khazanah informatika

Jurnal Ilmu Komputer dan Informatika

Load Balancing Server and Homomorphic Encryption in Internet of Things

Muhammad Hafiz Amrullah, Favian Dewanta*, Sussi
Fakultas Teknik Elektro
Universitas Telkom
Bandung, Indonesia
*favian@telkomuniversity.ac.id

Abstract-User demand for Internet of Things (IoT) services has been increasing. The growing number of user demand can lead to an escalation of server workloads and threat of critical data theft. Consequently, a system is necessary to balance the server load where the data is protected with encryption. In this study, we designed a system to share server workloads using load balancing methods. The load balancing technique uses open-source web server software. The system is equipped with data security using a homomorphic encryption algorithm from AES on the sender's side. The system embeds in an IoT telemedicine apparatus. During testing, we analyze the error requests that arrive at each server for the HTTP GET and POST methods. We also evaluate the speed of data encryption and decryption. The results showed that server load balancing reduces the number of error requests for the GET method by 97%. Meanwhile, the number of error requests for the POST method decreases by 66.75%. Observations reveal that the average homomorphic encryption speed, computation time, and decryption time are 15.66 ms, 764.18 µs, and 362.49 µs, respectively.

Keywords: load balancing, servers, requests, homomorphic encryption, AES algorithm

Article info: *submitted February 3, 2021, revised April 6, 2021, accepted May 26, 2021*

1. Introduction

The current industrial development encourages the development and application of the Internet of Things (IoT). The main objective of IoT technology is to enable connected devices to communicate with each other, exchange data, store data, and perform computing complying with the user requirement. However, there are several obstacles in IoT implementation, one of which is the server load and data security on the server and database. When the number of IoT users requesting service increases and the server cannot handle the requests, the server will receive too many requests that may cause the service to fail to respond as expected [1].

To overcome these problems and improve server performance, server load balancing and homomorphic encryption systems can be implemented as a solution in terms of uniform distribution of service loads and data protection with fast computing. The system implementation expectedly improves the reliability and security of the IoT system. Server load balancing works by considering the capacity of each server and distributing workloads to several servers, which may reduce failures on the servers [2]. Homomorphic encryption is a cryptographic algorithm that allows (arithmetic) computing on the ciphertext directly. It avoids the encryption process on the plain text, which forces the decryption process, which prolongs the steps. Homomorphic encryption result is the same with encryption to plaintext [3]. It allows safe data storage in the database in the form of ciphertext, and it achieves a faster processing time compared to the use of regular cryptographic algorithms.

In this study, the server design adopts an open-source webserver to implement a load balancing system and apply homomorphic encryption on a web server that runs on an IoT-based telemedicine system. Testing parameters include the error request received by the server and the length of time the homomorphic encryption process runs on the server.

2. Basic theory

a. Load Balancing

Load balancing is a technology to divide the traffic load evenly on two or more connection lines in a balanced way so that traffic does not experience congestion and can

run optimally. The target of load balancing is to maximize throughput, reduce response time, and anticipate overload on one connection line [4]. If the service users on the server continue to grow and exceed the capacity available in the network traffic path, the load balancer will distribute the load from users evenly to all available servers. In addition, load balancers can also evenly distribute loads of CPU, hard disk, RAM, and other computing resources to get the best performance from the server.

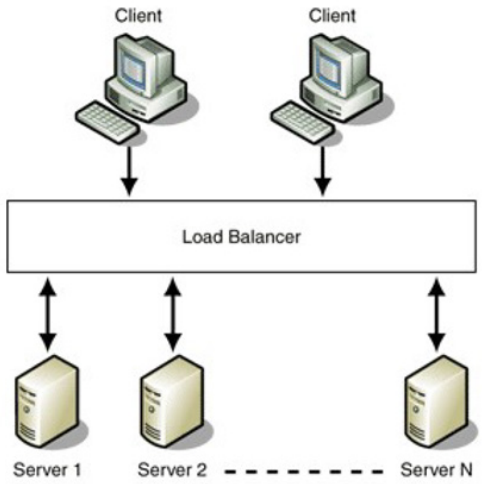


Figure 1. Typical load balancer configuration [4].

In this study, we implement a load balancer using open-source software called NGINX that uses the Round Robin algorithm which has its server or is separate from other servers. The Round Robin algorithm is the simplest algorithm and the most widely used algorithm for load balancer devices. The Round Robin algorithm works by distributing the load sequentially from one server to another. The basic concept of the Round Robin algorithm uses time-sharing, which simply processes the queue (traffic or computation) in turn [5].

b. Advanced Encryption Standard (AES)

Advanced Encryption Standard (AES) is an encryption algorithm with a symmetric key exchange and applies a block cipher system that has a block length of 128 bits [6]. In cryptography, there are terms plain text (plaintext) and ciphertext (ciphertext). Plaintext is the initial information or data before the information or data is encrypted, and ciphertext is information or data from encrypted plain text. AES encryption uses a different number of round keys for each type of block size, namely a block length of 128 bits is 10 rounds, a block length of 192 bits is 12 rounds, and a block length of 256 bits is 14 rounds [7].

The type of AES block cipher used in this study is AES Cipher Block Chaining (CBC) 256 bits as illustrated in Figure 2. The advantage is that if the information or data has the same plaintext, the encrypted information or data cannot be repeated with the same encryption. This is due to the use of an Initialization Vector (IV) which has a different and random value for each data encryption.

CBC is the operating mode of the block cipher whose IV length is the same as each plaintext block. The initial process of encryption is to XOR plaintext with IV, and then generate encrypted data (ciphertext) for plaintext blocks. Then the resulting ciphertext will be used as IV again in the next block. In this way, each ciphertext generated will depend on each ciphertext in the previous block, so that each encrypted data becomes unique [8].

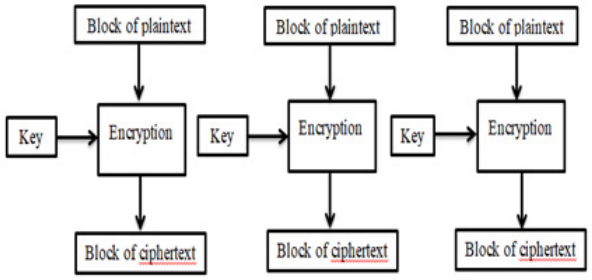


Figure 2. CBC mode of operation [9].

c. Homomorphic encryption

Homomorphic encryption is a cryptographic algorithm that makes it possible to compute encrypted data without decrypting the data directly as the concept is described briefly in Figure 3. . Homomorphic encryption uses an encryption function with addition (addition) and or multiplication (multiplication) operations on encrypted data [10].

There are two types of homomorphic encryption, namely Partially Homomorphic Encryption (PHE) and Fully Homomorphic Encryption (FHE). PHE is homomorphic encryption, which allows certain types of operations to be used on the ciphertext. While FHE is a homomorphic encryption, which allows two operations, namely addition and multiplication of ciphertext.

In this research, a homomorphic FHE encryption process is applied from AES encrypted data which is used to calculate the average value of the data sent from the sender. This homomorphic encryption process uses the help of a python library called Pyfhel.

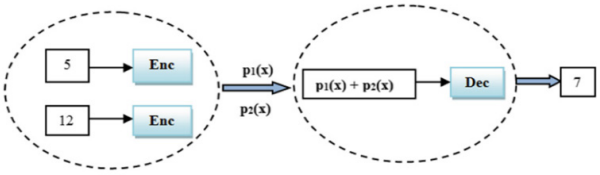


Figure 3. Homomorphic encryption concept [10].

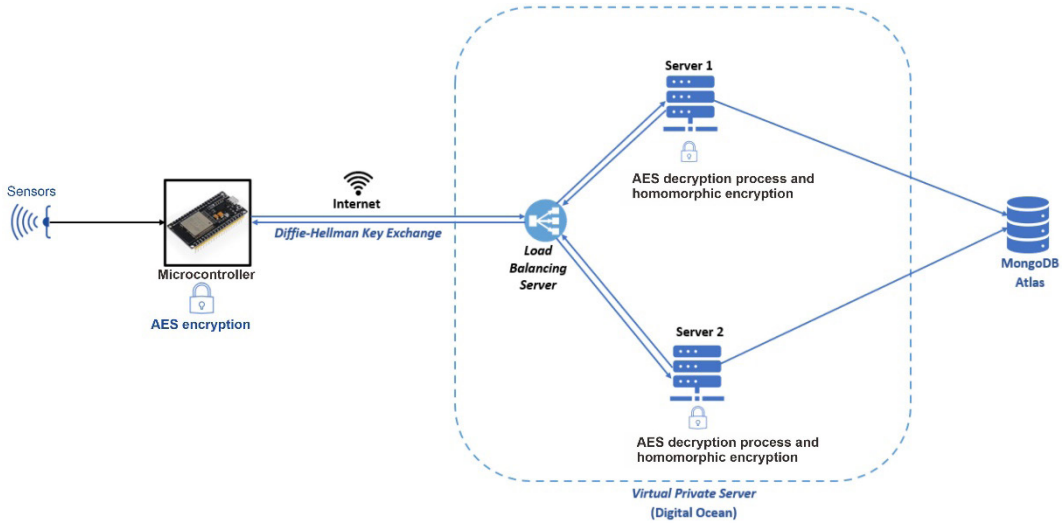


Figure 4. Design of the system created.

3. Method

a. System planning

At this stage, the author designs a system from sensors to the cloud server and its database as shown in Figure 4. The author also analyzes the communication method between the right elements in the system to be used in solving problems based on information from datasheets, tutorials, and other sources. others available.

Figure 4 shows the overall design of the telemedicine system. The input data comes from the sensor. The data sent by the microcontroller is the ciphertext data from AES, the shared key to calculate the secret key, and the IV generated from the microcontroller. Before encryption, the microcontroller and server use the Diffie-Hellman algorithm to exchange shared keys to calculate the symmetric secret-key value to perform the AES encryption process and generate the ciphertext. The ciphertext data is passed to the VPS via the created web server API. The ciphertext data first passes through the load balancer of the webserver which distributes the data traffic to one of the two servers used. After that decrypt the AES ciphertext data received by the server. After the ciphertext is decrypted, the plaintext results are then encrypted and computed using a homomorphic encryption algorithm to calculate the average value per 100 data received by server 1 and server 2, and the results of the average are decrypted again using a homomorphic algorithm. Based on the average value of the plaintext data, AES encryption is performed again using the received key and IV. After the AES encryption process is complete, the ciphertext data will be saved to the MongoDB Atlas database.

Figure 5 is a system flow diagram starting with the microcontroller performing the Diffie-Hellman key exchange process. First, the process of making shared keys X and IV, then the microcontroller will get the shared key Y from the server and then the shared keys X and IV are sent to the server. The microcontroller will generate the secret key K from the shared key Y computation, and then perform hashing to get the 256-bit key. After that, the AES algorithm is used to encrypt the data and generate the ciphertext. The AES ciphertext is sent to the cloud server. While on the cloud server, the data first arrives at the load balancer which is created using the Round Robin algorithm with the initial process of scheduling all user requests that enter the load balancer until all user requests are scheduled, the load balancer will distribute user requests to one of the servers. If the selected web server is overloaded, then the data requested by the user is reset to be sent to another web server. After the requested data is received by one of the servers, the request data in the form of ciphertext is decrypted and computed using homomorphic encryption. Once the computation is complete, the data will be passed to the MongoDB Atlas database for storage.

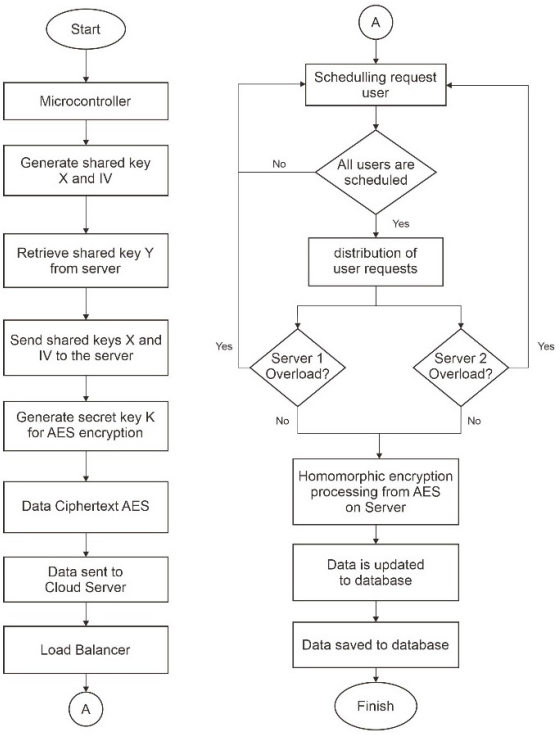


Figure 5. Process diagram flow of the system.

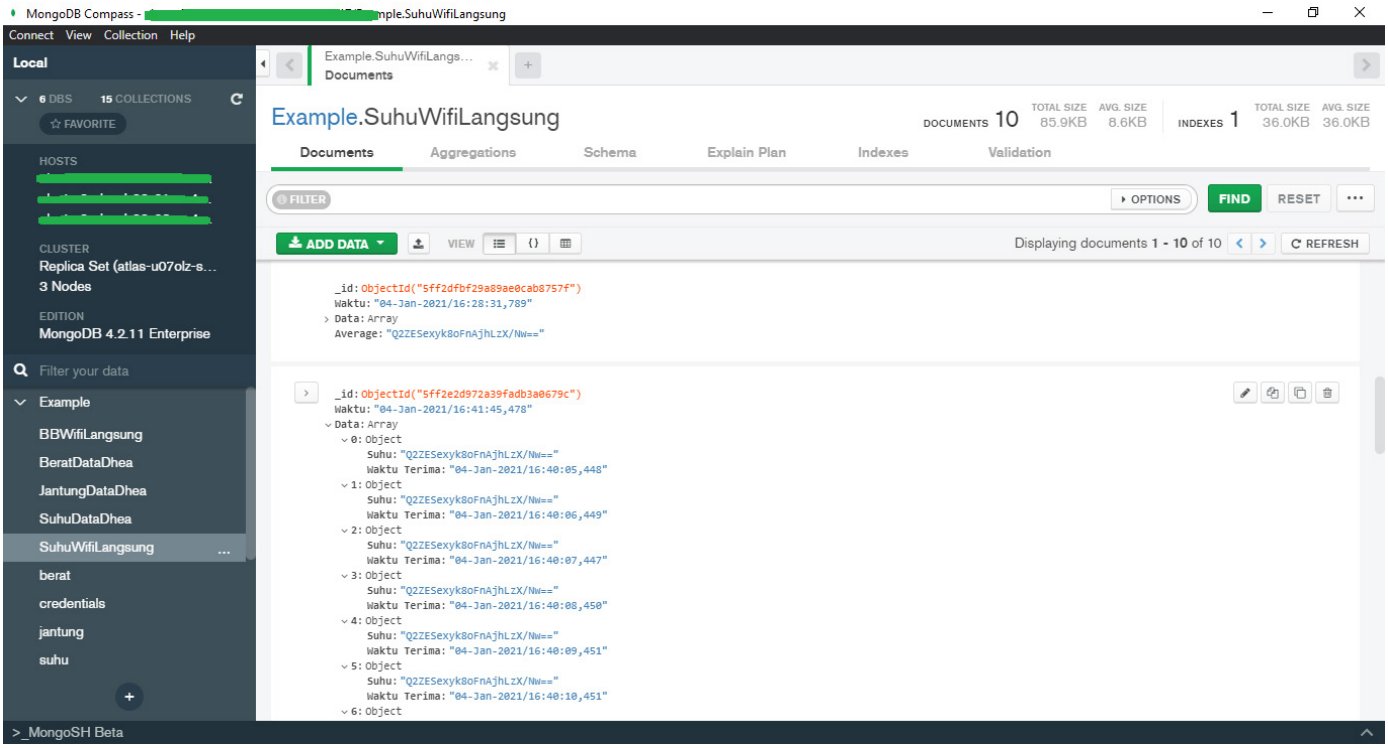


Figure 6. MongoDB database view.

Figure 6 is a display of ciphertext data storage in MongoDB. The encrypted temperature data will be stored in an array variable named "Data" as many as 100 encrypted data and from the 100 data the average value is calculated using homomorphic encryption which is stored in the "Average" variable in an encrypted fixed form.

b. Implementation

At this stage, the author makes the system after the previous design has been completed.

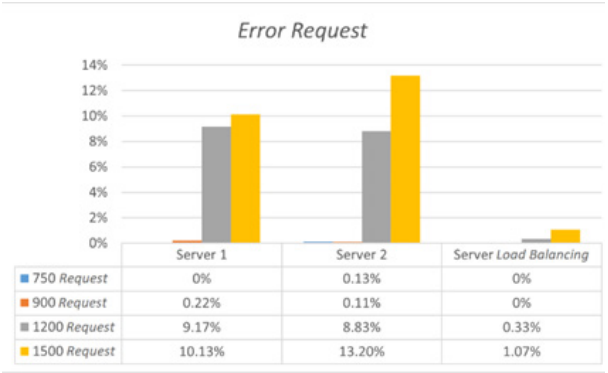
c. Testing and Analysis Phase

At this stage, the system is tested to observe the required data. The test is carried out by measuring QoS and obtaining data during the homomorphic encryption and decryption process.

The test is carried out using Apache Jmeter software, where the testing scheme for error request parameters is carried out by sending a different number of requests, namely 750, 900, 1200 and 1500 requests where all requests are sent within 10 seconds. To test AES processing speed and homomorphic processing, each experiment was performed 5 times by sending 100 requests within 100 seconds.

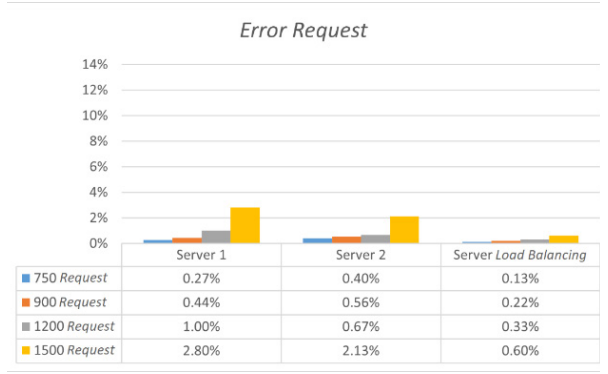
4. Result

Figure 7 and Figure 8 are the results of testing error requests received by the server for the GET and POST methods based on the number of different requests (ie 750, 900, 1200, and 1500 requests).



	Server 1	Server 2	Server Load Balancing
750 Request	0%	0.13%	0%
900 Request	0.22%	0.11%	0%
1200 Request	9.17%	8.83%	0.33%
1500 Request	10.13%	13.20%	1.07%

Figure 7. GET method request error chart.



	Server 1	Server 2	Server Load Balancing
750 Request	0.27%	0.40%	0.13%
900 Request	0.44%	0.56%	0.22%
1200 Request	1.00%	0.67%	0.33%
1500 Request	2.80%	2.13%	0.60%

Figure 8. POST method request error chart.

The result of the error request test on the GET method shows that the error request results between server

1 and server 2 are almost the same. However, when using server load balancing techniques, the number of error requests received is reduced by an average of about 97%. At the same time, the results of the POST method error request testing show that the error request results between server 1 and server 2 also show almost the same results. However, on the load balancing server used, the number of error requests received was reduced by an average of about 66.75%. This proves that the load balancing server implemented for the IoT telemedicine system is operating properly.

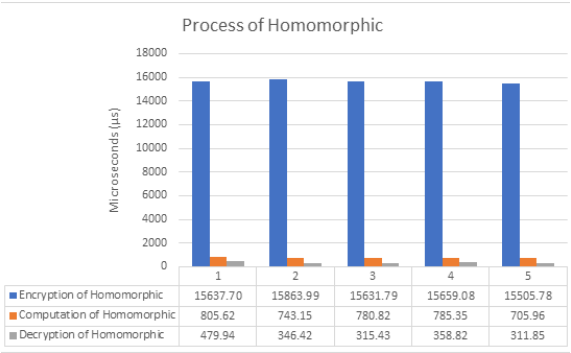


Figure 9. Homomorphic process time chart.

Figure 9 is the result of testing the time of the homomorphic encryption process which includes 3 processes, namely encryption, computation, and homomorphic decryption. When testing the speed of the homomorphic encryption and decryption process, the encryption and decryption process was carried out in 5 trials by sending 100 data to the server each time. The results of the encryption processing time show that the average homomorphic encryption process time is 15.66 milliseconds, the homomorphic computation time average is 764.18 microseconds (0.76 milliseconds), and the homomorphic decryption time average is 362.49 microseconds (0.36 milliseconds). The results of the computational speed test on the homomorphic cryptography algorithm show that the encryption process takes the most time because the encryption process in the algorithm uses a lot of computational overhead in processing plaintext data to ciphertext. This reason is sometimes a practical consideration not to apply homomorphic encryption to databases that require real-time data computation.

5. Conclusion

Server load balancing can reduce error requests received by the server. The test results show that error requests for the GET method are reduced by 97%, while error requests for the POST method are reduced by 66.75%. Homomorphic encryption can be applied to the server to calculate the average value of the data received by the server on the IoT system. It proves that homomorphic encryption runs well without errors. The average speed of homomorphic encryption, computation, and decryption is 15.66 ms, 764.18 s (0.76 ms), and 362.49 s (0.36 ms), respectively, which suggests that encryption works as expected.

Reference

[1] S. D. Riskiono and D. Pasha, "Analisis Metode Load Balancing Dalam Meningkatkan Kinerja Website E-Learning," J. Teknoinfo, vol. 14, no. 1, p. 22, 2020, doi: 10.33365/jti.v14i1.466.

[2] S. D. Riskiono, "Implementasi Metode Load Balancing Dalam Mendukung Sistem Kluster Server," pp. 455–460, 2018, doi: 10.31227/osf.io/9vuzx.

[3] Q. Wang, D. Zhou, and Y. Li, "Secure Outsourced Calculations with Homomorphic Encryption," Adv. Comput. An Int. J., vol. 9, no. 6, pp. 01–14, 2018, doi: 10.5121/acij.2018.9601.

[4] A. Rahmatulloh and F. MSN, "Implementasi Load Balancing Web Server menggunakan Haproxy dan Sinkronisasi File pada Sistem Informasi Akademik Universitas Siliwangi," J. Nas. Teknol. dan Sist. Inf., vol. 3, no. 2, pp. 241–248, 2017, doi: 10.25077/teknosi.v3i2.2017.241-248.

[5] F. Apriliansyah, I. Fitri, and A. Iskandar, "Implementasi Load Balancing Pada Web Server Menggunakan Nginx," J. Teknol. dan Manaj. Inform., vol. 6, no. 1, 2020, doi: 10.26905/jtmi.v6i1.3792.

[6] X. W. Wu, E. H. Yang, and J. Wang, "Lightweight security protocols for the Internet of Things," IEEE Int. Symp. Pers. Indoor Mob. Radio Commun. PIMRC, vol. 2017-Octob, pp. 1–7, 2018, doi: 10.1109/PIMRC.2017.8292779.

[7] B. K. S. Rajaram and N. Krishna Prakash, "Secure mqtt using aes for smart homes in iot network," Int. J. Innov. Technol. Explor. Eng., vol. 8, no. 5s, pp. 483–485, 2019.

[8] A. M. Al Naamany, A. Al Shidhani, and H. Bourdoucen, "IEEE 802 . 11 Wireless LAN Security Overview," Ijcsns, vol. 6, no. 5, pp. 138–156, 2006.

[9] M. E. Hameed, M. M. Ibrahim, N. A. Manap, and M. L. Attiah, "Comparative study of several operation modes of AES algorithm for encryption ECG biomedical signal," Int. J. Electr. Comput. Eng., vol. 9, no. 6, pp. 4850–4859, 2019, doi: 10.11591/ijece.v9i6.pp4850-4859.

[10] Y. Alkady, F. Farouk, and R. Rizk, "Fully Homomorphic Encryption with AES in Cloud Computing Security," Adv. Intell. Syst. Comput., vol. 845, pp. 370–382, 2019, doi: 10.1007/978-3-319-99010-1_34.