Columbus State University

CSU ePress

Theses and Dissertations

Student Publications

2024

# Blockchain for Computational Integrity and Privacy

Rahul Raj

Follow this and additional works at: https://csuepress.columbusstate.edu/theses_dissertations

Part of the Computer Sciences Commons

Recommended Citation

Raj, Rahul, "Blockchain for Computational Integrity and Privacy" (2024). *Theses and Dissertations*. 530.
https://csuepress.columbusstate.edu/theses_dissertations/530

This Thesis is brought to you for free and open access by the Student Publications at CSU ePress. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of CSU ePress.

COLUMBUS STATE UNIVERSITY

# BLOCKCHAIN FOR COMPUTATIONAL INTEGRITY AND PRIVACY

A THESIS SUBMITTED TO

THE TURNER COLLEGE OF BUSINESS AND

TECHNOLOGY

IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED COMPUTER SCIENCE

TSYS SCHOOL OF COMPUTER SCIENCE

BY

RAHUL RAJ

COLUMBUS, GEORGIA

2024

Copyright © 2024 Rahul Raj

All Rights Reserved.

# BLOCKCHAIN FOR COMPUTATIONAL INTEGRITY AND PRIVACY

By

Rahul Raj

Approved By

Committee Chair:

Dr. Yeşem Kurt Peker

Committee Members:

Dr. Lixin Wang
Dr. Linqiang Ge

Columbus State University
May 2024

## ABSTRACT

This study proposes a blockchain based system that utilizes fully homomorphic encryption to provide security of data in use as well as computational integrity. This is achieved by leveraging the attributes of blockchain which provides availability and data integrity combined with homomorphic encryption that provides confidentiality. The proposed system is designed to perform statistical operations, including mean, median and variance, on encrypted data, thus providing confidentiality of data while in use. The computations are performed on the smart contract, residing on the blockchain which provides computational integrity. The results indicate that it is possible to perform fully homomorphic computations on the blockchain. However, there is a need for more resources and enhancement in technology for such a system to be implemented as a real-world application.

INDEX WORDS: Computational Integrity, Privacy, Blockchain, Homomorphic Encryption

## ACKNOWLEDGEMENTS

I would like to extend my heartfelt gratitude to several individuals whose contributions were instrumental in the completion of this thesis. Foremost among them is my advisor, Dr. Yeşem Kurt Peker. I am deeply appreciative of her support, invaluable guidance, and insightful feedback throughout this journey. I am also indebted to Dr. Lixin Wang and Dr. Linqiang Ge for their expertise and constructive criticism, which have significantly enriched this work.

To my family, I owe a debt of gratitude for their love, encouragement and understanding. Their sacrifices and endless support have been the source of my strength and motivation.  Lastly, I would like to express my deepest appreciation to my friends for their encouragement, understanding and belief in my abilities.

# Table of Contents

## LIST OF TABLES

LIST OF FIGURES

# CHAPTER 1. INTRODUCTION

The term blockchain has become popular ever since cryptocurrencies were introduced. Commonly associated with Bitcoin, it is a technology used to perform and record transactions. Originally conceived as the underlying technology for digital currencies, blockchain has since been implemented across diverse sectors, including supply chain management, healthcare, and more. This was made possible because of its core attributes, namely decentralization, which distributes control among participants; immutability, which ensures that once recorded, data cannot be altered; and transparency, which provides visibility into transactions. These attributes have sparked a wave of innovation and experimentation, driving interest and investment in blockchain technology worldwide. However, despite having these attributes, blockchain has some limitations as well.

One significant challenge is the lack of data privacy. Traditional blockchain systems, while offering immutability and fault tolerance, often fall short in ensuring confidentiality of sensitive information. Thus, limiting its acceptance in sectors that deal with such data. This gap needs to be addressed for blockchain to be implemented in sensitive domains such as personal information of users, including but not limited to health records or financial information.

This study aims to bridge this gap and address the lack of privacy in current blockchain systems by combining homomorphic encryption with the immutable nature of blockchain. This is achieved by encrypting the data before it is sent to the smart contract residing on the blockchain and introducing computational integrity by performing the computations on encrypted data on the smart contract. Thus, in addition to providing security of data at rest, this system provides security of data in transit as well as in use. In this study, Zama, a blockchain platform which provides libraries for performing homomorphic encryption on blockchain, is used to design and implement a system that performs descriptive statistics on encrypted data. Specifically, calculation of the mean, median and variance of encrypted data is implemented in the system.

This paper is divided into six major chapters. Chapter 2 introduces the background of blockchain technology, homomorphic encryption, and Zama. Chapter 3 presents a literature review highlighting similar work done either on blockchain or on homomorphic encryption. Chapter 4 explains the methodology of the proposed system in detail, followed by the result and discussion, including the security analysis of the proposed system in Chapter 5. Finally, Chapter 6 provides a conclusion of the study.

# CHAPTER 2. BACKGROUND

## 2.1 Blockchain Technology

Blockchain is a digital ledger technology that records transactions and data in a tamper resistant and decentralized manner [1]. As the name suggests, it is basically a chain of blocks where each block contains some data or information regarding certain transactions along with a header that contains metadata including the hash value which acts as a unique identifier for that block. The first block is known as the genesis block. Each block contains the hash value of the previous block (except the genesis block) which is how the blocks are linked together. This chain of blocks is known as a ledger which is shared among every node in the network, making it decentralized (also known as peer-to-peer network) and eliminating single point of failure. A block is added to the chain via an agreed upon consensus mechanism - a protocol through which peers of the blockchain network reach agreement about the present state of the data in the network. Consensus algorithms establish reliability and trust in the blockchain network. The second major advantage of blockchain is that it is tamper resistant. Since each block contains the hash value of the previous block, changing the contents of one block changes its hash value which alters the hash value of every block that follows. This makes blockchain tamper resistant because changing the hash value of every block is a very difficult task. Another feature of blockchain, specifically public blockchain, is transparency which means that anyone can view the data in each block. While transparency may be desirable in some applications, the lack of confidentiality is a security concern for many sectors, especially the ones that handle sensitive information.

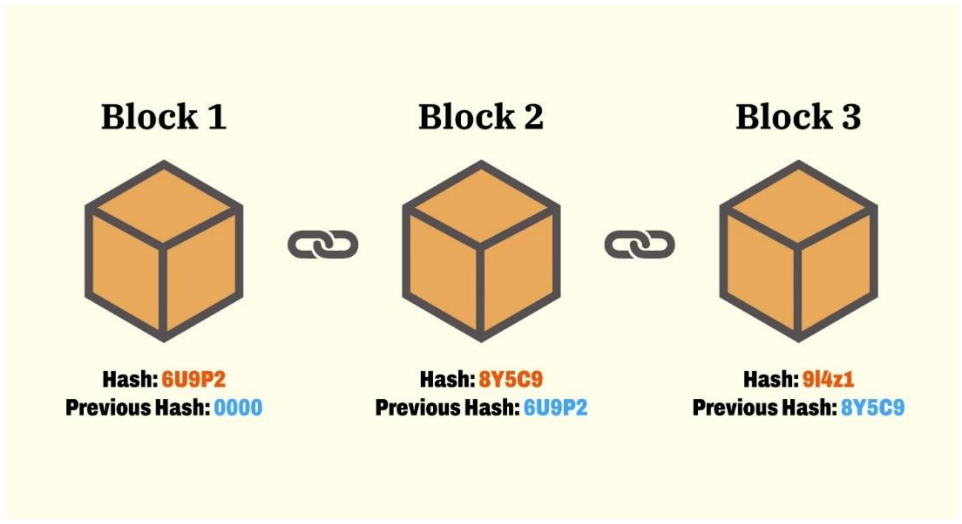


Figure 1: Chaining of blocks in a Blockchain

Source: https://img.money.com/2022/06/What-Is-Blockchain-Infographic.jpg

There are primarily 2 types of blockchain, namely: permissionless (public) blockchain and permissioned (private) blockchain [1]. In permissionless blockchain, any user can publish a new block, as there is no restriction on reading the blockchain and publishing new blocks. This could enable malicious users to publish blocks and undermine the integrity of the system. However, this is prevented by utilizing consensus mechanisms that achieve distributed agreement about the state of the blockchain. Some common consensus mechanisms are proof of work (PoW), proof of stake (PoS), and proof of elapsed time (PoET) [1]. Unlike permissionless blockchain, permissioned blockchains are controlled and maintained by an authority that determines which user can publish a new block. Therefore, it is possible to restrict read access. In the case of permissionless blockchain, the consensus models are generally slow and consume a lot of resources. Contrarily, the consensus models are generally faster and computationally less expensive in case of permissioned blockchain since the establishment of a user's identity is required to join such a network that creates a level of trust between the users.

Some blockchain platforms such as Ethereum have a cost associated with performing a transaction or computation on the network, known as gas. In the case of Ethereum, fees are priced in tiny fractions of the cryptocurrency ether (ETH)—denominations called gwei equivalent to $10^{-9}$ ETH. Gas is used to pay validators for the resources needed to conduct transactions [2].

## 2.2 Homomorphic Encryption

Homomorphic encryption is a form of encryption that allows specific types of computations to be carried out on ciphertexts and generates an encrypted result which, when decrypted, matches the result of operations performed on the plaintexts [3]. Mathematically, if x and y are plaintexts and E represents encryption, homomorphic encryption satisfies the addition and multiplication properties shown in Equation (1). The addition (+) and multiplication (*) operations on the right-hand side of the formula are not the usual addition and multiplication; they are the operations in the space of encrypted texts, namely ciphertexts.

$$E(x+y) = E(x) + E(y)$$
$$E(x*y) = E(x) * E(y)$$

Equation (1): Homomorphic Addition and Multiplication

This cryptographic technique preserves data privacy by enabling secure computations on sensitive information while it is in encrypted state. There are two main types of homomorphic encryption: partial and full. Partially homomorphic encryption only allows the addition of ciphertexts whereas fully homomorphic encryption allows addition as well as multiplication of ciphertexts. Paillier homomorphic encryption is one of the partially homomorphic encryption algorithms that supports addition of ciphertexts whereas Brakerski-Fan-Vercauteren (BFV), Brakerski-Gentry-Vaikuntanathan (BGV) and Cheon-Kim-Kim-Song (CKKS) are some of the fully homomorphic encryption algorithms that allow for both addition and multiplication of ciphertexts [4].

Homomorphic encryption has various applications across numerous domains including healthcare, data analytics, finance, and cloud computing. In cloud computing, it allows the data to be kept securely on the server which can be processed without decrypting. Healthcare applications include secure analysis of medical data without compromising the privacy of patients which allows for collaborative research [5]. In finance, homomorphic encryption allows secure computation of transactions and analysis of data [5].

### 2.3 Zama

Before introducing Zama, it is essential to mention Ethereum which is a major blockchain-based platform that allows execution of smart contracts [6]. Smart contracts are programs deployed on the blockchain and executed by computers running that blockchain [1]. Zama is an Ethereum Virtual Machine (EVM) based blockchain that supports computation on encrypted values [7]. Like Ethereum, which has a currency known as Ether, Zama has its own currency, called ZAMA. The basic idea behind Zama is to provide confidential smart contracts. This means the data sent to or received from the smart contract is encrypted and cannot be read if the data transfer is intercepted.

Zama uses asymmetric encryption and hence uses two keys: public and private. The public key, also known as global key, is stored publicly on-chain, and is used by every user to encrypt their data and perform calculations on that encrypted data. The global key is generated during a setup phase by the initial validators, and securely re-shared when the validator set is changing [7].  This allows mixing of encrypted data from multiple users and across multiple smart contracts. The private key is used to decrypt the data and is not owned by any single user. Instead, pieces of the private key are distributed among validator nodes in the network. To decrypt the data, several validators must cooperate and approve this action. This collaborative approval is termed a "Threshold Protocol" and the participation of at least one-third of the validators is necessary to perform decryption [7]. This method enhances security through decentralization, preventing rogue misuse of the key.

Zama is quite promising because, to the best of our current knowledge, it is currently the only platform that provides a library which allows fully homomorphic calculations to be performed on smart contracts. Solidity, a statically typed curly-braces programming language designed for developing smart contracts that run on Ethereum [8], can be used to implement smart contracts on Zama. Zama uses TFHE scheme (also known as CGGI, from the names of the authors Chillotti-Gama-Georgieva-Izabachène), which is a fully homomorphic encryption scheme [9]. Apart from traditional calculations on encrypted data such as addition, subtraction, multiplication, and division, Zama also provides comparisons that include equals, greater than, less than, greater than equal to, and less than equal to. The libraries for these operations provided by Zama along with their features and limitations will be discussed in the Methodology section.

# CHAPTER 3. LITERATURE REVIEW

There have been numerous studies that have attempted to introduce data security and integrity not only when it is at rest but also when it is in use through various means including homomorphic encryption. This section includes a brief overview of recent major work conducted on data security and integrity when data is in use.

Liang et al. [10] proposes integration of blockchain and homomorphic encryption to address the challenges in circuit copyright protection. Their study proposes a homomorphic encryption-based mathematical model within the blockchain that secures the transactions while also ensuring integrity and confidentiality of data by utilizing smart contracts.

Yaji et al [11] has proposed a system that utilizes Goldwasser-Micali and Paillier encryption schemes for the comparative evaluation study with a focus on data privacy techniques using blockchain technology for AI applications. The study found that attacks on blockchain such as collision, preimage and attack on the wallet can be avoided through encrypting blocks using proposed Goldwasser-Micali and Paillier encryption schemes.

Mutlu et al. [12] has proposed a system that uses blockchain technology and homomorphic encryption which enables third parties (researchers) to perform linear regression on encrypted data. They used Pallier algorithm to calculate the sum required for linear regression through smart contracts. The data owner encrypts the data using the public key of the researcher and sends it to the smart contract where the calculation is performed. The encrypted result can then be accessed by the researcher who can decrypt it on their system using their private key.

Vanin et al. [13] proposes a model to secure Personal Health Record (PHR) that uses interplanetary protocol file system based on distributed hash tables (DHT) along with blockchain. PHR metadata is stored on the blockchain and shared across the network, while PHR data is stored off-chain through the IPFS network. They use two elements: Data Steward (DS), which is responsible for storing PHR on behalf of the individual, and Shared Data Vault (SDV), which is a temporary IPFS storage area where health institutions can access PHR with the consent of the individual. Encrypted data is available to the public through statistical portals where they can perform operations on the data using homomorphic encryption to get meaningful results. For this purpose, they have used Microsoft Simple Encrypted Arithmetic Library (SEAL) library which implements BFV algorithm in JavaScript.

Umar et al. [14] has proposed a model for e-voting using Paillier algorithm. Once the voter casts their vote, it is encrypted homomorphically. A new block of the transaction is then created which contains the encrypted ballots, the pseudonymous address of the voter and the admin, the timestamp of the block creation, the hash of the previous block of the transaction, as well as the hash of the current block. Then the new block of the transaction is mined using the consensus mechanism. After mining, the new block is committed to the ledger of the blockchain. This process continues until the end of the election after which the admin tallies the encrypted votes using the Paillier algorithm which yields the final sum which can then be decrypted to get the results.

Shrestha et al. [15] has conducted a study which analyzes the security concerns with Internet of Things (IoT). One of the major concerns that has been highlighted is the privacy of data. This study also explores the possibility of integrating IoT with blockchain and with homomorphic encryption. The benefits include data immutability, unforgettability, removing single point of failure, along with confidentiality of data.

Kroger et al. [16] has conducted a study which analyzes small IoT devices that are usually hidden and overlooked from a security lens. This study concluded that many of these devices often contain personal and sensitive information about the user such as GPS location, and daily activities (cooking, grooming, washing dishes) that can be easily obtained because of the lack of security measures such as access control. This study further encourages classifying these sensors as sensitive and taking measures to secure that information.

Zhu et al [17] has proposed a secure and privacy-preserving body sensor data collection and query scheme, named SPCQ, for outsourced computing to address the privacy issues of sensitive personal data associated with body sensors. This scheme is basically a special weighted Euclidean distance contrast algorithm (WEDC) for multi-dimension vectors over encrypted data, based on an improved homomorphic encryption technology over composite order group. With the SPCQ scheme, the confidentiality of sensitive personal data, the privacy of data users' queries and accurate query service can be achieved in the cloud server. In addition, this scheme was also implemented on an embedded device, smart phone, and laptop with a real medical database.

The proposed system in this study can be compared to the work proposed by Liang [10], Yaji [11] and Mutlu [12] in the sense that it utilizes blockchain technology along with homomorphic encryption to provide privacy of data while it is being used. However, the above-mentioned studies utilize the Pallier algorithm, which is partially homomorphic whereas this study used fully homomorphic system that not only allows addition but multiplication as well on the encrypted data. Thus, allowing a broader range of computations to be performed.

# CHAPTER 4. METHODOLOGY

This study explores the use of a blockchain based system that allows performing mathematical calculations on encrypted data by utilizing fully homomorphic encryption on the blockchain using smart contracts. We propose and implement a system that allows data to be securely collected and stored on a smart contract and provides descriptive statistics of the data to users in a secure manner. In particular, this implementation includes the calculation of mean, median, and variance of the data. The system provides confidentiality of data in transit, at rest as well as in use. The system is implemented in Zama because, to the best of our current knowledge, it is the only platform that implements fully homomorphic encryption for smart contracts and provides a library for utilizing homomorphic calculations in Solidity, the programming language of our choice for smart contracts.

## 4.1 Description of the System

The system comprises of two main actors namely: data owner and researcher (figure 2).

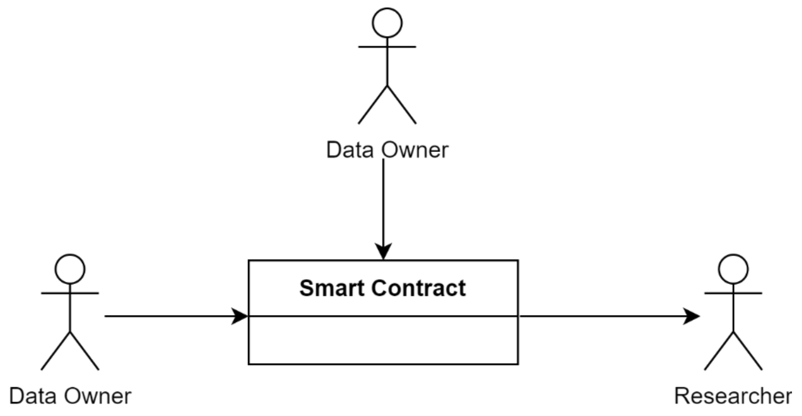


Figure 2: Actors in the proposed system

The data owner is a user or a sensor that provides the data whereas the researcher is a user who needs the descriptive statistics of the data. For example, consider a scenario of a healthcare application. Data owner (the hospital administration) possesses information of the patients which includes sensitive data such as name, age, address along with medical diagnosis and the prescriptions. The patients as well as the hospital can encrypt that relevant data and send it to the smart contract where the desired analysis is performed, and the result is sent to the researcher. In this scenario, the confidential information of the patients is never revealed. As described later in this section, the data is encrypted when it leaves the data owner and is never decrypted even when the computations are performed on it. Furthermore, the result of these computations is sent to the researcher in encrypted form, whenever the analysis is requested.

An overview of the interaction of data owner and researcher with the smart contract is shown in Figure 3. Initially the contract is deployed using ethers.js library. Once the contract is compiled, it produces the Application Binary Interface (ABI) that allows users to interact with the smart contract.

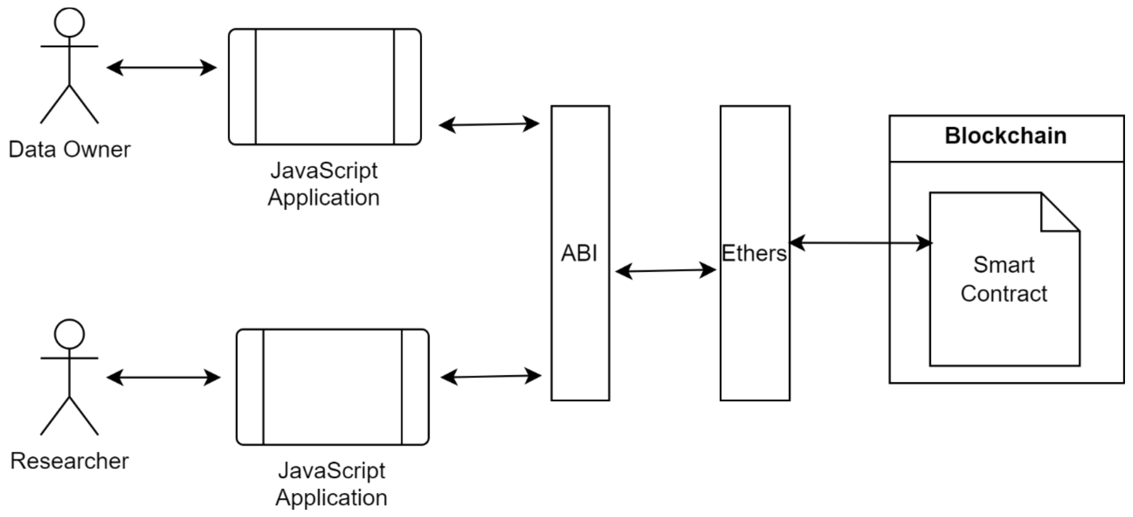


Figure 3: Overview of Interaction with the Smart Contract

The ABI specifies the functions available in the smart contract along with their parameters and return types, allowing users to interact with the smart contract. The data owner and researcher both have their JavaScript application, that utilizes the ABI and methods provided by Ethers library, to communicate with the smart contract. The data owner sends data to the smart contract and the researcher requests descriptive statistics. Figure 4 shows a detailed flow of the interaction of actors with the smart contract.

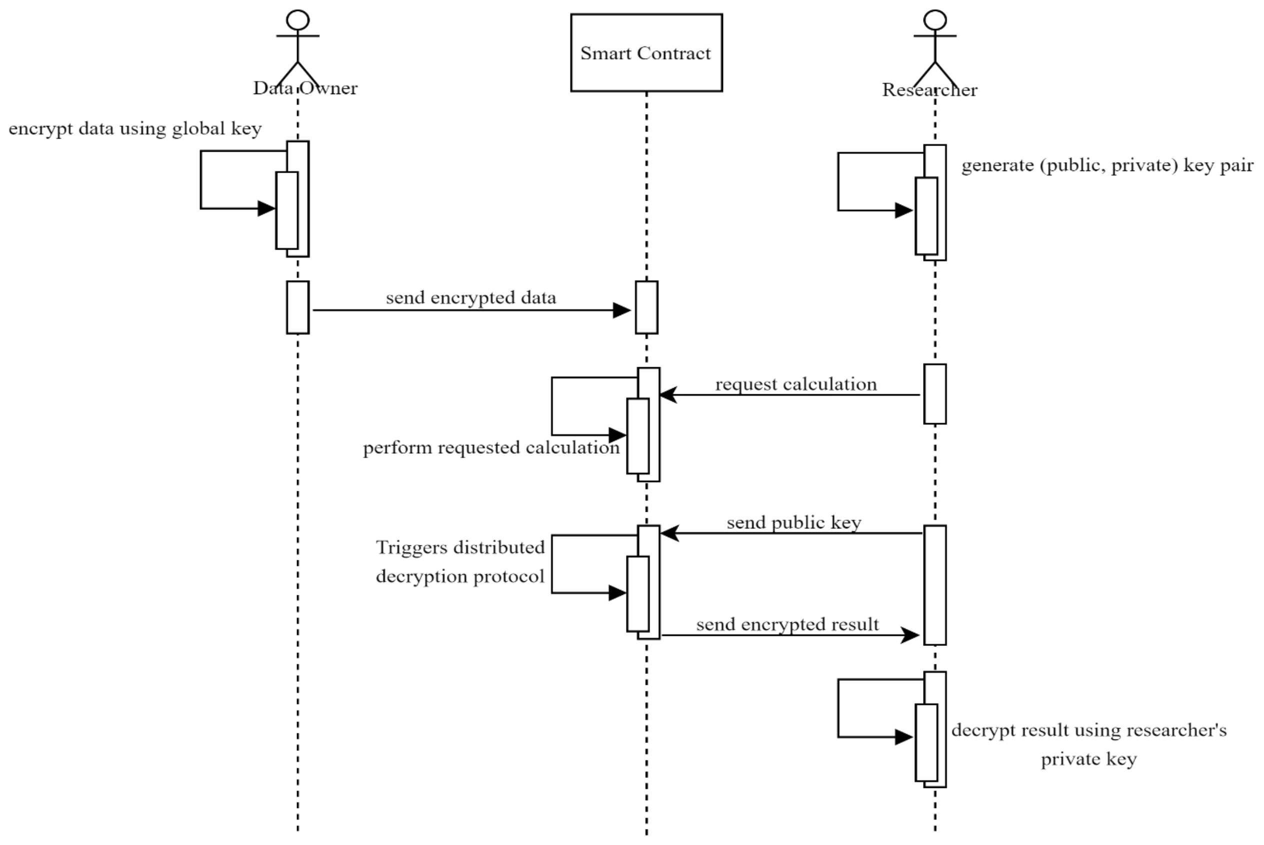

Data Owner
Smart Contract
Researcher
encrypt data using global key
generate (public, private) key pair
send encrypted data
request calculation
perform requested calculation
send public key
Triggers distributed decryption protocol
send encrypted result
decrypt result using researcher's private key

Figure 4: Flow of interaction with the smart

## 4.2 Implementation

The proposed system is implemented on the Zama Blockchain using the libraries for homomorphic calculations provided by Zama. Following is a list of tools and technologies that were used to implement the system:

- Solidity: A programming language that is used to write smart contracts on various platforms including Ethereum.
- Node.js: It is a JavaScript runtime environment which allows execution of JavaScript code outside the web browser. Version 18.18.0 was used for this system.
- Remix: It is a browser based Integrated Development Environment (IDE) used for development and deployment of smart contracts.
- MetaMask: It is a cryptocurrency wallet which is used to connect and interact with blockchain.

- Visual Studio Code: It is an IDE that supports development in various programming languages including JavaScript.
- Ethers.js: It is a JavaScript library that allows interaction with smart contracts. Version 6.10.0 was used for this system.
- FhEVM: It is a library provided by Zama which allows creation of confidential smart contracts on EVM using Solidity. Version 0.3.0 was used for this system.
- FhEVMjs: It is a JavaScript library provided by Zama that allows interaction with smart contracts. Version 0.3.2 was used for this system.

The fhEVMjs library provides encypt8, encrypt16 and encrypt32 functions for encrypting numbers. In these functions 8, 16 and 32 represent the number of bits. In the implementation of the proposed system, all three encrypt functions were utilized to get a better understanding of the impact of the input size on the practicality of the system.

Other methods provided by Zama that have been utilized by the proposed system include:

- Add: A method provided by fhEVM that takes two encrypted integers as parameters and returns their encrypted sum.
- Sub: A method provided by fhEVM that takes two encrypted integers as parameters (a, b) and returns the encrypted result by subtracting the second parameter from the first (a - b).
- Mul: A method provided by fhEVM that takes two integers as parameters, multiplies these numbers, and returns the encrypted result.
- Div: A method provided by fhEVM that takes one encrypted integer and one plaintext integer as paraments, divides encrypted integer by plaintext integer and returns the result which is also an encrypted integer.
- Cmux: A method provided by fhEVM, which takes a condition as the first parameter (comparison of encrypted numbers), encrypted number or Boolean as the second parameter and an encrypted number of Boolean as the third parameter. The datatype of second and third

parameters must be same. This function returns the second parameter if the condition is true and the third parameter if the condition is false.

- Eq: A method provided by fhEVM which takes two encrypted numbers (a, b) as parameters, compares whether a is equal to b and returns an encrypted Boolean based on whether the condition is true or false.
- Gt: A method provided by fhEVM which takes two encrypted numbers (a, b), compares whether a is greater than b and returns an encrypted Boolean based on whether the condition is true or not.
- Reencrypt: A method provided by fhEVM which returns an encrypted value after re-encrypting it using the public key that is provided as the parameter. It accepts three parameters, out of which the third is optional. First parameter is the encrypted value that is to be re-encrypted, second is the public key which would be used to re-encrypt the value and third is the default encrypted value which would be returned if re-encryption is unsuccessful.

Three major statistical operations mean, median, and variance were implemented on the encrypted data. Each statistical operation is performed in three different versions based on the bit size of the data types. For example, in the case of median there is one smart contract which calculates the median only for 8-bit values, another calculates median for 16-bit values and the third calculates median for 32-bit values. For all the statistical operations, random numbers are being generated and stored in array instead of being hard coded. These random numbers act as the data which is encrypted and sent to the smart contract. Once the script runs, the user is prompted for the upper limit of the number. For example, if the user enters 40, only numbers from 1 to 40 would be generated. The next prompt is for the number of elements to be generated; this is the length of the array. This allows proper testing of the logic and provides a level of control in terms of data size. The implementation of each statistical operation will be further explained after a discussion of the challenges with implementation.

### 4.2.1 Challenges

While implementing the system, the following limitations and challenges were identified in Solidity and in Zama.

- Data Types: fhEVM only supports unsinged integers that are either 8-bit, 16-bit, and 32-bit. Furthermore, the methods provided by fhEVM for calculations on encrypted data such as addition, subtraction, multiplication, and division return an encrypted unsigned integer. In case of division, there is an added limitation where an encrypted integer can only be divided with a plaintext integer. This means it is not currently possible to divide two encrypted integers. Even if there was a possibility of such division, the result would always be an encrypted integer because of its return type, proving that it is not possible to work with decimal numbers. Moreover, the integers are unsigned, therefore, it is not possible to handle negative integers either. These limitations restrict the number of analyses that can be performed on the smart contract.
- Gas Limit: Homomorphic calculations are quite expensive in terms of resources and since these calculations are performed on the smart contract, it consumes a lot of gas as well. Gas consumption depends on the operations that are performed on the encrypted data and increases with the complexity of the algorithm, bits of datatype (32-bit operations would consume more gas compared to 8-bit operations). Currently there is a gas limit of 10,000,000 on Zama devnet [18] which also restricts the functionality in terms of the number of data points that can be sent and analyzed.
- Solidity: Unlike traditional programming languages such as Python and Java, Solidity has various limitations. One of which is that the data structure used to store key pair values (mappings) is quite limited in terms of functionality and it is not possible to iterate over keys or values. This makes it challenging to implement various algorithms such as calculation of mode (the most frequent value in an array).

### 4.2.2 Calculation of Mean

Mean is calculated by taking the sum of each element in the sequence and then dividing it by the number of elements (equation 2).

$$Mean = \frac{\sum x_i}{n}$$

Equation (2): Formula for calculating Mean.

In the designed system, only the sum of the sequence on the smart contract was calculated, and division operation was performed at the researcher's end. So, the smart contract sends the sum as well as the number of elements in encrypted form. The reason for this is because of the limitation mentioned in section 4.2.1 that division will always return an integer whereas mean can be a decimal value.

The sum of the encrypted sequence is calculated when the researcher requests the calculation. Once the result is calculated, it is re-encrypted using the researcher’s public key. The re-encryption is achieved using the reencrypt method provided by fhEVM library in a secure manner as mentioned in section 4.1. The encrypted result is then sent to the researcher along with the encrypted number of elements (n) who decrypts the result using their private key and then performs the final division to get the result.

### 4.2.3 Calculation of Median

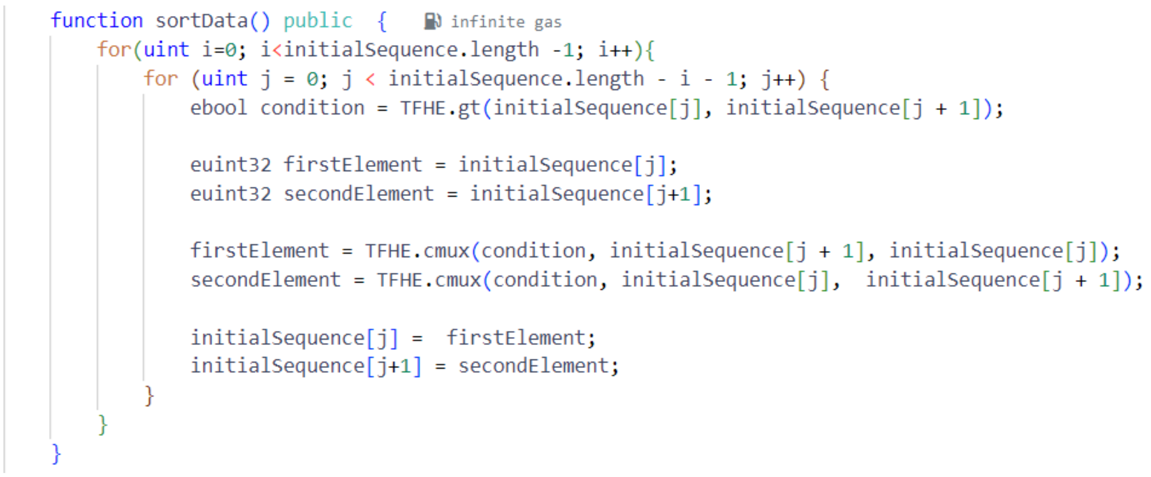
Calculating median is a simple task, assuming the data is sorted. If the total number of elements in the array is odd, median is the middle element, i.e. the element whose position is calculated by adding number of elements (n) by 1 and dividing the result by 2 (equation 3). If, however, the total number of elements in the array is even, median is calculated by taking the average of the middle two elements (equation 3).

$$Median = \begin{cases} \left(\frac{n+1}{2}\right)^{th} element\ if\ n\ is\ odd \\ \frac{\left(\frac{n}{2}\right)^{th} element + \left(\frac{n}{2}+1\right)^{th}\ element}{2}\ if\ n\ is\ even \end{cases}$$

Equation (3): Formula for calculating Median.

In the system, this differentiation is simply handled by using an if statement. If n is odd, the median is calculated on the smart contract and returned to the researcher in encrypted form. This is done because adding an odd number by one returns an even number and if that number is divided by 2, the result will always be an integer. This removes the possibility of getting a decimal result. In case n is even, the sum of the middle two elements is calculated on the smart contract and sent to the researcher. The result is then decrypted by the researcher and divided by 2 to get the median. In this case the division is not performed on the smart contract because there is no guarantee that the average will be an even number so there is a possibility that the result will not be accurate because of the limitation discussed in section 4.2.1. Thus, the researcher makes the decision of performing the division based on the number of elements. The biggest challenge in calculating median, however, was the implementation of a sorting algorithm on the smart contract since it is not a trivial task to sort encrypted data. We chose to implement the bubble sort algorithm because of its simplicity. The algorithm works by comparing two adjacent values in an array and swaps the numbers based on the result of that comparison which makes it easy to implement. Furthermore, implementing bubble sort does not require any additional data structure. This made it suitable to write in Solidity, considering the limitations such as gas consumption. FhEVM provides methods for comparing two encrypted numbers, however, the return type of these methods is eBool (encrypted boolean) which is not compatible with a conventional if statement. Initial attempt to overcome this limitation involved decrypting the condition and passing it to the if statement which solved the problem. However, this approach made the algorithm very expensive computationally because the decrypt method consumes a lot of gas. It is, therefore, not recommended by Zama and will be deprecated in the future versions.

Thus, it was not possible to sort an array of more than 3 elements. To increase the feasibility and efficiency of the algorithm, the use of the decrypt method was removed from our implementation. Instead, the cmux method (described in section 4.1) was used. This change considerably increased the efficiency of the sorting algorithm in terms of gas consumption and time.  Figure 5 below shows the implementation of bubble sort algorithm.



```
function sortData() public  {    infinite gas
    for(uint i=0; i<initialSequence.length -1; i++){
        for (uint j = 0; j < initialSequence.length - i - 1; j++) {
            ebool condition = TFHE.gt(initialSequence[j], initialSequence[j + 1]);

            euint32 firstElement = initialSequence[j];
            euint32 secondElement = initialSequence[j+1];

            firstElement = TFHE.cmux(condition, initialSequence[j + 1], initialSequence[j]);
            secondElement = TFHE.cmux(condition, initialSequence[j],  initialSequence[j + 1]);

            initialSequence[j] =  firstElement;
            initialSequence[j+1] = secondElement;
        }
    }
}
```

Figure 5: Implementation of bubble sort algorithm on encrypted data

In the designed system, when the researcher requests median calculation, the encrypted data is first sorted and then the median is calculated. The result is re-encrypted using the researcher's public key. The re-encryption is achieved using the reencrypt method provided by fhEVM library in a secure manner as mentioned in section 4.1. The encrypted result is then sent to the researcher who decrypts the result using their private key to get the median.

### 4.2.4 Calculation of Variance

The formula for calculating variance is given in Equation (4) where $\bar{x}$ represents the mean of the data.

$$V = \frac{\sum(x_i - \bar{x})^2}{n-1}$$

Equation (4): Formula for calculating Variance.

It is not feasible to calculate mean of any sequence on the smart contract because the division function in the Zama fhEVM library always returns an encrypted integer. To eliminate the calculation of mean in the variance formula, we derived an equivalent formula that did not include mean or any division operator other than the major division operation. The step-by-step derivation of the formula starting with equation (4) is shown in equation (5).

$$V = \frac{\sum(x_i - \bar{x})^2}{n-1}$$

$$V = \frac{\sum(x_i - \frac{\sum x_i}{n})^2}{n-1}$$

$$V = \frac{\sum\left[{x_i}^2 - 2x_i\frac{\sum x_i}{n} + \left(\frac{\sum x_i}{n}\right)^2\right]}{n-1}$$

$$V = \frac{n^2}{n^2}\frac{\sum\left[{x_i}^2 - 2x_i\frac{\sum x_i}{n} + \frac{\left(\sum x_i\right)^2}{n^2}\right]}{n-1}$$

$$V = \frac{\sum\left(n^2{x_i}^2 - 2n^2x_i\frac{\sum x_i}{n} + n^2\frac{(\sum x_i)^2}{n^2}\right)}{n^2(n-1)}$$

$$V = \frac{n^2\sum {x_i}^2 - 2n\sum x_i \sum x_i + \sum(\sum x_i)^2}{n^2(n-1)}$$

$$V = \frac{n^2\sum {x_i}^2 - 2n\sum\left(x_i \sum x_j\right) + \sum\left(\sum x_j\right)^2}{n^2(n-1)}$$

$$V = \frac{n^2\sum {x_i}^2 - 2n\sum\left(x_i \sum x_j\right) + n\left(\sum x_j\right)^2}{n^2(n-1)}$$

Equation (5): Derivation of the formula for calculating variance.

This approach removed calculation of mean from the algorithm and in turn removed division as well. However, as shown in equation 5, it was not possible to eliminate division completely. Keeping the limitations in mind, only the calculation of numerator was implemented in the smart contract. The encrypted result of the numerator was sent to the researcher along with the encrypted value of n where the final division was performed. Upon closer inspection of equation 5, it was observed that there were three major components in the numerator itself. Furthermore, the numerator involved subtraction as well. As discussed earlier, Zama only provides unsigned integers which means any result below zero would undermine the accuracy of the result. To mitigate the issue, these three components were calculated separately, and their result was stored in separate variables. Then the first component was added to the third, and the second component was then subtracted from the result of addition. This ensured that the result would not fall below zero.

In the designed system, the calculation of numerator of the variance is initiated when it is requested by the researcher. Once the calculation is complete, the result is re-encrypted using the researcher's public key. The re-encryption is achieved using the reencrypt method provided by fhEVM library in a secure manner as mentioned in section 4.1. The encrypted result is then sent to the researcher along with the encrypted number of elements (n) who decrypts these values using their private key. The final division is then performed by the researcher to get the result.

The calculations for variance with an array having 6 elements could not be performed for 32-bit datatypes because of the limitation on gas consumption.

### 4.2.5 Calculation of Mode

The attempt to implement mode (most frequent value in a list) was unsuccessful because of the limitations mentioned in 4.2.1, specifically the limitation of mappings data structure in solidity and the methods provided by Zama. While attempting to implement an algorithm to calculate mode, it was discovered that encryption is probabilistic which means that the encrypted value for a single plaintext will not always be the same. For example, consider an array having 4 elements. [2, 4, 1, 4]. Once the elements of this array are encrypted, hypothetically, it would become: [d5048, e2314, x3215, o9849]. Notice that the number four appears twice in the plaintext array at index 1 and index 3 but its encrypted version does not have any repetitions and the values at index 1 and index 3 are different. This is not necessarily a limitation, but it made the implementation even more complex. It was not possible to simply use mappings, assuming that the encrypted number was stored as a key and the frequency (count) was stored as the value, to compare the keys in a loop and increment the counter whenever the keys matched. There was a need for another data structure (array) which held the unique elements. This array had to be checked in each iteration to see whether the current element was already in mappings or not. In this case using the cmux method was not suitable, and a conventional if statement was needed which is not compatible with ebool. Hence, the mode function was not implemented in this study.

# CHAPTER 5. RESULTS AND DISCUSSION

In this chapter, we present the results of our study along with a discussion of the challenges and implications of these results in two subsections. The first subsection includes the timing data and its analysis for the functions implemented in the study. The second subsection focuses on how the proposed system provides the security properties of integrity, availability, and confidentiality when data is in transit, at rest, and in use.

## 5.1 The timings of the functions

Three statistical operations, mean, median, and variance were performed on encrypted data. The data collected was the time it took to send encrypted data to smart contract, and the time it took to perform calculations on that encrypted data. Since there is a gas limit associated with Zama, the gas estimation for each of these tasks was also recorded.

It was possible to send more than 6 data points to the smart contract, however, not all statistical operations could be performed on more than 6 numbers. For example, in the case of variance calculation for 32-bit datatype, the calculation was unsuccessful because the gas consumption exceeded the limit set by Zama.

Table 1 shows the time taken and estimated gas consumption to send encrypted data to the smart contract. It includes timings for the three different sizes of input (8-bit, 16-bit, and 32-bit). The table shows that the estimated gas consumption provided by Zama increases with the number of data points. However, there is negligible increase across the different number of bits for a specific number of data points.

Table 1: Time for Sending Data

| Number of Data Points | Time to send data (milliseconds) | Estimated Gas |
|---|---|---|
| | 8-bit | |
| 3 | 5673.02 | 219545 |
| 4 | 17957.12 | 285400 |
| 5 | 12993.89 | 352285 |
| 6 | 17332.09 | 420089 |
| | 16-bit | |
| 3 | 29649.73 | 220081 |
| 4 | 9633.12 | 268932 |
| 5 | 14136.23 | 353092 |
| 6 | 14169.20 | 421228 |
| | 32-bit | |
| 3 | 5540.57 | 221067 |
| 4 | 34451.06 | 287391 |
| 5 | 9765.06 | 354715 |
| 6 | 18110.06 | 423208 |

Figure 6 provides a comparison of the time it takes to send encrypted data to the smart contract for 8, 16, and 32 bits. As is visible in the graph, the time consumption does not follow a specific pattern but instead shows fluctuation. We believe this is probably because of the level of congestion of the blockchain network.

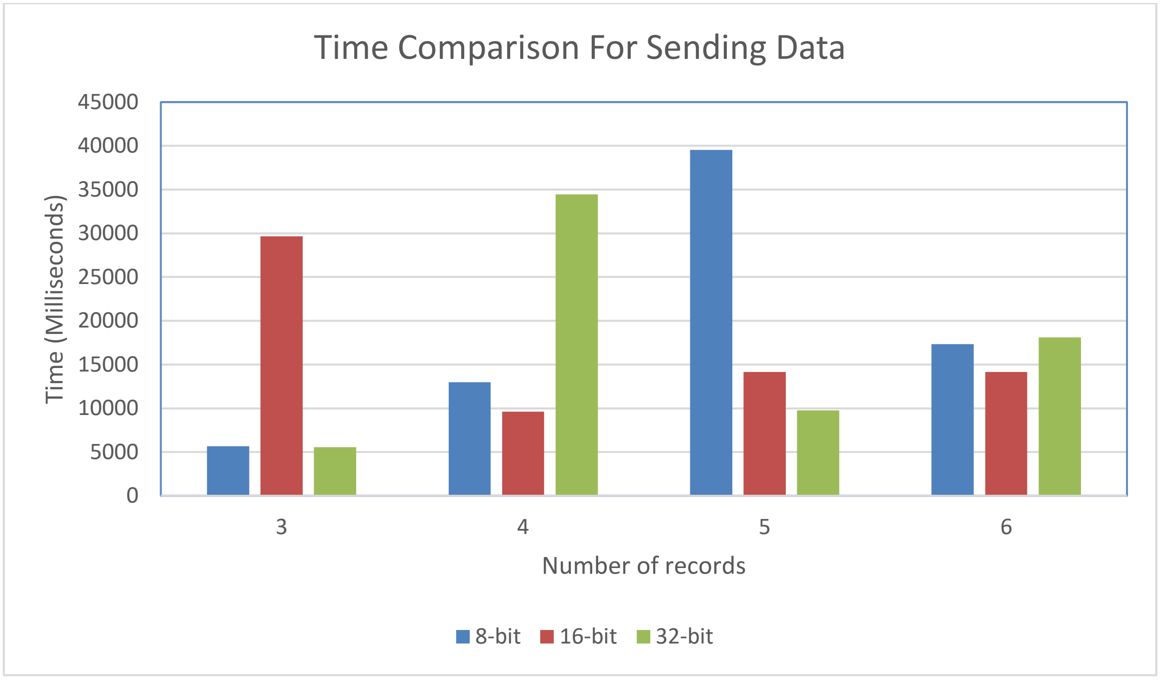


Figure 6: Time for sending data with respect to data size and number of data

Table 2 shows the time taken and the estimated gas consumption to sort the encrypted data and to calculate the median of the sorted data. The results show a gradual increase in estimated gas consumption for sorting data. However, there is fluctuation in terms of estimated gas consumption for median calculation. Moreover, the estimated gas consumption to sort is higher compared to the estimated gas consumption for median calculation since sorting is more complex.

Table 2: Time for Sorting Data and Median Calculation

| Number of Data Points | Sorting Time (milliseconds) | Estimated Gas | Median Calculation Time (milliseconds) | Estimated Gas |
|---|---|---|---|---|
| 8-bit | | | | |
| 3 | 19119.22 | 575130 | 12450.73 | 32748 |
| 4 | 18261.09 | 1130740 | 16533.62 | 156539 |
| 5 | 39525.12 | 1869667 | 9271.73 | 32748 |
| 6 | 15503.17 | 2791913 | 15653.9 | 7598 |
| 16-bit | | | | |
| 3 | 10052.12 | 640518 | 5271.85 | 32748 |
| 4 | 33625.33 | 1261516 | 12563.37 | 186539 |
| 5 | 14046.25 | 2087629 | 12533.72 | 32748 |
| 6 | 18571.57 | 3118856 | 13602.48 | 186539 |
| 32-bit | | | | |
| 3 | 18373.43 | 736518 | 21975.28 | 32748 |
| 4 | 32987.43 | 1453516 | 9723.58 | 216539 |
| 5 | 19111.54 | 2407629 | 21931.56 | 32748 |
| 6 | 35707.49 | 3598856 | 5736.78 | 216539 |

Figure 7 provides a comparison of the time it takes to sort encrypted data on the smart contract for 8, 16, and 32 bits. The graph appears to be following a pattern of gradual increase with the number of bits at some instances. However, the pattern does not appear to be constant and there are some fluctuations. Specifically, in the case of sorting 5 numbers of 8-bit datatype where the time consumption is significantly higher compared to the time taken to sort 3 and 4 numbers of the same bit size.

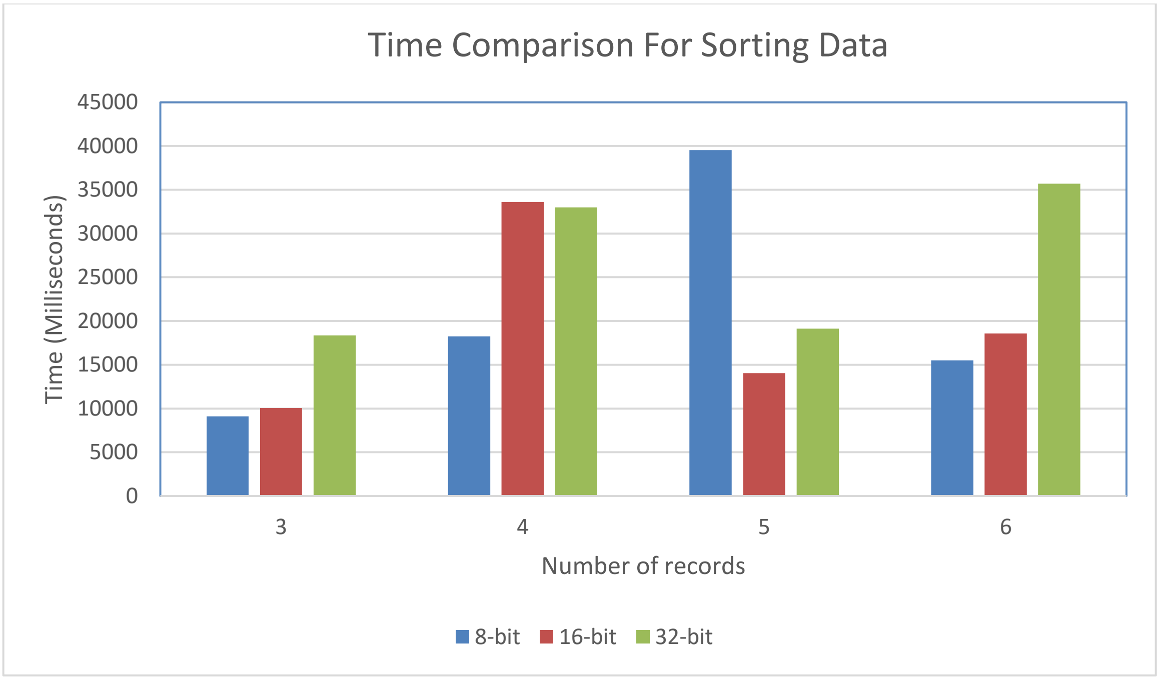


Figure 7: Time for sorting data with respect to data size and number of data points

Figure 8 provides a comparison of the time it takes to calculate median on the smart contract for 8, 16, and 32 bits. The graph does not follow any specific pattern which indicates that the time consumption is variable.

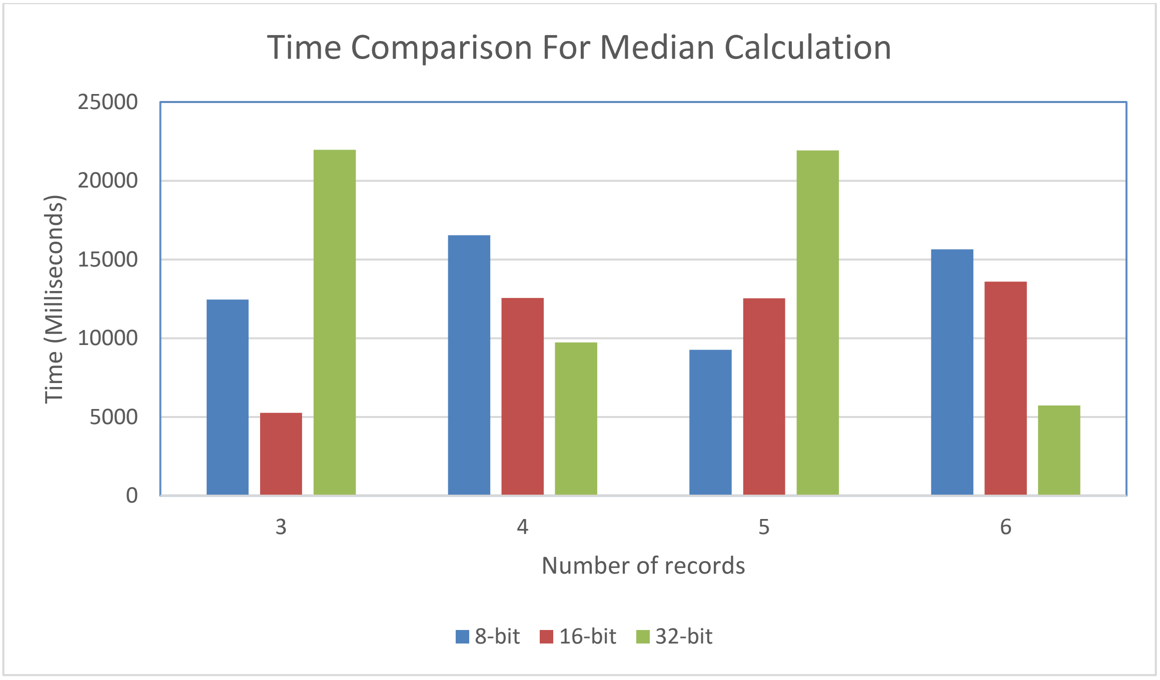


Figure 8: Time for calculating median w.r.t data size and number of data

Table 3 shows the time taken to calculate the mean of the encrypted data along with the estimated gas consumption. Like the result for sorting and median calculation, the calculation of mean shows a gradual increase in terms of gas consumption with the increase in number of records and number of bits. However, a slight variation can be observed specifically while calculating mean of 4 32-bit numbers which is quite lower than expected.

Table 3: Time for Mean Calculation

| Number of Data Points | Time to Calculate Mean (milliseconds) | Estimated Gas |
|---|---|---|
| 8-bit | | |
| 3 | 5355.20 | 403744 |
| 4 | 20547.40 | 527516 |
| 5 | 11903.95 | 651288 |
| 6 | 13870.02 | 775060 |
| 16-bit | | |
| 3 | 5570.09 | 493845 |
| 4 | 9920.67 | 647617 |
| 5 | 5717.84 | 801389 |
| 6 | 16122.90 | 955161 |
| 32-bit | | |
| 3 | 11709.23 | 565875 |
| 4 | 31972.38 | 21000 |
| 5 | 19665.36 | 951665 |
| 6 | 14069.88 | 1135437 |

Figure 9 provides a comparison of the time it takes to calculate mean on the smart contract for 8, 16, and 32 bits. Like the previous graphs, mean calculation does not follow a pattern in case of time consumption which means that time fluctuates based on the quality of the network and on the congestion of the blockchain network.

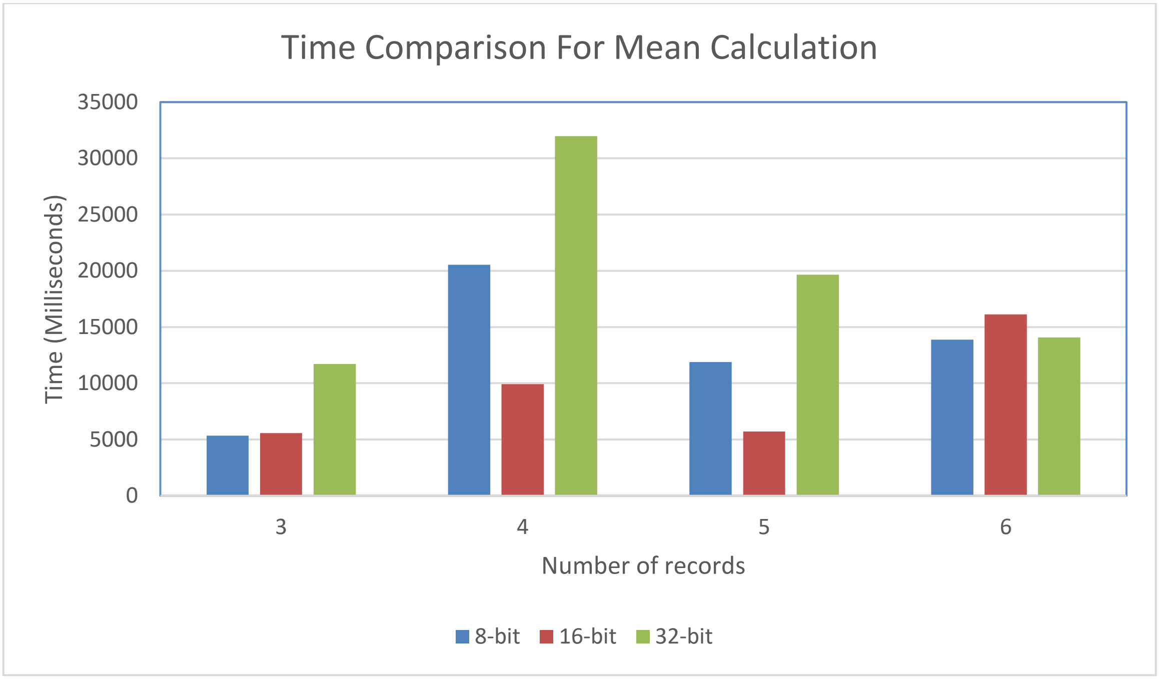


Figure 9: Time for calculating mean w.r.t data size and number of data

Table 4 shows the time taken to calculate the variance of the encrypted data along with the estimated gas consumption. The calculation of mean shows a gradual increase in terms of estimated gas consumption with the increase in number of records and number of bits.

Table 4: Time for Variance Calculation

| Number of Data Points | Time to Calculate Variance (milliseconds) | Estimated Gas |
|---|---|---|
| 8-bit | | |
| 3 | 14822.20 | 3711137 |
| 4 | 36876.56 | 4480525 |
| 5 | 24371.67 | 5249913 |
| 6 | 29637.00 | 6019301 |
| 16-bit | | |
| 3 | 22344.63 | 4706440 |
| 4 | 47008.47 | 5685828 |
| 5 | 26634.87 | 6665216 |
| 6 | 36003.35 | 7644604 |
| 32-bit | | |
| 3 | 48343.56 | 6386740 |
| 4 | 59402.02 | 7696128 |
| 5 | 46764.37 | 9005516 |

Figure 10 provides a comparison of the time it takes to calculate variance on the smart contract for 8, 16, and 32 bits. Unlike the previous graphs, there appears to be a pattern for time consumption which increases with the increase in bit size. However, there is variation in terms of the number of records. Meaning that time taken to calculate variance for an array having 5 elements is lower than the time taken to calculate variance for an array having 4 elements.

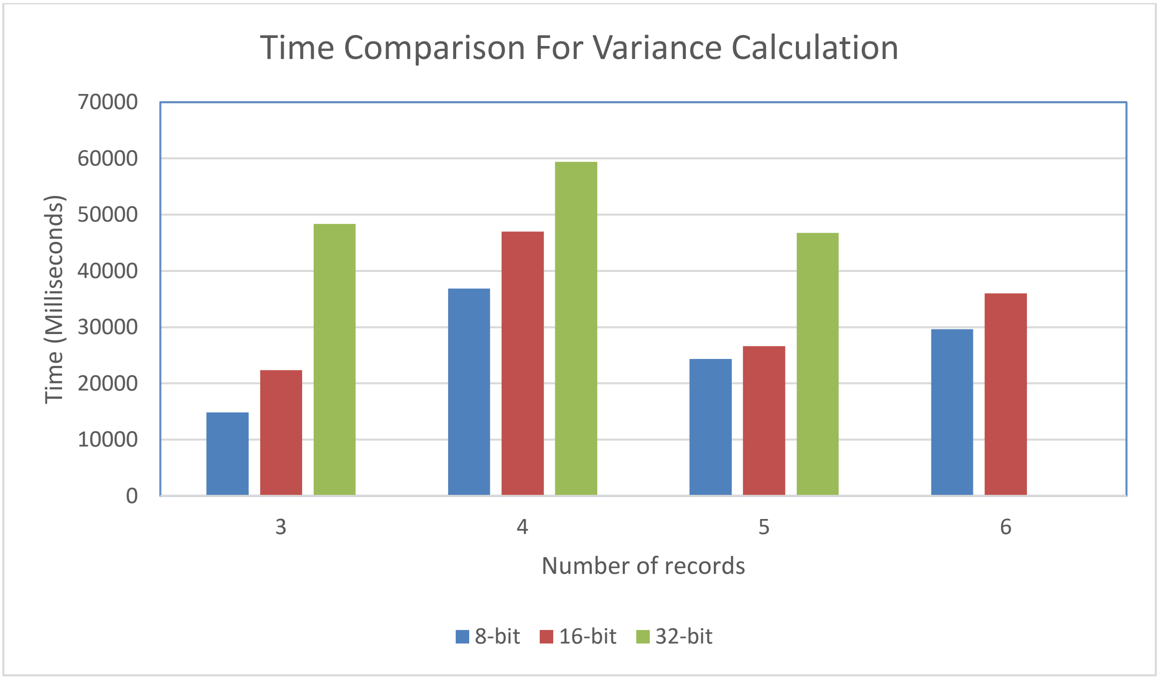


Figure 10: Time for calculating variance w.r.t data size and number of data

### 5.2 Security of the Proposed System

This system ensures three key aspects of security that are confidentiality, integrity, and availability. Each of these attributes will be discussed in the following sections.

**Availability**: Availability is a feature that is provided by blockchain itself because of its decentralized nature. Every user in the network has their own copy of the ledger so even if one node becomes unavailable, the whole system itself would still be accessible. This removes a single point of failure and introduces fault tolerance in the system.

**Integrity**: Integrity is another feature that is provided by blockchain itself since it is tamper resistant. This is possible because each block contains the hash of the previous block which creates a chain.

Tampering with one block changes the hash which would then have to be updated in the next block. Apart from that, every user has a copy of the ledger, so the tampered copy would simply be rejected. Since smart contracts reside on the blockchain, they cannot be tampered with. In that sense this system provides trust in computations because the algorithm resides on the smart contract.

**Confidentiality**: In case of public blockchain, data present on the blockchain is visible to everyone, therefore it does not provide confidentiality. This system bridges that gap by using homomorphic encryption. Data is encrypted by the data owner, using the global key generated through Zama. The encrypted data is sent to the smart contract, where the calculations are performed, and the encrypted result is then sent to the researcher where it is decrypted. Thus, the plaintext is never visible to the researcher, and they only get the result of computations in which they are interested. The result of computations is never completely decrypted on the smart contract even during the process of re-encryption which is requested by the researcher to decrypt the result using their private key. In that sense, the system provides security of data in use which is one of the major advantages of homomorphic encryption. Apart from homomorphic encryption, this feature is only provided by special hardware systems that might not be accessible for many users.

# CHAPTER 6. Conclusion

In this study, Zama was used to provide a system that allows encryption/decryption of data and library in Solidity which enables fully homomorphic computations on the smart contract. This study explored the possibility and practicality of using fully homomorphic encryption on a blockchain to perform descriptive statistics on data.

The results obtained indicate that the time taken to send the data to the smart contract and to perform the calculations was variable; it fluctuated with the quality of internet connection and how busy the blockchain network was but overall, it increased with the number of records and with the number of bits. The estimated gas consumption, however, showed a pattern of gradual increase with the number of records and the number of bits along with the complexity of the algorithm.

The proposed system provides confidentiality of data in use by utilizing the libraries provided by Zama that allow encryption of data and fully homomorphic calculations on that data. Furthermore, the system also provides computational integrity by implementing the algorithms for statistical operations on the smart contract that resides on the blockchain. Hence making it tamper resistant and allowing trust in computations.

While this study successfully demonstrates that it is possible to secure data on blockchain using fully homomorphic encryption, there are certain challenges and areas for improvement. Fully homomorphic operations are quite expensive in terms of resources. This combined with the limited data structures and features provided by Solidity itself, limits the number of calculations that can be performed. Furthermore, there is a lack of options when it comes to the number of libraries or technologies available to implement fully homomorphic encryption in solidity language since the community is small as compared to other technologies or languages such as Java.

Future studies in this area can incorporate more complex algorithms with a wider data range as more resources become available and technology becomes more advanced. Another possible future direction would be to include access control to the current system. Meaning that the data owner and researcher need to be verified before they can send the data or before they can receive the result of the calculations.

## References

[1] D. Yaga, P. Mell, N. Roby, and K. Scarfone, "Blockchain technology overview," NISTIR, vol. 8202, 2018.

[2] Investopedia. (n.d.). Gas (Ethereum). [Online]. Available: https://www.investopedia.com/terms/g/gas-ethereum.asp [accessed May. 6, 2024].

[3] X. Yi, R. Paulet, and E. Bertino, "Homomorphic encryption" in Homomorphic Encryption and Applications, *Springer*, 2014, pp. 27–46. [Online]. Available: https://doi.org/10.1007/978-3-319-12229-8_2

[4] V. F. Rocha and Julio López, "An Overview on Homomorphic Encryption Algorithms," Technical Report IC-PFG-18-28, Final Graduation Project, November 2018.

[5] P. Oh, "Exploring Homomorphic encryption and its benefits" [Online]. Available: https://medium.com/@patrick-oh-sglion65/exploring-homomorphic-encryption-and-its-benefits-d43f821eae9c [accessed May. 6, 2024]

[6] S. Tikhomirov, "Ethereum: State of Knowledge and Research Perspectives," in *Foundations and Practice of Security*, A. Imine, J. Fernandez, JY. Marion, L. Logrippo, and J. Garcia-Alfaro, Eds., vol. 10723, *Springer*, Cham, 2018. [Online]. Available: https://doi.org/10.1007/978-3-319-75650-9_14.

[7] M. D. Clément, D. Danjou, D. Demmler, T. Frederiksen, P. Ivanov, M. Joye, D. Rotaru, N. Smart, L. Tremblay, and T. Thibault, "fhEVM Confidential EVM Smart Contracts using Fully Homomorphic Encryption,". [Online]. Available: https://github.com/zama-ai/fhevm/blob/main/fhevm-whitepaper.pdf

[8] Solidity, "Solidity - Solidity 0.8.10 documentation," Soliditylang.org. [Online]. Available: https://soliditylang.org/. [accessed May. 1, 2024]

[9] I. Chillotti, "TFHE Deep Dive - Part I - Ciphertext types" [Online]. Available: https://www.zama.ai/post/tfhe-deep-dive-part-1 [accessed May. 1, 2024]

[10] W. Liang, D. Zhang, X. Lei, M. Tang, K. Li, and A. Zomaya, "Circuit Copyright Blockchain: Blockchain-Based Homomorphic Encryption for IP Circuit Protection," in 2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), 2019, pp. 9–13.

[11] S. Yaji, K. Bangera, and B. Neelima, "Privacy Preserving in Blockchain based on Partial Homomorphic Encryption System for AI Applications," in 2020 International Conference on Advances in Computing and Communication Engineering (ICACCE), 2020, pp. 1–5.

[12] Z. Mutlu, Y. Peker, and A. Selçuk, "Blockchain-based Privacy Preserving Linear Regression," in 2018 26th Signal Processing and Communications Applications Conference (SIU), 2018, pp. 1–4.

[13] F. N. d. S. Vanin, L. M. Policarpo, R. d. R. Righi, S. M. Heck, V. F. da Silva, J. Goldim, and C. A. da Costa, "A Blockchain-Based End-to-End Data Protection Model for Personal Health Records Sharing: A Fully Homomorphic Encryption Approach," Sensors, vol. 23, no. 1, pp. 14, 2023. [Online]. Available: https://doi.org/10.3390/s23010014

[14] B. Umar, O. Olaniyi, D. Olajide, and E. Dogo, "Paillier Cryptosystem Based ChainNode for Secure Electronic Voting," in 2020 IEEE 4th International Conference on Blockchain (Blockchain), 2020, pp. 153–157.

[15] Shrestha, R., & Kim, S. (2019, Month). Integration of IoT with blockchain and homomorphic encryption: Challenging issues and opportunities. Advances in Computers (Vol. 115, pp. 293-331). https://www.sciencedirect.com/science/article/abs/pii/S0065245819300269.

[16] J. Kröger, "Unexpected Inferences from Sensor Data: A Hidden Privacy Threat in the Internet of Things," in Internet of Things. Information Processing in an Increasingly Connected World, L. Strous and V. Cerf, Eds. Springer, Cham, 2019, pp. 147-159. doi: 10.1007/978-3-030-15651-0_13.

[17] H. Zhu, L. Gao, and H. Li, "Secure and Privacy-Preserving Body Sensor Data Collection and Query Scheme," Sensors, vol. 16, no. 2, p. 198, Feb. 2016.

[18] Zama, "FHEVM," 2020. [Online]. Available: https://docs.zama.ai/fhevm/v/0.3-2/how-to/gas. [Accessed: May 3, 2024]