Implementation of Division algorithm and Ensemble Voting in Fully Homomorphic Encryption Scheme

Project-II (CH47024) report submitted to

Indian Institute of Technology Kharagpur

in partial fulfilment for the award of the degree of

Bachelor of Technology

in

Chemical Engineering

by

Vatsal Mitesh Tailor

(17CH10047)

Under the supervision of

Professor Ayantika Chatterjee

Advanced Technology Development Centre

Indian Institute of Technology Kharagpur

Spring Semester, 2020-21

April 12, 2021

DECLARATION

I certify that

(a) The work contained in this report has been done by me under the guidance of my supervisor.

(b) The work has not been submitted to any other Institute for any degree or diploma.

(c) I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.

(d) Whenever I have used materials (data, theoretical analysis, figures, and text) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references. Further, I have taken permission from the copyright owners of the sources, whenever necessary.

Date: April 12, 2021 (Vatsal Mitesh Tailor)

Place: Kharagpur (17CH10047)

ADVANCED TECHNOLOGY DEVELOPMENT CENTRE

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

KHARAGPUR - 721302, INDIA

CERTIFICATE

This is to certify that the project report entitled "**Implementation of Division algorithm and Ensemble Voting in Fully Homomorphic Encryption Scheme**" submitted by **Vatsal Mitesh Tailor** (Roll No. 17CH10047) to Indian Institute of Technology Kharagpur towards partial fulfilment of requirements for the award of degree of Bachelor of Technology in Chemical Engineering is a record of bona fide work carried out by him under my supervision and guidance during Spring Semester, 2020-21.

Date: April 12, 2021

Place: Kharagpur

Professor Ayantika Chatterjee
Advanced Technology Development Centre
Indian Institute of Technology Kharagpur
Kharagpur - 721302, India

Abstract

Name of the student: **Vatsal Mitesh Tailor** Roll No: **17CH10047**

Degree for which submitted: **Bachelor of Technology**

Department: **Advanced Technology Development Centre**

Thesis title: **Implementation of Division algorithm and Ensemble Voting in Fully Homomorphic Encryption Scheme**

Thesis supervisor: **Professor Ayantika Chatterjee**

Month and year of thesis submission: **April 12, 2021**

Fully homomorphic encryption scheme allows calculations directly on the ciphertext. While the operations addition, subtraction and multiplication have been widely implemented, the implementation of division using TFHE library is scarce. Here a successful implementation of the division operation using the non-restoring division algorithm is demonstrated, thus completing the set of 4 arithmetic operations, paving way for implementation of numerous other complicated algorithms.

Ensemble models are producing state-of-the-art results for numerous machine learning tasks. We have here implemented a very popular ensemble learning technique Voting, in Fully homomorphic encryption domain, thus bypassing the need of decrypting the data before computation

Acknowledgements

Firstly, I would like to thank my guide Professor Ayantika Chatterjee for guiding me thoughtfully and efficiently through this project, giving me an opportunity to work at my own pace along my own lines, while providing me with very useful directions whenever necessary.

I would also like to thank Pradeep Kumar Reddy (PhD Research Scholar, ATDC, IIT Kharagpur) for guiding me with the theoretical basics and the experimental works of the project. I would like to sincerely thank everyone who knowingly or unknowingly helped me accomplish the project deliverables.

Contents

List of Figures

Abbreviations

FHE	**F**ully **H**omomorphic **E**ncyrption
TFHE	Fast Fully Homomorphic Encyrption over Torus
SVM	**S**upport **V**ector **M**achine
MSB	**M**ost **S**ignificant **B**it

Chapter 1

Introduction

1.1 Encryption

The process of converting data into code, such that the code is neither readable nor interpretable, is called encryption. Changing the above stated code back to original data, with the use of a special key is called decryption. The Encryption scheme that we are to talk about here can be viewed as an extension of asymmetric encryption, where there are two "key" in play: public key and a private key Public Key is widely available to everyone, not a secret knowledge, and with this key one can encrypt data that is from data to unreadable code. 5 Private Key, on the other hand, is secret knowledge, and with this key one can decrypt the data that is from unreadable code to data. Encryption is very important are is allows one to safeguard data.

Give a brief of the chapter and introduce what you will talk about.

Homomorphic Encryption This form of encryption allows one to do computation directly on cyphertext, thus generating an encrypted result, which when decrypted will be equal to the result which we would get in case the operation was performed on plaintext. In simple terms, taking the + operation, Encrypted(3) +

Encrypted(4) will yield Encrypted(7), which works in line with the operations performed on plaintext, that is 3 + 4 = 7. Homomorphic encryption can be divided into three main parts:

• Partial Homomorphic Encryption

Select mathematical operations can be performed here in encrypted domain

• Somewhat Homomorphic Encryption

Limited mathematical operations can be performed here for a limited set number of times

• Fully Homomorphic Encryption

Allows mathematical operations without limitations

1.2 Fully Homomorphic Encryption

This is the gold-standard of homomorphic encryption, an encryption system which will support arbitrary operations on the cypher texts. In this case, one never has to decrypt any data, and thus one can give the computation such data to an untrusted party as the data can never be revealed. The applications of Fully Homomorphic Encryption are growing with each passing day, especially in the domains of cloud computing.

1.3 TFHE - Fast Homomorphic Encryption over Torus

The TFHE library homomorphically can compute any binary gate on encrypted data. The library can evaluate binary gates homomorphically at the rate of 50 gates per second per core, without decrypting the input. The TFHE library is used by us here to implement Division.

Chapter 2

Division

2.1 Introduction to Division

The inverse operation of multiplication, division forms one of the 4 arithmetic operations.

Terminology If we divide 25 by 3, we can represent the calculation as 25/3 and thus 25 is the Numerator and 3 is the Denominator.

An alternative terminology we resort to and which is used further is:

- 25 is Dividend
- 3 is Divisor

The operation yields 8 as Quotient and 1 as Remainder

There are numerous algorithms to achieve the above stated algorithm so of which being:

- Long Division
- Restoring Division
- Non-Restoring Division
- SRT Division
- Newton-Raphson Division
- Goldschmidt Division

FHE operations are far slower than normal operations, we need a fast algorithm with no error. Given the limitations posed by the operations available in TFHE library, Non-Restoring Division was the way to go, of the above stated algorithms

2.2 Perspective Division Algorithms

Euclidean Division The simplest of all division algorithm, we reject this algorithm in the face of Non-Restoring division, as it involves evaluating more gates than the later algorithm. While it is easier to implement, the algorithm will be slower than non-restoring division

Restoring Division The nearest to the selected division algorithm, this algorithm falls in the class of slow division methods, with non-restoring division, but it goes on to use potentially twice as many operations as non-restoring division

Newton Ralphson Division While faster than the chosen Non-restoring division method, handling decimal places in binary form will increase the number of cipherbits to store. While faster than Non-Restoring Division, it does not yield precise division results. Thus, we might encounter collosal errors in applications where repeated division is required.

Goldschmidt Division An example of iterative algorithm similar to Newton-Ralphson, this too does not yeild precise division results, and thus we might not be able to use it for all implementations

2.3 Non-restoring division Algorithm

This division algorithm solely relies on operations, Addition and Subtraction, to get the Quotient and Remainder for the given Divisor and Dividend.

Flow Chart The below is the flowchart of a modified non-restoring division algorithm. The algorithm is modified such that there is an ease in implementing it with the gates handy in the TFHE library.

At the end of the execution of this flow chart, A yields the remainder and ciphertext1 yields the quotient

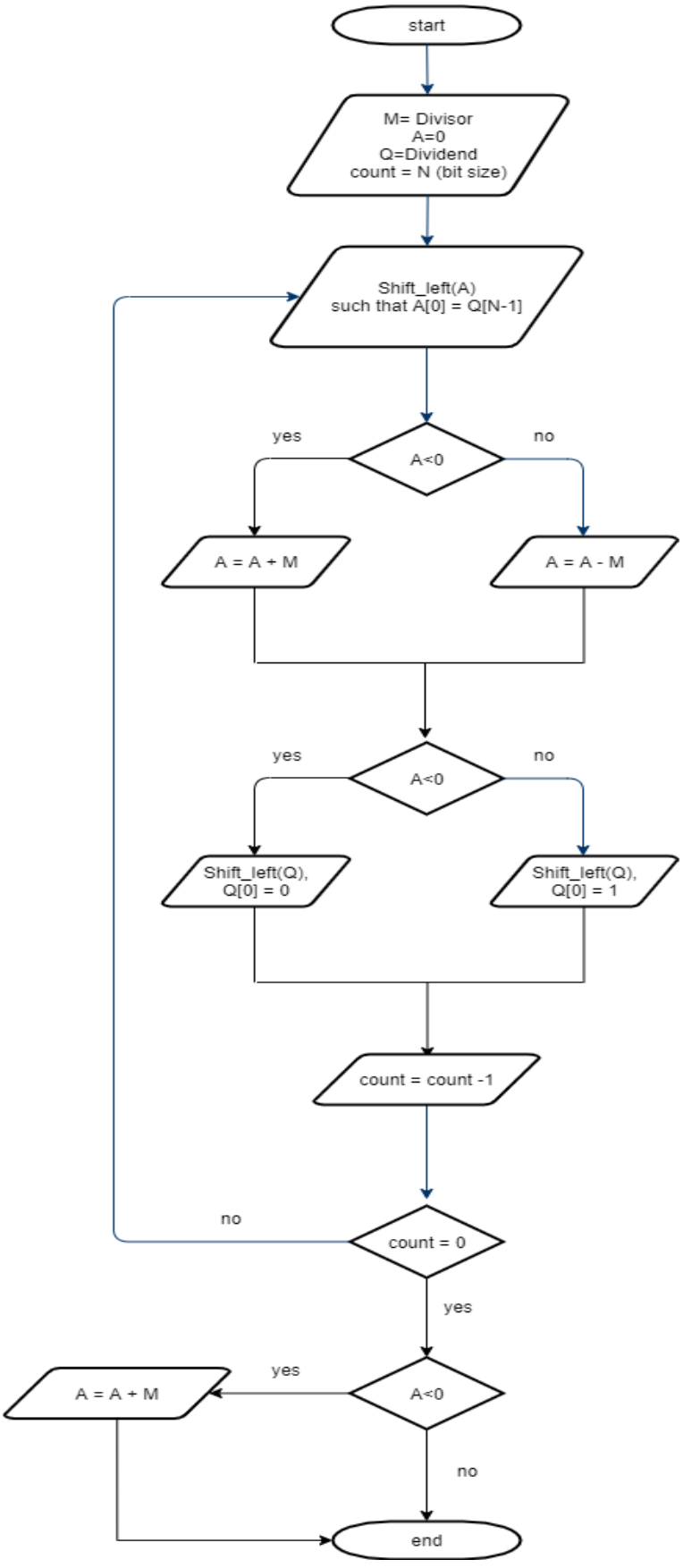

FIGURE 2.1: Flow Chart - modified non-restoring division

Chapter 3

Implementation - Division

The implementation of the above algorithm can be divided can be divided into two chief block which we will name as the following:

1. The Looping Block
2. The After the loop Block

3.1 The Looping Block

This block is executed for the number of times equal to the number of bits we are working on.

This block takes in 4 fixed ciphertexts :

- $ciphertext_{one}$ - The number 1 in ciphertext
- $ciphertext_{zero}$ - The number 0 in ciphertext
- $ciphertext2$ - DENOMINATOR

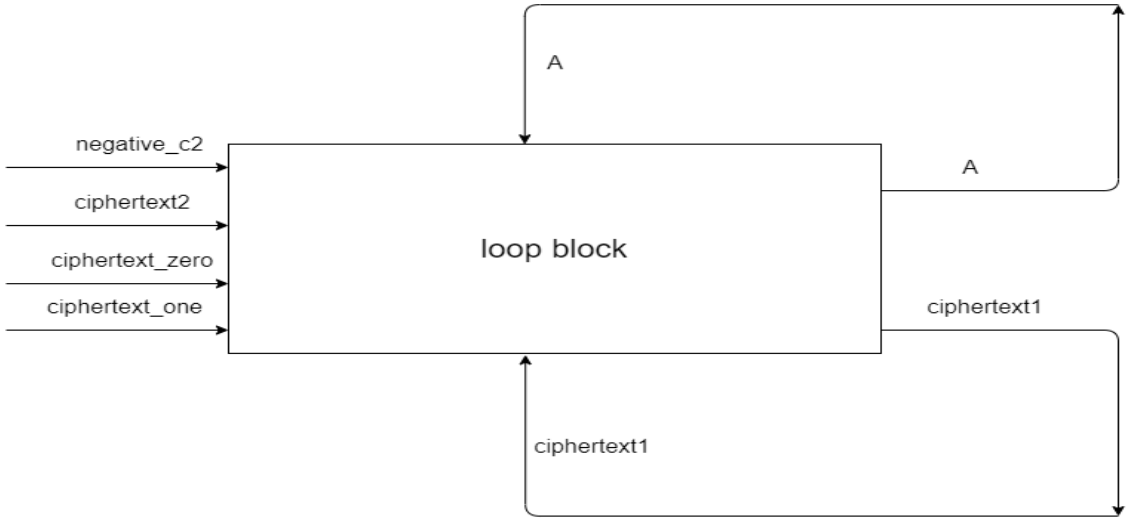

FIGURE 3.1: Diagram - overview of the looping block

- $negative_{c2}$ - negative of ciphertext2

And, each execution of the block updates two ciphertexts :

- A: initialized as 0, it will yield the Remainder
- ciphertext1 : initialized as the NUMERATOR, it will yield the Quotient

The ciphertext1 yielded by this block is the final Quotient. The further working to be done on A to get the remainder is described in the next block.

3.2 The After the loop Block

The value of A that we get from the loop block needs correction if it is negative. To check if the value of A is negative, we subtract A by the cipertext value of 0 and then check the Most Significant Bit(MSB) of the result. If the MSB is 0, then the result is a positive number, thus A>=0. Else, execute the corrections prescribed for A<0.

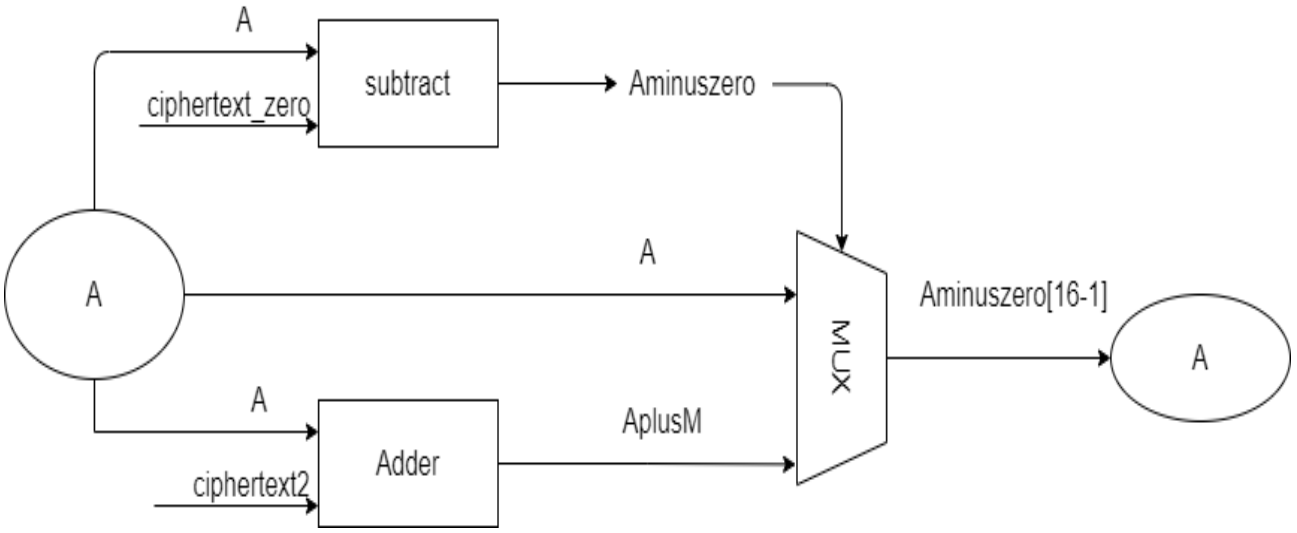

FIGURE 3.2: Diagram - after the loop

3.3 Inside the Looping Block

Once we take a look at the looping part of the flow chart, we notice two conditional blocks. These blocks are implemented in the same way as described in the above section, that is, by checking the MSB.

Note that the MSB implementation for the first conditional block A<0, cannot be done with a single bootsMUX gate. Doing so, we miss out on covering the A= 0 condition. Thus a second bootsMUX gate is added in continuation to the first gate to take care of A=0 condition.

For the cases where we want to assign a particular bit of a given number to 0 or 1, we utilize the Least Significant Bit (LSB) of the ciphertexts of 0 and 1.

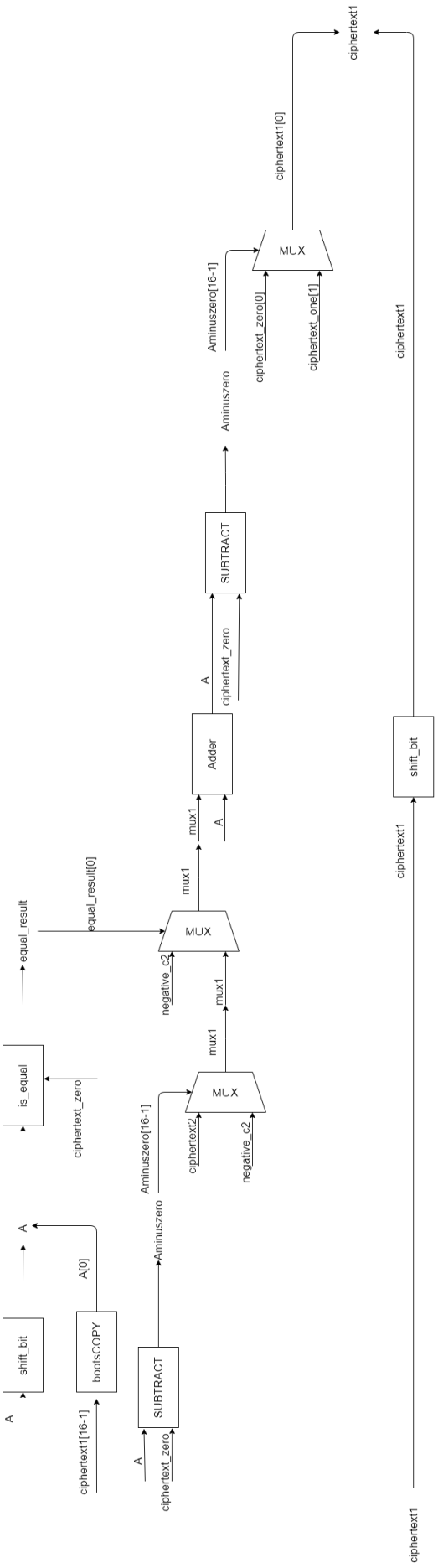

FIGURE 3.3: Diagram - the looping block

3.4 Results

The division algorithm takes on an average a run-time of 94 seconds when the computed numbers are 16 bits.

3.5 Conclusion

The successful implementation of the division block opens pathways for future implementation of numerous algorithms, especially deep learning algorithms where normalization is critical and the normalization layer utilizes division extensively

3.6 Future Work

Optimizing the above algorithm to further reduce the number of gates, by using alternative methods for the conditional operations in the plain text algorithm will reduce the overall run-time. Future plans are to use this division algorithm to fully implement the SVM classifier, and then using the implemented Ensemble Voting block, to create a machine learning ensemble classifier comprising of SVM and KNN models.

Chapter 4

Ensemble Voting

4.1 Ensemble Models

Ensemble Models combine numerous individual learning models and produce predictive results, which are more accurate than the individual learning models themselves. Have been a hot topic since 1990s, ensemble models are still in fashion, as they are continuously producing state-of-the-art results for various tasks, both classification and regression.

4.2 Voting

Voting comes under a subset of ensemble models, combination models. It is used for the case of a classification task. We train numerous classifiers, and from the labels of the individual classifiers, we pick a label, by voting. That is, the class label which receives the most number of votes, or the class label which is predicted by the most number of classifiers is the final answer by voting.

Chapter 5

Implementation - Ensemble Voting

5.1 Implementation and Code

While the code is self-explanatory given the above information, one must take note of the following chief code snippets:

listings

```
for(i=0; i<COLUMNS; i++){
        for(j=0; j<CLASSES; j++){
                bootsCONSTANT(&equal[0], 0, bk);
                bootsCONSTANT(&equal[1], 0, bk);
                is_equal(equal, details[j].class_name, cypher[test_case][i], BLEN, bk);
                Adder(temp, details[j].frequency, equal, BLEN, bk);
                for(k=0; k<BLEN; k++){
                        bootsCOPY(&details[j].frequency[k], &temp[k], bk);
                        bootsCONSTANT(&temp[k], 0, bk);
                }
        }
}
```

The above code snippet, in summary makes a hash table, mapping each class label to its frequency. The frequency is counted for all the class labels in a single traversal of all the classifiers outputs

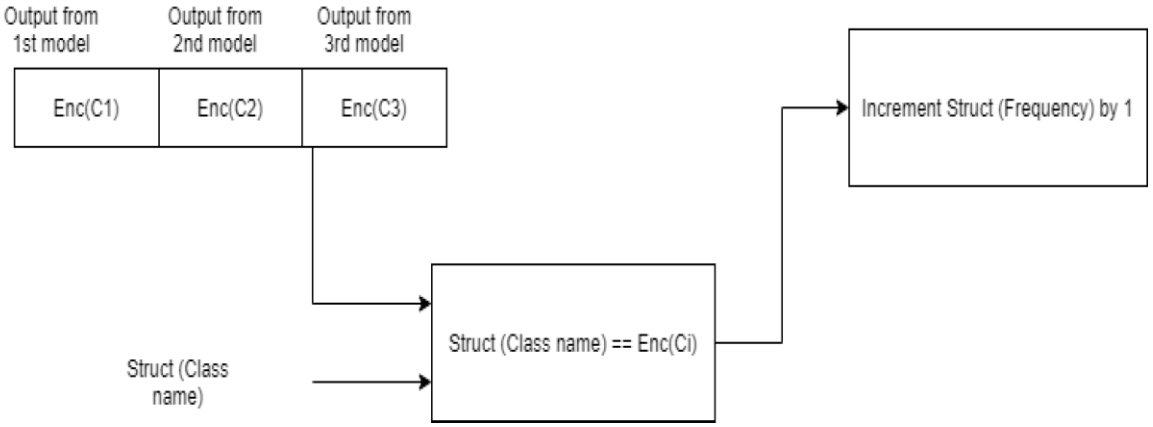

FIGURE 5.1: Diagram - the looping block

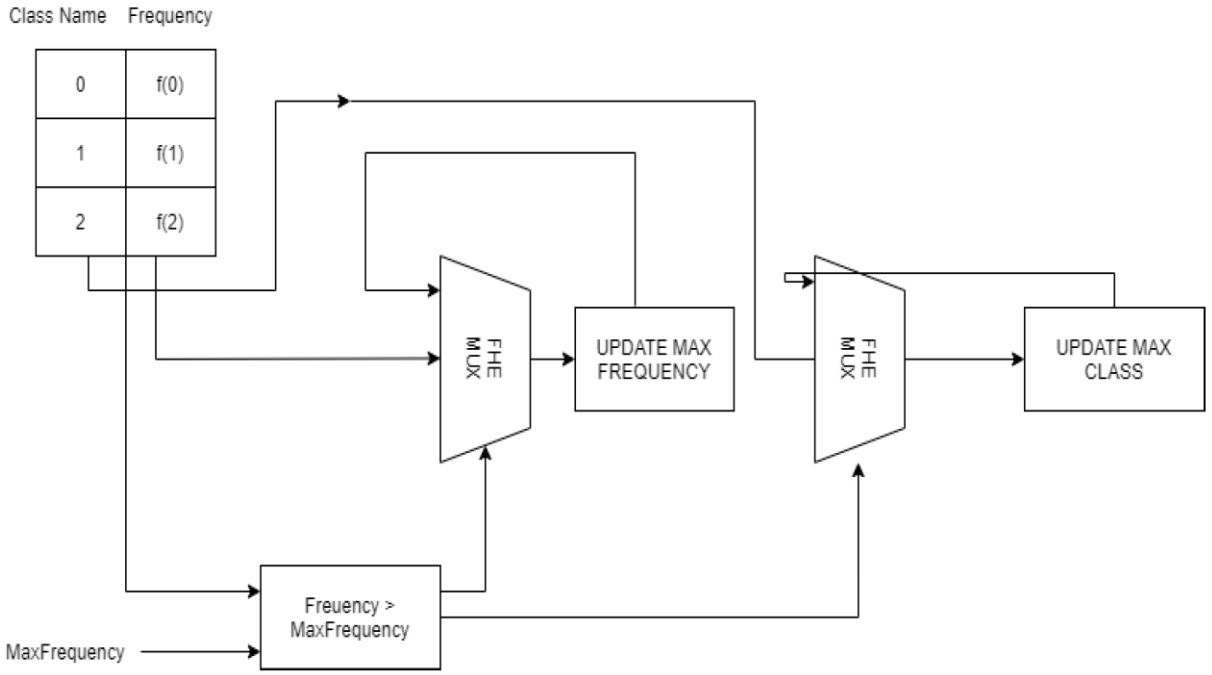

FIGURE 5.2: Diagram - the looping block

```
for(i=0; i<CLASSES; i++){
        bootsCONSTANT(&compare[0], 0, bk);
        subtract(temp2, compare, details[i].frequency, max_frequency, BLEN, bk);
        multiplexer(max_class, details[i].class_name, max_class, compare, BLEN, bk);
        multiplexer(max_frequency, details[i].frequency, max_frequency, compare, BLEN, bk);
}
```

The code snippet follows the first code snippet, and this code in summary will find out the class label having the maximum frequency of all the class labels, from the hash table which the former code has created for us. The approach proposed is supposed to give the best time complexity for the task by the implantation of hash tables.

5.2 Results

At the start of the code, change the BLEN variable to the bit length one wants for the program. The code was tested for 16 bits, 32 bits and 64 bits. The performance of the code:

Number of Bits	Average Runtime(in seconds)
16	19
32	41
64	91

5.3 Conclusions

The successful implementation of this combination ensemble method, now opens paths for implementation of numerous custom ensemble models in the encrypted domain. The Voting module is a universally accepted block or layer in ensemble models.

5.4 Future Work

Now, we can implement other Ensemble learning techniques, like Averaging, Bagging and Boosting. With the promising work on the division operator, we can now also step onto regression tasks, as normalization will be no problem now. Moreover, we

might be able to improve the result timings of our research by resorting to more optimal computations.

Bibliography

Chatterjee, Ayantika, and Indranil Sengupta. "Sorting of fully homomorphic encrypted cloud data: Can partitioning be effective?." IEEE Transactions on Services Computing 13.3 (2017): 545-558.

Gentry, Craig. A fully homomorphic encryption scheme. Vol. 20. No. 9. Stanford: Stanford university, 2009.

Fan, Junfeng, and Frederik Vercauteren. "Somewhat practical fully homomorphic encryption." IACR Cryptol. ePrint Arch. 2012 (2012): 144.

Gentry, Craig. "Fully homomorphic encryption using ideal lattices." Proceedings of the forty-first annual ACM symposium on Theory of computing. 2009.

Hall, Rob, Stephen E. Fienberg, and Yuval Nardi. "Secure multiple linear regression based on homomorphic encryption." Journal of Official Statistics 27.4 (2011): 669.

Chen, Hao, et al. "Logistic regression over encrypted data from fully homomorphic encryption." BMC medical genomics 11.4 (2018): 3-12.

Reddy, B. Praeep Kumar, and Ayantika Chatterjee. "Encrypted Classification Using Secure K-Nearest Neighbour Computation." International Conference on Security, Privacy, and Applied Cryptography Engineering. Springer, Cham, 2019.

Chatterjee, Ayantika, and Indranil Sengupta. "Translating algorithms to handle fully homomorphic encrypted data on the cloud." IEEE Transactions on Cloud Computing 6.1 (2015): 287-300.