Security-Based Online Store Development using Fully Homomorphic Encryption Algorithm with BFV Scheme

Syarifuddin Fakhri A
School of Electrical Engineering and Informatics
Bandung Institute of Technology
Bandung, Indonesia
syarifuddinfa@gmail.com

Rinaldi Munir
School of Electrical Engineering and Informatics
Bandung Institute of Technology
Bandung, Indonesia
rinaldi@informatika.org

Infall Syafalni
School of Electrical Engineering and Informatics
Bandung Institute of Technology
Bandung, Indonesia
infall@staff.stei.itb.ac.id

***Abstract*—The rapid and advanced development of technology has impacted various sectors of life, one of which is the commerce sector, namely the existence of online stores. With this shift in habits, a new type of crime has emerged, known as cybercrime. One consequence of cybercrime, which is still prevalent in Indonesia, is data breaches. To address this issue, it is necessary to create a security-based online store application. The development of a security-based online store was carried out by implementing the fully homomorphic encryption (FHE) algorithm with the BFV (Brakerski-Fan-Vercauteren) scheme. This algorithm was chosen for its ability to perform computations on encrypted data, eliminating the need for prior decryption. The BFV scheme was chosen for its faster performance compared to other schemes such as BGV or CKKS. Based on development and testing results, this algorithm can perform computations such as addition, subtraction, negation, and multiplication on previously encrypted data, producing the corresponding values when decrypted. In addition, it is necessary to pay attention to the parameters of the FHE algorithm builder with the BFV scheme because the greater the ring dimension value and the multiplicative depth value, the longer the computation time required, while the security level parameter does not significantly affect application performance.**

Keywords—online shop, web application, web security, fully homomorphic encryption, BFV scheme

I. Introduction

With current technological advancements, commerce, particularly the buying and selling of goods, can now be conducted online. This has transformed the buying and selling paradigm, which was previously limited to brick-and-mortar stores, into online stores. The emergence of online stores has enabled the buying and selling process to transcend time and place. Buyers and sellers can conduct transactions wherever and whenever they choose. These online transactions are known as electronic commerce (e-commerce). Between 2019 and 2020, 138.1 million people in Indonesia shopped through e-commerce, with an average purchase of \$219 per person, resulting in total transactions valued at \$30.31 billion, a 49% increase from the previous year [1].

Despite the many advantages of online stores, they also have disadvantages, namely being vulnerable to cybercrime, often referred to as cyberattacks. A cyberattack is a deliberate attempt to steal, expose, alter, disable, or destroy data, applications, or other assets through unauthorized access to a network, computer system, or digital device [2]. One consequence of cybercrime, which is still prevalent in Indonesia, is data breaches. According to BSSN, in 2024, there were 241 suspected data breach incidents [3]. One such breach incident involving an online store was the data breach involving 91 million Tokopedia user accounts that occurred in May 2020.

One way to secure online stores from cyber threats, especially data breaches, is by using cryptography to secure data and credentials. This data, including user accounts, purchase data, payment data, and so on, is highly private and vulnerable to misuse if it falls into the wrong hands. Cryptography is the art and science of maintaining the security of messages [4].

Traditional encryption methods provide an efficient and secure way to store private data in encrypted form. However, the problem with traditional methods is that encrypted data cannot be computed, so it must first be decrypted before computation can be performed. Homomorphic encryption (HE) is a type of encryption method that allows computations to be performed on ciphertext data without first decrypting it using a secret key [5]. This encryption algorithm is believed to be difficult to crack, even using quantum computers. Therefore, in the development of this online store application, the HE algorithm will be used to secure credential data. In its implementation, the type of algorithm used is fully homomorphic encryption (FHE) with the BFV (Brakerski-Fan-Vercauteren) scheme. FHE is an encryption method that allows computations, both addition and multiplication, to be performed directly on encrypted data. The BFV scheme is one of the FHE implementation schemes chosen because it is designed for exact computations on integers and has faster processing time than other schemes such as BFV or CKKS. It is hoped that by using this algorithm, data can be stored securely and prevent data leaks in Indonesia.

II. FHE ALGORITHM WITH BFV SCHEME

Fully homomorphic encryption (FHE) is an encryption method that allows both addition and multiplication computations to be performed directly on encrypted data. FHE was first formally introduced by Craig Gentry in 2009, as a solution to the limitations of the partially homomorphic encryption (PHE) algorithm, which only allows one type of operation (addition or multiplication) to be performed on ciphertext [5]. The FHE algorithm is a type of public key encryption that uses a public key to encrypt a message and a private key to decrypt it.

The BFV (Brakerski-Fan-Vercauteren) scheme is one of the most common and efficient FHE implementations, especially for modular integer computation. BFV utilizes the Ring Learning with Error (RLWE) assumption as a security foundation and is designed to support arithmetic operations on the plaintext ring $\Phi_n(x)$, where t is the plaintext modulus. This scheme consists of the following.

- Key generation to generate the public key, private key, and evaluation key (multiplication / relinearization / rotation).
- Encryption to convert the plaintext to ciphertext.
- Evaluation to support arithmetic operations on the ciphertext.
- Decryption to return the evaluation result to the plaintext domain.

Mathematical operations for the FHE algorithm with the BFV scheme, which include key generation, encoding/decoding, encryption, decryption, addition operations, and multiplication operations. Below, each mathematical operation will be explained, the references of which are taken from Fan & Vercauteren [6] and Brakerski [7].

A. *Basic parameters and notation*

- n: Degree of the polynomial (usually a power of 2). This defines the space $R = \mathbb{Z}[x] / (x^n + 1)$.
- q: Ciphertext modulus (integer). This produces the space $R_q = \mathbb{Z}_q[x] / (x^n + 1)$.
- t: Plaintext modulus (integer t<q). This produces the space $R_t = \mathbb{Z}_t[x] / (x^n + 1)$.
- χ: Error distribution (noise) (usually a discrete Gaussian distribution).
- Δ: Scale factor. Calculated as $\Delta = \lfloor q / t \rfloor$.
- Private Key: A polynomial $s \in R_q$ with small coefficients taken from χ.
- Public Key: A pair of polynomials $(p_0, p_1) \in R_q \times R_q$.
- Plaintext: A polynomial $m \in R_t$.
- Ciphertext: A pair of polynomials $(c_0, c_1) \in R_q \times R_q$.

B. *Key Generation*

- Secret key (sk)

  Sample a polynomial s from the error distribution χ. The secret key is $sk = s$.

- Public key (pk)
  1. Random polynomial sample $a \in R_q$.
  2. Small error sample $e \in \chi$.
  3. Calculate $p_0 = -(a \cdot s + e)\ (mod\ q)$.
  4. Resulting public key $pk = (p_0, p_1)$.

$$pk = (p_0, p_1) = (p_0, a) = (-(a \cdot s + e), a) \quad (1)$$

C. *Encoding/Decoding*

- $Encode(v)$: Generates the polynomials $m(x) \in R_t$.
- $Decode(m(x))$: Extracts the constant coefficients from the polynomial $m(x)$.

D. *Encrypt*

Encryption on plaintext m using public key pk.

1. Sample three small error polynomials $u, e_1, e_2 \in \chi$.
2. Scale the message $m_{scaled} = \Delta \cdot m \in R_q$.
3. The resulting ciphertext is $c_t = (c_0, c_1)$.

$$c_0 = p_0 \cdot u + e_1 + m_{scaled} (mod\ q) \quad (2)$$

$$c_1 = p_1 \cdot u + e_2\ (mod\ q) \quad (3)$$

E. *Decrypt*

Decryption on cihpertext ct using secret key sk.

1. Calculate noisy message.

$$m_{noisy} = c_0 + c_1 \cdot s\ (mod\ q) \quad (4)$$

2. Get plaintext from scaling cancellation and reduce noise.

$$m = \left\lfloor \frac{t}{q} \cdot m_{noisy} \right\rceil (mod\ t) \quad (5)$$

F. *Addition Operation*

Addition operation is performed on the ciphertext $ct_1 = (c_{1,0}, c_{1,1})$ which encrypts m_1 and $ct_2 = (c_{2,0}, c_{2,1})$ which encrypts m_2 resulting new ciphertext $ct_{add} = (c_{add,0}, c_{add,1})$.

$$c_{add,0} = c_{1,0} + c_{2,0}\ (mod\ q) \quad (6)$$

$$c_{add,1} = c_{1,1} + c_{2,1}\ (mod\ q) \quad (7)$$

G. *Multiplication Operation*

Multiplication operation is performed on the ciphertext $ct_1 = (c_{1,0}, c_{1,1})$ which encrypts m_1 and $ct_2 = (c_{2,0}, c_{2,1})$ which encrypts m_2.

1. Calculate three new polynomials c_0', c_1', c_2'.

$$c_0' = c_{1,0} \cdot c_{2,0} \pmod q \quad (8)$$

$$c_1' = c_{1,0} \cdot c_{2,1} + c_{1,1} \cdot c_{2,0} \pmod q \quad (9)$$

$$c_2' = c_{1,1} \cdot c_{2,1} \pmod q \quad (10)$$

2. Scale the previous polynomials.

$$c_0'' = \left\lfloor \frac{t}{q} c_0' \right\rceil \pmod q \quad (11)$$

$$c_1'' = \left\lfloor \frac{t}{q} c_1' \right\rceil \pmod q \quad (12)$$

$$c_2'' = \left\lfloor \frac{t}{q} c_2' \right\rceil \pmod q \quad (13)$$

In part 1 will produce ciphertext $ct'_{mult} = (c_0', c_1', c_2')$ which has 3 components in it. If decrypted with the modification in part 2 to $ct''_{mult} = c_0'' + c_1'' + c_2''$ will produce $Decrypt(ct''_{mult}) = m_1 \cdot m_2 \pmod t$.

III. Online Store Application Implementation using FHE with BFV Scheme

A. User Functional Requirements

The following *TABLE I* defines the functional requirements and security implementation of the FHE algorithm.

TABLE I. User Functional Requirements

Functional requirement	FHE Operation
Users can register and log in to the application	Initiate crypto context parameters and create public and private keys
Users can top up and use balances	Plaintext encryption and ciphertext addition operations
Users can view balances and previous balance history	Ciphertext decryption operations
Users can view products and add them to their cart	-
Users can make transactions to order products	Plaintext encryption, ciphertext addition, and ciphertext multiplication operations
Users can view previous transaction history	Ciphertext decryption operations

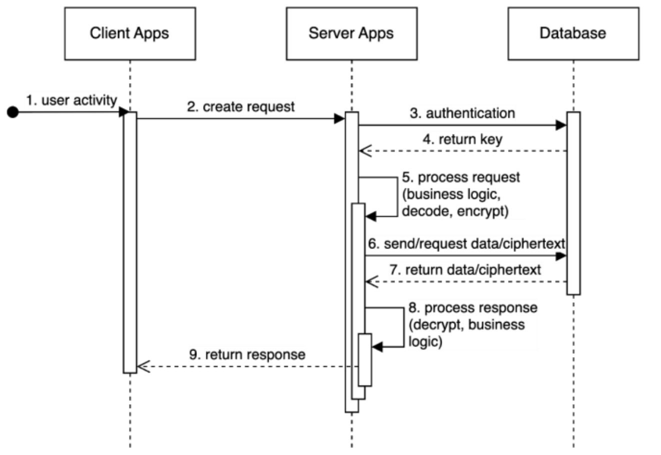

Client Apps
Server Apps
Database
1. user activity
2. create request
3. authentication
4. return key
5. process request (business logic, decode, encrypt)
6. send/request data/ciphertext
7. return data/ciphertext
8. process response (decrypt, business logic)
9. return response

Fig. 1. Data flow diagram in architecture

B. Solution Design and Implementation

The online store implementation will utilize a three-tier architecture consisting of a database, back-end, and front-end. The database implementation will utilize a relational database with the PostgreSQL database system. The back-end implementation will utilize Python, utilizing the OpenFHE library to handle the FHE algorithm and the FastAPI framework to create the API service. The front-end implementation will also utilize Python, utilizing the Django framework for the GUI and the Requests library to make requests to the API service. The data processing diagram for this online store can be seen in Fig. 1.

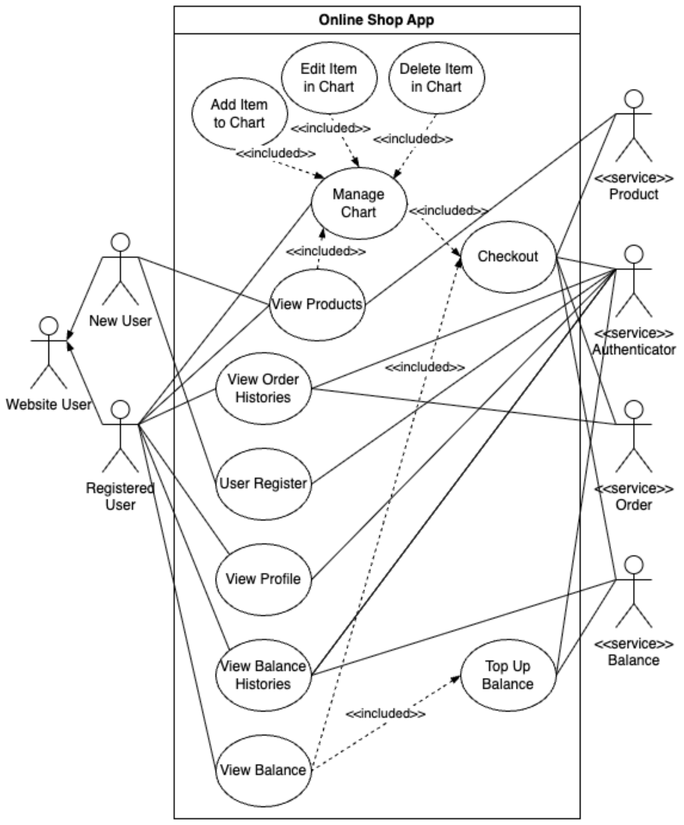
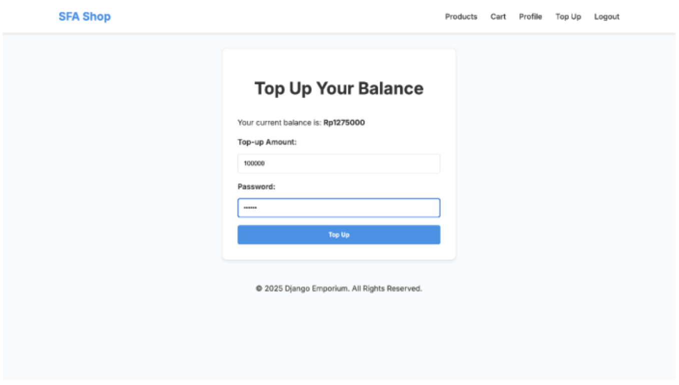


Fig. 2. Application Use Case Diagram

Based on the user functionality requirements in *TABLE I*, the proposed solution system has four main components: authentication service, balance service, product service, and ordering service. These four components need to be implemented at each architectural tier. The authentication service regulates how users verify their identity. However, if a user has not registered, the authentication service functions to register the user. The balance service is used to view and manage the user's balance. Users can use their balance for transactions; if the balance is insufficient, they can top up the balance first. The product service is used to view products that can be added to the cart and later purchased by the user. Meanwhile, the ordering service is used to carry out product ordering transactions. In addition, this service is also used to view previous transaction history. Of the four services mentioned, only the product service does not use FHE operations. For details on the relationship between services and functionality, see Fig. 2.

Below you can see product ordering service implemented in Fig. 3 and Fig. 4. With an initial balance of Rp1.275.000, then an additional balance of Rp100.000 is added. After clicking “Top Up”, the balance will increase to Rp1.375.000.



Top Up Your Balance

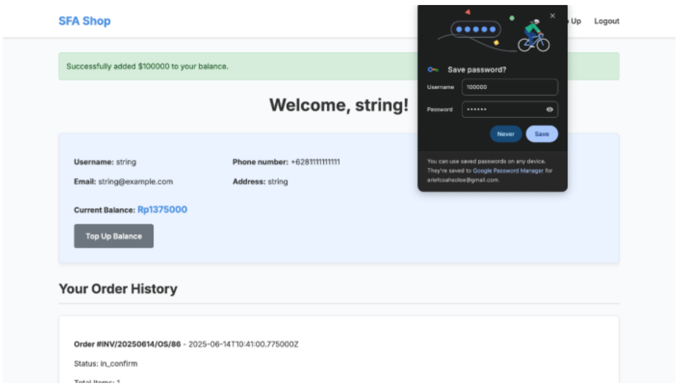
Your current balance is: **Rp1275000**

Top-up Amount: 100000

Password: ••••••

© 2025 Django Emporium. All Rights Reserved.

Fig. 3. App view when top up balance



Successfully added $100000 to your balance.

Save password?

Username: 100000

Password: ••••••

You can use saved passwords on any device. They're saved to Google Password Manager for ariefcoahsoloe@gmail.com.

Welcome, string!

Username: string

Phone number: +6281111111111

Email: string@example.com

Address: string

Current Balance: Rp1375000

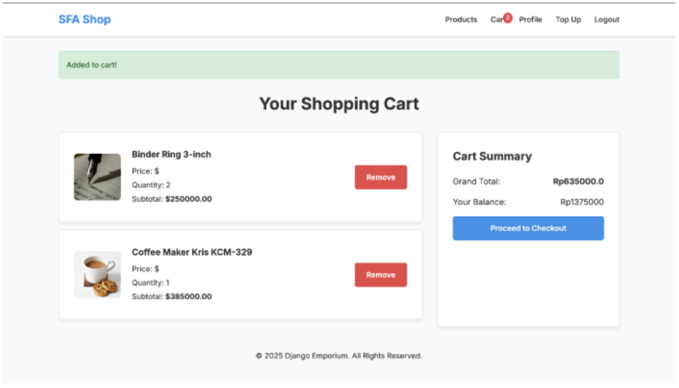
Your Order History

Order #INV/20250614/OS/86 - 2025-06-14T10:41:00.775000Z

Status: in_confirm

Fig. 4. App view after top up balance

Below you can see product ordering service implemented in Fig. 5 and Fig. 6. With an initial balance of Rp1.375.000, an order is placed for two items for a total purchase of Rp635.000. After clicking "Proceed to Checkout", the order is processed and the previous balance is deducted, leaving a remaining balance of Rp740.000.



Added to cart!

Your Shopping Cart

Binder Ring 3-inch

Price: $

Quantity: 2

Subtotal: **$250000.00**

Coffee Maker Kris KCM-329

Price: $

Quantity: 1

Subtotal: **$385000.00**

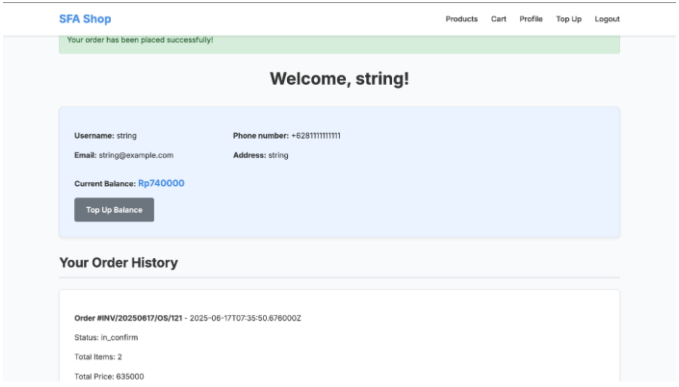
Cart Summary

Grand Total: **Rp635000.0**

Your Balance: Rp1375000

© 2025 Django Emporium. All Rights Reserved.

Fig. 5. App view when ordering products



Your order has been placed successfully!

Welcome, string!

Username: string

Phone number: +6281111111111

Email: string@example.com

Address: string

Current Balance: Rp740000

Your Order History

Order #INV/20250617/OS/121 - 2025-06-17T07:35:50.676000Z

Status: in_confirm

Total Items: 2

Total Price: 635000

Fig. 6. App view after ordering products

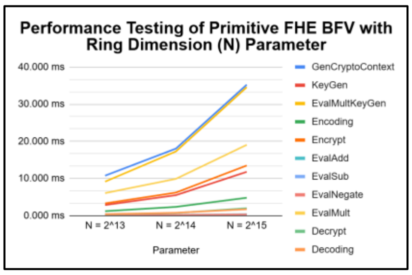


Fig. 7. Performance Testing Result of Primitive FHE BFV with Ring Dimension (N) Parameter

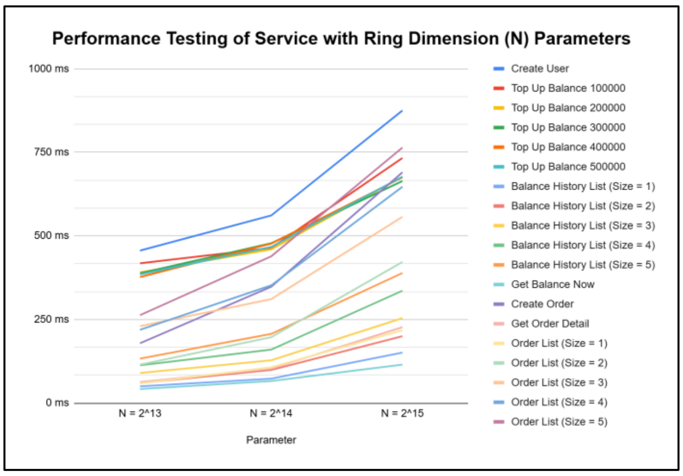


Fig. 8. Performance Testing Result of Service with Ring Dimension (N) Parameter

IV. Application Computational Performance Testing

Application computational performance testing is conducted to evaluate the computational time performance of primitive functions of the FHE algorithm using the BFV scheme, the performance of features utilizing the FHE algorithm using the BFV scheme, and performance when receiving a certain amount of load or traffic. The metric used for this test evaluation is response time, which is the time required to process and respond to a request. In this test, performance testing was conducted on variations in the parameters used to build the FHE algorithm using the BFV scheme. Testing was conducted using Python, utilizing the Pytest and Locust libraries. This test consisted of testing FHE primitive functions, testing services, and testing multiple user loads. All tests were performed with the plain modulus (t) parameter set to 119603201. This parameter defines the value space or range for your original data (plaintext) before encryption. The following parameters were used for testing:

1. ring dimension (N): 2^{13}, 2^{14}, and 2^{15}
2. security level: HEStd_128_classic, HEStd_192_classic, and HEStd_256_classic
3. multiplicative depth: 3, 4, and 5

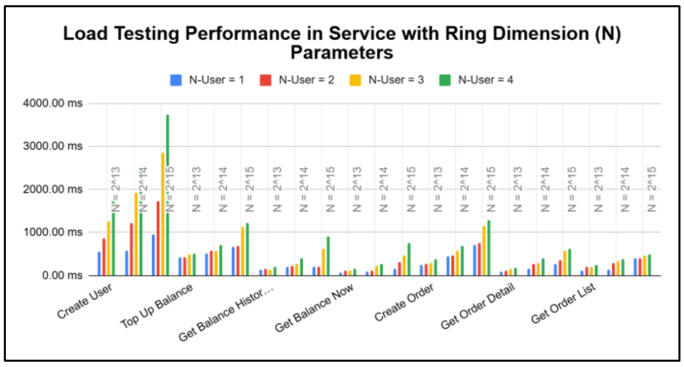


Fig. 9. Load Testing Performance Result in Service with Ring Dimension (N) Parameters

A. *Testing on Ring Dimension Parameters*

Ring dimension (N) parameter is a parameter that determines the degree of the polynomial that determines the size of the algebraic structure (polynomial ring) used in the encryption scheme. Ring dimension testing is carried out using the security level HEStd_128_classic and multiplicative depth 3. In the results of the ring dimension testing that has been tried in Fig. 7, it can be seen that the larger the number N, the required operating time increases in all primitive functions of the FHE algorithm with the BFV scheme. A significant increase occurs when generating the crypto context value and generating the eval mult key value but is not too significant in the evalSub and evalNegate functions. The increase in this primitive function has an impact on increasing the execution time of the service using the FHE algorithm with the BFV scheme as can be seen in Fig. 8. It can also be seen in Fig. 9 that with the increasing number of users, there is an increase in service execution time.

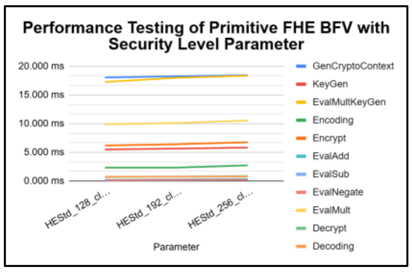


Fig. 10. Performance Testing Result of Primitive FHE BFV with Level Security Parameter

B. *Testing on Security Level Parameters*

The security level parameter is a parameter that defines the resilience of the encryption scheme against cryptanalysis attacks. This parameter sets the level of bit security equivalent to established cryptographic standards. Security level testing is carried out using Ring Dimension (N) 2^{14} and multiplicative depth 3. In the results of the security level testing that has been tried in Fig. 10, it can be seen that the higher the security level, the execution time on the entire primitive function of the FHE algorithm with the BFV scheme tends to be unchanged but there is still a very slight increase of less than one millisecond. This can also be seen in Fig. 11, the results of the service performance test tend to only increase slightly, namely less than fifty milliseconds. Although it tends to increase, it can be seen in Fig. 12 that there is still an increase in execution time if the number of service users increases.

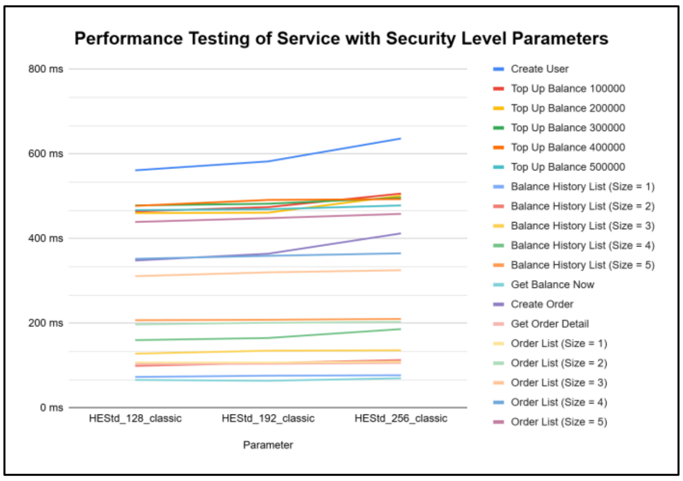


Fig. 11. Performance Testing Result of Service with Security Level Parameter

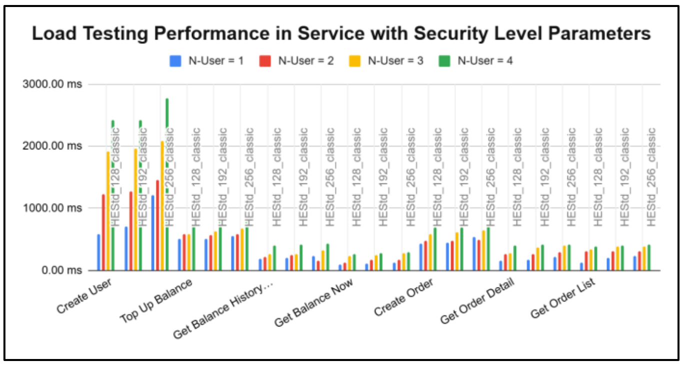


Fig. 12. Load Testing Performance Result in Service with Security Level Parameters

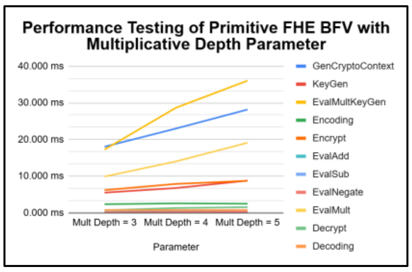


Fig. 13. Performance Testing Result of Primitive FHE BFV with Multiplicative Depth Parameter

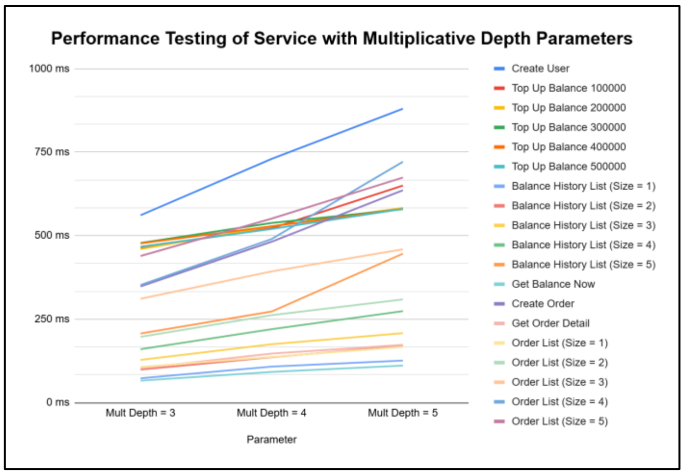


Fig. 14. Performance Testing Result of Service with Multiplicative Depth Parameter

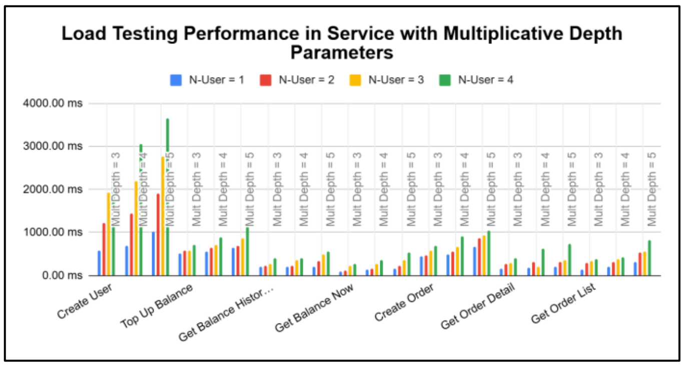


Fig. 15. Load Testing Performance Result in Service with Multiplicative Depth Parameters

C. *Testing on Multiplicative Depth*

The multiplicative depth parameter defines the maximum number of consecutive multiplication operations that can be performed on encrypted data before the noise inherent in the ciphertext grows too large, making the data unable to be decrypted correctly. Each multiplication operation between two ciphertexts will increase this noise level. Multiplicative depth testing was performed using a ring dimension (N) of 2^{14} and a security level of HEStd_128_classic. The results of the multiplicative depth testing that have been tried, can be seen in Fig. 13 that the larger the multiplicative depth number, the more the required operation time increases for all primitive functions of the FHE algorithm with the BFV scheme. Significant increases occur when generating crypto context values and generating eval mult key values but are not very significant for the evalSub and evalNegate functions. Increases in these primitive functions have an impact on increasing execution time on services using the FHE algorithm with the BFV scheme as can be seen in Fig. 14. In the load testing, it can be seen in Fig. 15 that there is an increase in time when the number of service users increases.

V. Conclusion

Based on the design results, test results, and analysis, the following conclusions can be drawn.

1. Online store applications can be built and developed with a security-based approach using the FHE algorithm with the BFV scheme. This utilizes the Python programming language, the OpenFHE, FastAPI, and Django libraries, and the PostgreSQL relational database, allowing users to conduct transactions and securely store stored data.
2. With the FHE algorithm with the BFV scheme, computations (addition, subtraction, negation, and multiplication) can be performed on encrypted data and produce the appropriate values without requiring prior decryption.
3. The larger the ring dimension (N) and multiplicative depth parameters, the greater the computation time of the FHE algorithm with the BFV scheme. The security level parameter does not significantly increase computation time. Furthermore, the greater the number of application users, the greater the computation time.
4. To build an online store application using the FHE algorithm with a secure BFV scheme with a reasonable computation time, you can use the ring dimension 2^14, security level HEStd_256_classic, and multiplicative depth 3.
5. The private key cannot be stored privately by the user, so it still needs to be stored in the database and user credentials are required to access it.

References

[1] Kemp, S. (2021). DIGITAL 2021: INDONESIA. https://datareportal.com/reports/digital-2021-indonesia

[2] IBM. (2021). What is a cyberattack? https://www.ibm.com/think/topics/cyber-attack

[3] BSSN. (2025). Lanskap Keamanan Siber Indonesia 2024. https://www.bssn.go.id/wp-content/uploads/2025/02/LANSKAP-KEAMANAN-SIBER-2024-1.pdf

[4] Schneier, B. (1997). Applied Cryptography. CRC Press, Inc.

[5] Gentry, C. (2009). A Fully Homomorphic Encryption Scheme.

[6] Fan, J., & Vercauteren, F. (2012). Somewhat Practical Fully Homomorphic Encryption. IACR Cryptology ePrint Archive, 2012, 144.

[7] Brakerski, Z. (2012). Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. CRYPTO 2012