

[10] C. Adams, "The Simple Public-Key GSS-API Mechanism (SPKM)," Bell-Northern Research, 1996.

[11] I. A. D. V. S. Bagad, "Computer Networks (5th revised edition, 2010 ed.)," *Technical Publications Pune*, 2008.

[12] G. R. Sanchez, "Implementation of a chat applicationfor developers," *A Final Degree Project presented for the degree ofComputer Engineering*, 2017.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 26 April 2021

Gede Satya Adi Dharma