SECURE COMPUTATION ON SENSITIVE DATA USING HOMOMORPHIC ENCRYPTION ON ETHEREUM BLOCKCHAIN

Răzvan Şerban[1], Alexandru Vochescu[2], Daniel Dosaru[3]

Due to the proliferation of electronic devices, there is an ongoing creation of new data for each and every human. There is an ongoing debate about data because of the trade off that happens: security of the data versus the convenience of accessing the data. The exclusivity between the two can simply be narrowed by using a homomorphic encryption algorithm over one of the many blockchains available. This allows for the data to be available to the owner at any time, while also permitting for third parties to operate on the data while not leaking anything about the data involved in the computations.

Keywords: Homomorphic encryption, blockchain, data security

1. Introduction

Technology is in a continuous evolution because it is based on certain human demands that need to be fulfilled. This led to the current situation that computers are ubiquitous [1] in our lives. By simply interacting with payment systems, public transport, different businesses, or even government services, we are using the interface of a computer and are creating a lot of data [2]. Most of this data generates other demands from people, that in turn generates more data in a never ending circle. This data could be harmless, but the majority of it is not [3] because no one wants to share with third parties the places they have been or how much money they have spent on what items. There is a paradox that this huge amount of data entails [4] regarding the needs that appear from it. This paradox can be summarized as the 3 needs for data that, nowadays, at any given time, at most 2 of them are checked off. Firstly, we need to process it, but we do not have the computational power, storage capacity or algorithms to get a final useful result. Secondly, because we are using other actors' resources, they can be considered from a security

[1]PhD candidate, Faculty of Automatic Control and Computer Science, University POLITEHNICA of Bucharest, Romania, e-mail: razvan.serban@upb.ro

[2]PhD candidate,, Faculty of Automatic Control and Computer Science, University POLITEHNICA of Bucharest, Romania

[3]PhD candidate, Faculty of Automatic Control and Computer Science, University POLITEHNICA of Bucharest, Romania

standpoint that they are a third party and any data that we have and are giving up freely to them is a liability for us [5] because, as mentioned before, the data that a person usually generates in their day-to-day life is usually sensitive. Thirdly, another need for the data that we create is to be available for whomever we are agreeing to give access to, that is convenience.

With this in mind, we propose a way to combine these 3 seemingly disparate needs for data, by allowing a third party to perform computations on it, not allowing unrestricted third party access to the data, while also allowing access to other parties that we whitelisted.

The rest of the paper is structured as follows. In Section 2 we present the main challenges and the background of the existing status quo, in Section 3 we present a way to reconcile all the three needs for sensitive data presented above, Section 4 discusses an implementation, while Section 5 presents the evaluation and the caveats of the implementation, as the system is a mostly theoretical one, as we have not reached the technological computing speeds for a fully-fledged system. The final section presents our conclusion and future work.

2. Background and challenges

Currently, at the moment of writing this paper, all the three requirements for data that we have mentioned above: privacy, away computation, and convenience, are not fully solved by any system. There are some systems that manage to check 2 of them while not fulfilling the last one [6] [7].

2.1. Homomorphic encryption

The first 2 ones, privacy and the possibility of computing at a third party, are being allowed nowadays by using homomorphic encryption.

Homomorphic encryption [8] [9] is the possibility of doing operations on sets of data while the data stays encrypted. As an example, instead of doing multiplication on 2 clear text numbers, if we encrypt those 2 numbers separately, but with the same key, and then run a modified multiplication operation on them, the final result would be a ciphertext that when decrypted with the same key that was used on the initial numbers, the multiplication result would appear.

This has big security implications regarding the possibility of someone else handling our data, while at the same time not knowing what the data contains. The clear text version of the data is only present on our premises and will never leave the system unencrypted. Anything that happens between the personal system and the system that performs the computations, for example modifying or simply reading the message, is harmless as it happens only on ciphertext. The system that performs the computations will receive the data not in clear text.

2.2. Blockchain

The blockchain [10] [11] is a collective name for technologies that allow for distributed computing. They are, in most scenarios, networks of a group of computing devices that create read-only data and store them as blocks linked by hashes (the N block depends on the hash of the block at N-1) by using the data structure named Merkle Tree [12]. Over the years, blockchains have evolved to allow for different ways of proving that the block N+1 is a block that needs to be created after block N. Proof of Work [13] is one such system in which the miners verify a huge number of calculations, and the threshold for adding a block is for finding a result within certain limits. Proof of Stake [14] and Proof of Location [15] are novel new ways for the network to reach a consensus on the next blocks to be added.

Some blockchains such as Ethereum [17] allow for snippets of code to be run by the miners [18]. Miners are working computer nodes that validate the blocks. As a consequence, they offer a decentralized consensus [19] on the result of that computation. There is a special language used for writing these code snippets called Solidity [20], which was inspired by JavaScript, C++, and Python [21] .

The main element in the Solidity language is called "contract". It is a decentralized class, that lives in the blockchain network and can be instantiated at will by anyone. Like the classes that usual programming languages have, it has fields and methods that change the fields. Anything in the contract is known by everybody [21]. The value of the fields at any point during the life of the contract and the methods that were called, their order of calling, and who called them - the private address. That means that any data put on the blockchain is visible to anybody, while at the same time, the decentralized nature of the network can vouch that the data will be kept as it was intended [22]. If the data is not changing, the data will not become suddenly corrupt or be gone.

3. Design of the system

In this section, we propose a system that allows the 3 aforementioned human needs for data management and processing, to be resolved and to co-exist at the same time.

- The ability to process the data but by using resources, such as memory and computational power, that are not in our possession.
- There is an obligation to give as little information as we can to the third parties that perform computations on the data. These actors mentioned here could range in size from small companies to big companies and in the categorization of them, binning them in public governments versus private enterprises. Any data sent to them in clear text is a liability because of

both the destination of data and the malicious actors that can intercept the data in passing

- Open access to data at any moment is of high importance. To do this, it needs to have high availability. Anytime we want to inspect the data or give the data to a third party to process it, it should be available. Physical boundaries can be overcome by networks that link the storage memory and the computing power, and they need to be powered at all times.

Such a system can be created by combining privacy-aware homomorphic encryption alongside the blockchain capability to be a read-only database.

By implementing the homomorphic encryption methods as functions into a contract on the blockchain, the decentralized system can operate on data that is encrypted and return trustful results, without needing to reveal what is hidden under the encryption.

As the encrypted data is public on the blockchain to use, the convenience of data sharing would amount only to sharing the decryption key of the compounding data and the result to the whitelisted parties interested. The high availability of the data on the blockchain combined with the decentralized verification that the reporter of the encrypted data is not changing the data, allows for "convenience and high availability", one of the most important need for our data, to be checked off.

The system as a whole allows people to share their data with whomever they want, while at the same time making use of a more powerful computer not in their control to do the calculations on the data for them.

4. **Implementation**

The implementation chosen is just a mock one. There are numerous homomorphic encryption algorithms and blockchains. For this paper, we settled on just one of each for the demonstration.

4.1. **Choice of homomorphic encryption**

As mentioned before, homomorphic encryption is still in its infancy as a cryptographic system, but some implementations with limitations exist. One of them is the FV12 [23] that, despite the name, is not a fully homomorphic encryption because the technological speed of the electronic components has not reached yet a practical level [24]. It has an implementation in multiple languages, including Python, Go, JavaScript and Ruby.

4.2. **Choice of blockchain**

The most popular blockchain [16] that allows smart contracts to be written inside it is Ethereum [17], and it uses Solidity [20] as the programming language inside it. We have chosen Ethereum because it is the blockchain that

implements the most up-to-date version Solidity language, and the team that maintains the Solidity language is made of Ethereum core contributors.

Because by its nature the blockchain system is scalable, so by deploying this onto the Ethereum blockchain, we can take full leverage of all the nodes that comprise the network.

4.3. Build and deployment

By being Turing-complete languages, we can take Python implementation of the homomorphic encryption algorithms and map them to the Solidity programming language. As a result, we can have a way to create a contract that has all the primitives a homomorphic encryption system has: the functions that can operate on the data being introduced. In this situation, the 2 operations used are multiplication and addition of ciphertext numbers encrypted with the same key separately.

The way the contract works is by taking the encrypted numbers as arguments to the function being called and having the contract itself keep the results as data. Alongside the numbers, the method takes an **ID** for an easier identification of the result. We would be using an array as a field for keeping the data on the blockchain, but we are not concerned with modifying the data, only with adding more results from the methods that are called. Therefore, we are using the transaction's log as an out-of-band data store of the contract, those are the events in Solidity. We will then send an event with the calculation done and with the address that initiated the calculation. The polling of the result is simple, just with the address or with the **ID** that was given at the method call. After getting the result back, by decrypting it with the private secret key, we can get the actual addition of the numbers sent the first time.

4.4. Functional testing

We first used a local deployed blockchain network that copies the Ethereum using the Hardhat suite and having only one node. Then, after checking that everything works as they should, we deployed to the Goerli test network. Because there were no issues, we finally published on the Ethereum main network.

After creating the contract and deploying it, the only way to test the functions is to get some random numbers **A** and **B** and then create a public key (**pk**) and a secret key (**sk**).

After encrypting the **A** and **B** numbers with the **pk**, we will get **A'** and **B'**, the encrypted versions of **A** and **B**.

We want to test that the application of the addition function on the **A'** and **B'** will yield the encrypted sum. More generally, we describe the flow of the function in Figure 1 for addition. For simplicity, we used the **'** mark to signify encrypted data and visually separate it from its clear text counterpart.

After choosing **A** to be 80 and **B** to be 100, we encrypted them to **A'** and **B'** using **pk**, we called the addition function, took the result from the

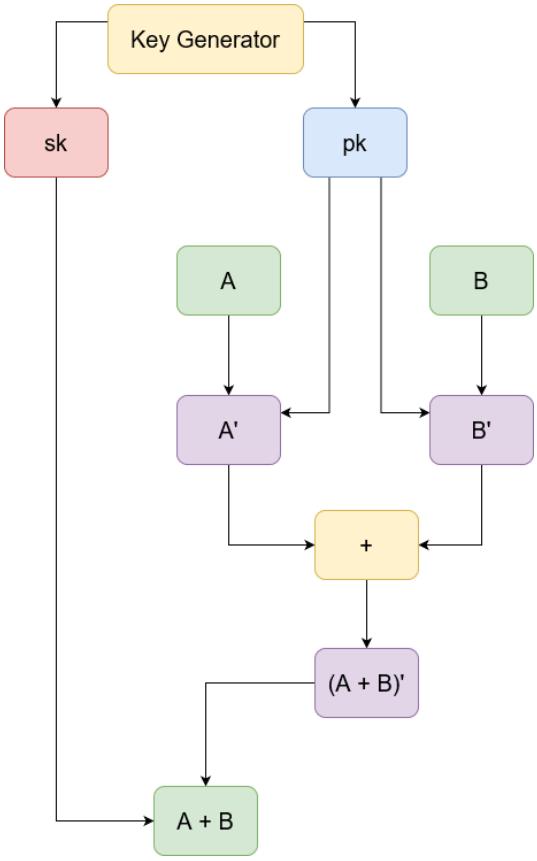

FIG. 1. General homomorphic encryption data flow

contract event, and applied offline on our computer the decryption with **sk** that never left our device. The final result is, as expected, the number 180.

Besides the functions that operate on encrypted data, there is a version of the functions that operate on one argument being clear text and the other one encrypted, both for addition and multiplication. As a result, linear combinations of 2 numbers with known coefficients can be made even if the numbers are not in plain text.

We take our initial 3 needs for data and check if they are satisfied.

- The bulk processing of the data is done on the blockchain. Even if the system that holds our data is not capable to do the computations, the only requirement is to encrypt the data that will be in transit and decrypt it at the moment we want our result back.
- Because the data does not leave our system in clear text, there is no way for any actor outside our system to gain access to our data. As a result,

the data is safe and no accidental leakage of information escapes our safe environment.

- As we mentioned before, all the computations and the storage take place on a blockchain, the Ethereum blockchain. Because the blockchain can be as a model substituted with a read-only database, we can be sure that the data that gets committed to the chain stays there and is available 24 hours a day, 7 days a week. We can always access our data, and we can easily "send" it to someone else to process it (sending is merely a symbolic operation; the "processor" takes the info directly from the blockchain without any approval from us). The malicious actors can also take our data and process it, but without a decryption key, the result is at least meaningless to use, and at most energy, time, and resource consuming, thus deterring any bad actors from even touching our data.

AND

Operand 1	Operand 2	Result
0	0	0
0	1	0
1	0	0
1	1	1

Multiplication

Operand 1	Operand 2	Result
0	0	0
0	1	0
1	0	0
1	1	1

FIG. 2. Multiplication and AND operations truth table

XOR

Operand 1	Operand 2	Result
0	0	0
0	1	1
1	0	1
1	1	0

Addition(mod 2)

Operand 1	Operand 2	Result
0	0	0
0	1	1
1	0	1
1	1	0

FIG. 3. Addition and XOR operations truth table

Figure 1 presents the flow of data using **addition** as a core homomorphic operation. We can use the usual known addition on two unencrypted numbers,

or the special homomorphic addition that allows us to operate on encrypted numbers. An identical flow exists using **multiplication** as a core operation. In Figure 2 and Figure 3, we have the truth tables for the bitwise operations of **addition**, **multiplication**, logical **XOR**, and logical **AND**. The **addition** operation used here is the one used in the homomorphic encryption algorithm schema, the one that takes a modulo out of the final result. We can observe that the truth tables are identical in both the **addition/XOR** pair and the **multiplication/AND** pair if the modulo set is 2. Because we can see that the bitwise versions of addition and multiplication are just the operation of **AND** and **XOR** in disguise, we can use them to create other logic gates.

We can create all the existing logic gates by using the **XOR** and **AND** gates and combining them for a new gate [25]. To simplify the walkthrough, we know that **NAND** and **NOR** are the acknowledged gates that we can use to create all the other existing boolean functions. We will use a **XOR** and an **AND** in Figure 4 to generate a **NAND** universal gate by using the following formula:

$$\mathbf{XOR}(\mathbf{AND}(A, B), 1) = \mathbf{NAND}(A, B)$$

Starting from the **NAND** gate, all the other gates can be created and as a result, the arbitrary computation can be executed on the data that is input into the system.

The only requirement that the circuit has is a standalone extra data that is encrypted, which represents a **1**. This can easily be pushed to the blockchain at the initial moment of sending the data, concatenating the **1** alongside it.

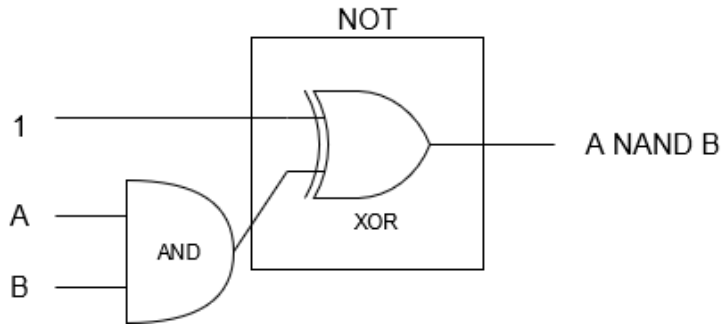

Fig. 4. Creating a **NAND** out of a **XOR** and an **AND**

5. Evaluation and caveats

Because the homomorphic encryption in our example is mostly theoretical, we can measure how this partial homomorphic encryption scheme stands up in the blockchain against repeated operations. Because of their formation,

the models introduce some noise that can override and even change the result of the final calculations.

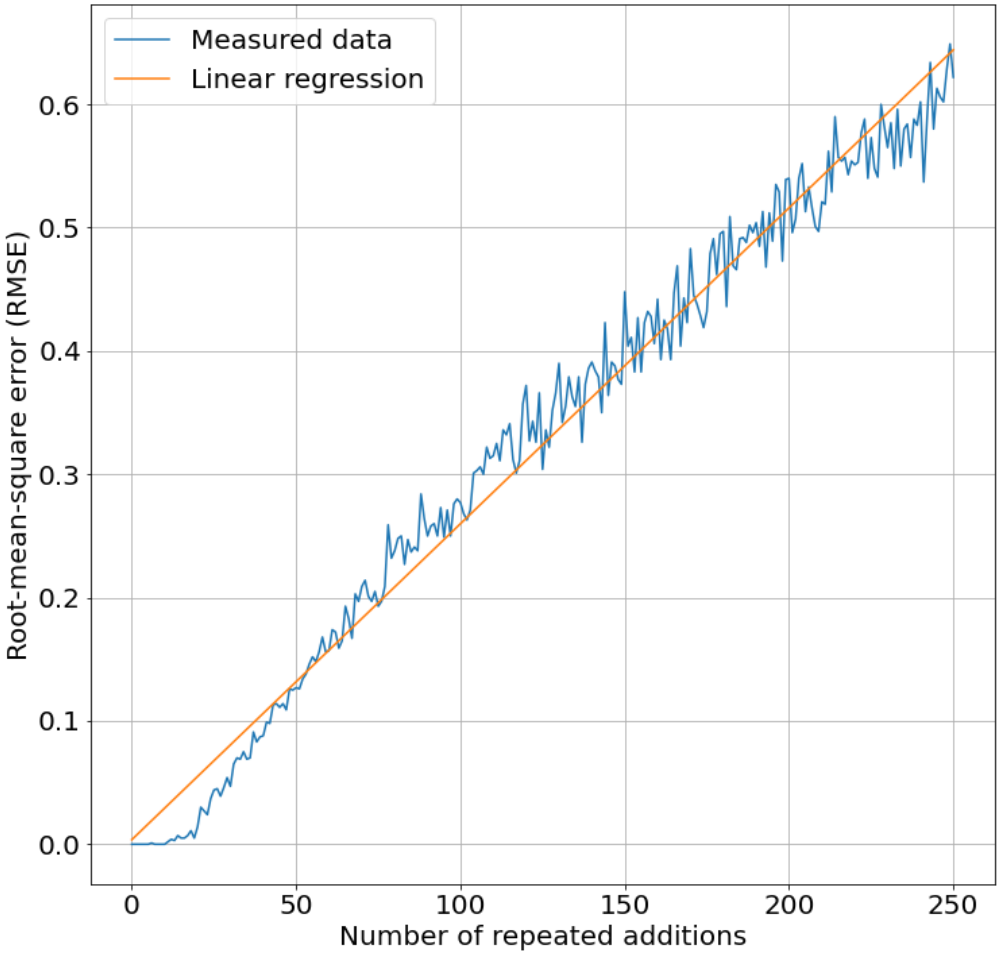

FIG. 5. Graph of the errors plotted against the number of operations

We devised a test to check how big the noise disturbance is. We can subtract the obtained value from the value that we expect to find and measure the difference. The blockchain that we used for these tests is a local clone of the Ethereum blockchain created using Hardhat, a blockchain testing suite, as the tests that we are devising are economically cost-prohibitive to be run on a

public blockchain even for a test Ethereum network, compared to just running the homomorphic encryption algorithm once.

We used the addition operation in this test. By adding 2 terms repeatedly several times (number of additions) and doing this set of operations multiple times (number of iterations), we can extract a mean error of the difference to the originally expected result.

We have tried to keep the number of additions to a maximum of 250 and the number of iterations to 10000. In Figure 5 we have plotted the number of additions relative to the error that the accumulated operations entail.

Alongside the raw measurements, we have done a linear regression on the data and plotted it in the Figure 5 so we can have a better understanding of how the error evolves with a higher number of operations. The equation that we obtained from the model is:

$$0.00256195 * X + 0.00368895 = Y$$

The equation shows that by each extra operation that we make, an extra ~ 0.0025 of error gets added to the final result.

As explained before, the experiments done are on a very early iteration of what homomorphic encryption is capable of. At the moment of experimenting, the computing power has not reached a comfortable level to be used in fully homomorphic encryption while also having the error be negligible. The computing power available also does not economically allow for more complicated operations to be done on the blockchain and so, any complicated polynomial handling still cannot be run in a real user-facing blockchain.

We presented here only addition and multiplication as the operations that allows homomorphic encryption to be used. If we consider these operations as the numerical equivalent to the logical **AND** and **XOR**, we can conclude that we can make any gate from just the addition and multiplication, performing any data manipulation that we want, not being limited to only addition and multiplication.

When the computing power will increase worldwide [26] [27] and both blockchain operations and homomorphic encryption capabilities will be easier to reach, there will be another roadblock. The storage needed on the blockchain to store the data that people create needs to be sufficiently large and adaptive to allow replication and checking between the blockchain nodes that hold the aforementioned storage linked to them.

6. **Future uses**

A potential use of homomorphic encryption using the blockchain networks would consist in secure voting systems. Creating a protected, transparent and tamper-proof voting system always comes with the need for security above all. The blockchain can provide a decentralized ledger that records votes,

ensuring that no single entity can manipulate the results. At the same time, homomorphic encryption allows votes to be encrypted, so that they can be counted without being revealed

Another use is in the healthcare data management. Hospitals and healthcare providers can use blockchain to create a secure, decentralized record of patient data. Homomorphic encryption can be used to encrypt this data, allowing it to be analyzed and used for research without compromising patient privacy.

7. Conclusions

In this paper, we managed to theorize a solution for the ever-increasing volume of data that people hold. All this data, by itself, is of little to no use, but at the same time can be sensitive enough to not allow third-party actors to see a clear text version of it. By using homomorphic encryption algorithms, we can allow external actors to act upon our data and obtain useful results, while at the same time making the result verifiable to the external checkers without leaking any clear text information from the data in question. The result is available on the blockchain and can be accessed at any time from any of the nodes that take part in the network. Regarding the computation power availability, the resources existent at the moment do not allow for a widespread and secure functional implementation of the aforementioned solution, but the constant lowering of prices for both storage and pure computation will make the execution of the model doable and practical to the masses.

REFERENCES

[1] *Krumm, J.* Ubiquitous computing fundamentals. (CRC Press,2018)

[2] *Ghosh, K., Dohan, M., Veldandi, H. & Garfield, M.* Digital transformation in healthcare: Insights on value creation. Journal Of Computer Information Systems. pp. 1-11 (2022)

[3] *Ezrachi, A. & Robertson, V.* Competition, market power and third-party tracking. World Competition. **42** (2019)

[4] *Günther, W., Mehrizi, M., Huysman, M. & Feldberg, F.* Debating big data: A literature review on realizing value from big data. The Journal Of Strategic Information Systems. **26**, 191-209 (2017)

[5] *Torre, I., Sanchez, O., Koceva, F. & Adorni, G.* Supporting users to take informed decisions on privacy settings of personal devices. Personal And Ubiquitous Computing. **22** pp. 345-364 (2018)

[6] *Xia, Qi and Tao, Zeyi and Li, Qun* Privacy issues in edge computing. Fog/Edge Computing For Security, Privacy, and Applications. pp. 147-169 (2021)

[7] *Hong, Sun-Ha* Technofutures in stasis: Smart machines, ubiquitous computing, and the future that keeps coming back. International Journal of Communication. **50** pp. 21 (2021)

[8] *Martins, P., Sousa, L. & Mariano, A.* A survey on fully homomorphic encryption: An engineering perspective. ACM Computing Surveys (CSUR). **50**, 1-33 (2017)

[9] *Acar, A., Aksu, H., Uluagac, A. & Conti, M.* A survey on homomorphic encryption schemes: Theory and implementation. ACM Computing Surveys (Csur). **51**, 1-35 (2018)

[10] *Nofer, M., Gomber, P., Hinz, O. & Schiereck, D.* Blockchain. Business & Information Systems Engineering. **59** pp. 183-187 (2017)

[11] *Belotti, M., Božić, N., Pujolle, G. & Secci, S.* A vademecum on blockchain technologies: When, which, and how. IEEE Communications Surveys & Tutorials. **21**, 3796-3838 (2019)

[12] *Ocáriz Borde, H.* An Overview of Trees in Blockchain Technology: Merkle Trees and Merkle Patricia Tries. (2022)

[13] *Schinckus, C.* Proof-of-work based blockchain technology and Anthropocene: An undermined situation?. Renewable And Sustainable Energy Reviews. **152** pp. 111682 (2021)

[14] *Saleh, F.* Blockchain without waste: Proof-of-stake. The Review Of Financial Studies. **34**, 1156-1190 (2021)

[15] *Wu, W., Liu, E., Gong, X. & Wang, R.* Blockchain based zero-knowledge proof of location in iot. ICC 2020-2020 IEEE International Conference On Communications (ICC). pp. 1-7 (2020)

[16] *König, Lukas and Korobeinikova, Yuliia and Tjoa, Simon and Kieseberg, Peter* Comparing blockchain standards and recommendations. Future Internet. **12** (2020)

[17] *Metcalfe, W. & Others* Ethereum, smart contracts, DApps. Blockchain And Crypt Currency. **77** (2020)

[18] *Feng, C. & Niu, J.* Selfish mining in ethereum. 2019 IEEE 39th International Conference On Distributed Computing Systems (ICDCS). pp. 1306-1316 (2019)

[19] *Panda, S., Mohanta, B., Satapathy, U., Jena, D., Gountia, D. & Patra, T.* Study of blockchain based decentralized consensus algorithms. TENCON 2019-2019 IEEE Region 10 Conference (TENCON). pp. 908-913 (2019)

[20] *Dannen, C.* Introducing Ethereum and solidity. (Springer,2017)

[21] *Gramlich, B.* Smart contract languages: A thorough comparison. ResearchGate Preprint. (2020)

[22] *Liang, X., Zhao, J., Shetty, S. & Li, D.* Towards data assurance and resilience in IoT using blockchain. MILCOM 2017-2017 IEEE Military Communications Conference (MILCOM). pp. 261-266 (2017)

[23] *Fan, J. & Vercauteren, F.* Somewhat practical fully homomorphic encryption. Cryptology EPrint Archive. (2012)

[24] *Yousuf, H., Lahzi, M., Salloum, S. & Shaalan, K.* Systematic review on fully homomorphic encryption scheme and its application. Recent Advances In Intelligent Systems And Smart Applications. pp. 537-551 (2020)

[25] *Drechsler, R. & Gunther, W.* Generation of optimal universal logic modules. Proceedings 25th EUROMICRO Conference. Informatics: Theory And Practice For The New Millennium. **1** pp. 80-85 (1999)

[26] *Leiserson, C., Thompson, N., Emer, J., Kuszmaul, B., Lampson, B., Sanchez, D. & Schardl, T.* There's plenty of room at the Top: What will drive computer performance after Moore's law?. Science. **368**, eaam9744 (2020)

[27] *Gargini, P.* How to successfully overcome inflection points, or long live Moore's law. Computing In Science & Engineering. **19**, 51-62 (2017)