Costs of Encrypted Computation: Why Fully homomorphic computations are Slow

Introduction

The importance of Data Privacy has increased significantly with incidents such as the Facebook-Cambridge Analytica data breach, which harvested the personal data of millions of users, leaving the credibility of democracy questionable. Even more alarming are cases where breach involving sensitive and private information like personal health information or genetic testing information like DNA can result in irreversible damage for users[1]. Data privacy breaches are exerting a lot of pressure on organizations and holding them accountable for decisions that affect user privacy.

Over the years, data privacy rules in several geographies have evolved and several countries are strictly mandating organizations to ensure data privacy regulation and enhance transparency on storage and processing of customers' data. Non-compliance to these regulations is causing huge financial and credibility implications to organizations[2]. Currently, 10% of data is covered under privacy regulations – this is expected to be 65% by 2023[3]. Also, with organizations increasingly relying on cloud service providers (CSPs), the major challenge for CSPs is to protect privacy and confidentiality of the data while still being able to cater to users' needs.

Fully homomorphic encryption (FHE), an evolving approach with mathematically provable security guarantees, enables computations on the encrypted data; thus, offering protection to the privacy of data. As privacy regulations are critical for both organizations as well as CSPs, FHE enables both of them to reduce their liabilities.

[1] Why a DNA data breach is much worse than a credit card leak: https://www.theverge.com/2018/6/6/17435166/myheritage-dna-breach-genetic-privacy-bioethics
[2] https://gdpr.eu/fines/
[3] https://www.gartner.com/smarterwithgartner/gartner-predicts-for-the-future-of-privacy-2020/

While FHE solves the problem with adequate security guarantees, it incurs some cost in terms of computational complexity and memory requirement. As remarked in our previous white paper on this subject[4], FHE based computation is a million times slower than normal computation on plaintext. We now intuitively describe several fundamental aspects that hinder performance and how the computing model in FHE differs from the normal scenario. This can help explain the limitations in a simpler manner, which in turn can help users envision new applications in FHE domain.

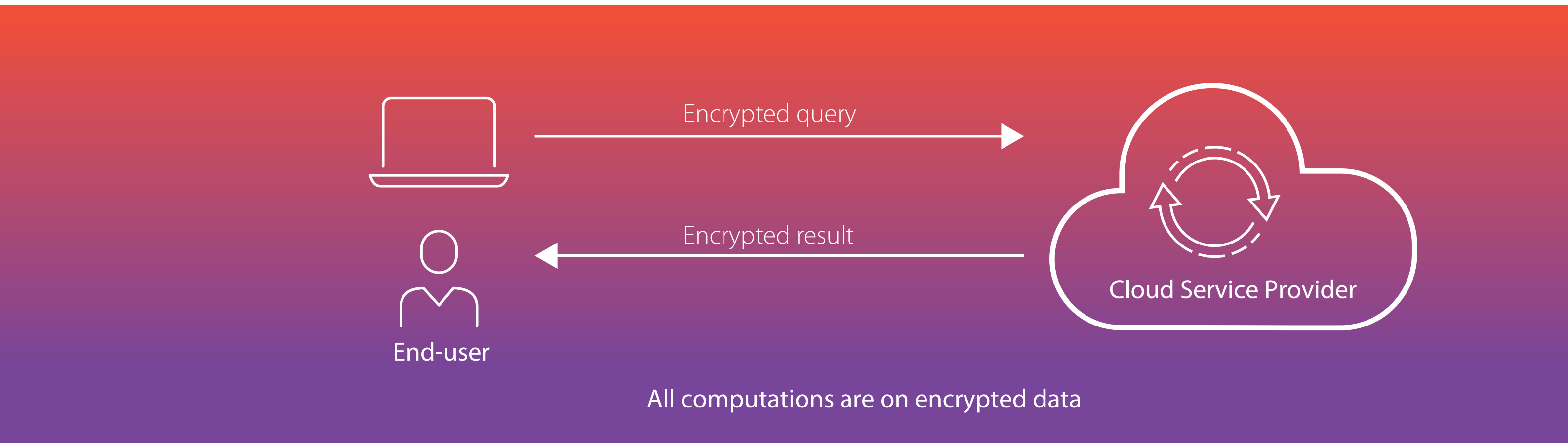

Figure 1: Operating Model for Homomorphic Encryption

Performing arbitrary computations on encrypted data was an open problem for more than 30 years. During the initial days, partial results were shown using RSA, Paillier and several other cryptosystems that could perform only limited number of additions or multiplications but not both. This hinders us to realize arbitrary functions using these cryptosystems. Craig Gentry[5] described the first ever construction of FHE which can perform arbitrary computations on the encrypted data by supporting both additions and multiplications. Later, this was improved [6, 7, 8, 9] by employing better mathematical advances that led to more efficient FHE constructions.

FHE supports addition and multiplication as primitive operations.

$$E(a)+E(b)=E(a+b) \text{ and } E(a)*E(b)=E(a*b)$$

[4] https://www.tcs.com/enabling-secure-computations-on-encrypted-data
[5] Craig Gentry.A fully homomorphic encryption scheme. Ph.d. thesis,2009.
[6] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In Proceedings of the 3rd ACM workshop on Cloud computing security workshop, pages113–124. ACM, 2011
[7] Nigel Smart and Fre Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In Public Key Cryptography-PKC2010, pages 420–443.Springer LNCS 6056,2010
[8] Vinod Vaikuntanathan. Computing blindfolded: New developments in fully homomorphic encryption. In FOCS, pages 5–16. IEEE Computer Society, 2011.
[9] Cheon, Jung Hee, et al. "Homomorphic encryption for arithmetic of approximate numbers." International Conference on the Theory and Application of Cryptology and Information Security. Springer, Cham, 2017.

These primitive operations can be used to realize arbitrary computations on encrypted data that can further be used to realize privacy preserving applications. FHE can be used in wide range of applications that have critical privacy requirements such as private search, encrypted database queries, outsourcing computations to cloud and machine learning. However, at the current state of research, it is not feasible to perform generic computations using FHE, but for certain class of problems, we can achieve performance that is usable today. In this paper, we discuss the factors affecting the performance of applications in FHE domain.

Algorithms in Encrypted Domain

An algorithm is a procedure with finite sequence of instructions that can be implemented in a computer program. These computer programs in turn are used to build an application. In the context of FHE, algorithms (or applications) are represented as circuits that can be realized using the primitive FHE operations.

Normally, a program executes instructions with full knowledge of previously computed results. This helps keep the program efficient by avoiding computations that were already performed or exiting when some condition has been satisfied. This is advantageous in the plaintext domain as it sets the execution path in the right direction and avoids unnecessary computations. However, in case of homomorphic domain, as the intermediate states are encrypted, the circuits are oblivious (blind) to the exact data, thus execution path cannot be controlled. This can be very expensive in instances where there are enormous amounts of computations that cannot be circumvented based on an intermediate state. Therefore, the machine ends up doing a whole lot of computations resulting in high usage of resources, inhibiting the efficiency of the program. We describe how these conditions are evaluated in encrypted domain and costs incurred for realizing such conditions.

Evaluating Encrypted Conditions:

Conditional statements are key decision-making elements that control the flow of execution in a program by performing different actions depending on the result of the evaluated conditional expression. A simple example of conditional execution in any programming platform is an if–then-else statement where the instructions related to the if case are executed if the conditional expression is true or the else instructions otherwise. In plaintext domain, as the intermediate states while evaluating conditions are in clear, the conditional flow of the program is transparent to the compiler. However, in homomorphic domain, all the input data and intermediate evaluation results are encrypted. Although the condition can be evaluated, the result of the evaluation is still encrypted and cannot be accessed by compiler. Thus, there is uncertainty in the decision-making process, resulting in execution of both the if case and else case, thus domains.

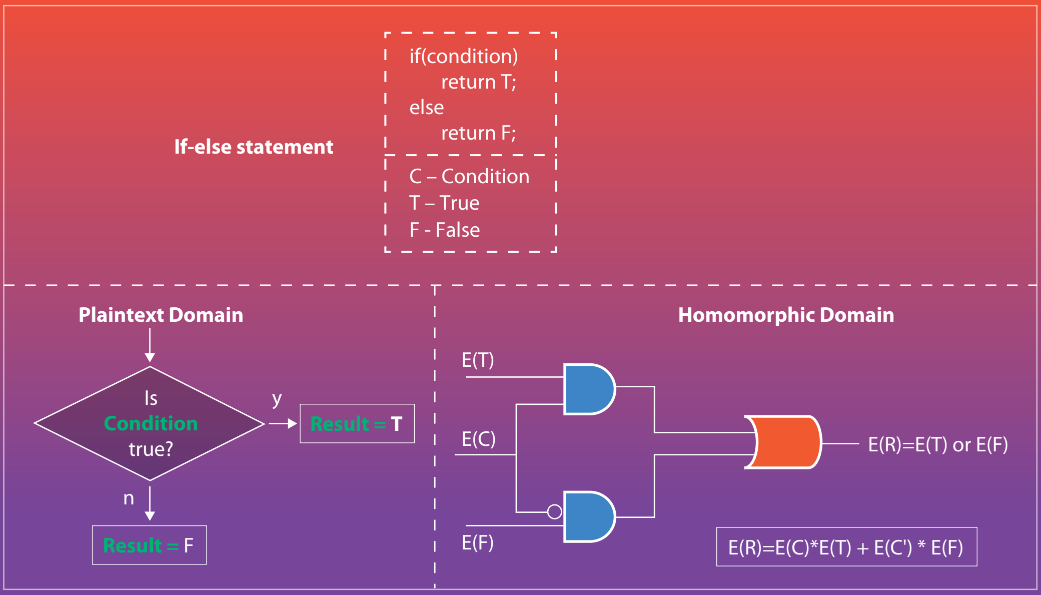

Figure 2: Conditional execution

The same issue can be observed in the case of repetitive control structures such as while or do-while loops that are used to realize more complex functionality.

Branching Problem:

Let us consider a more complex problem such as the branching problem (for instance, in a tree data structure) where the computational overhead is significantly larger in homomorphic domain when compared to the plain domain. A tree is an abstract data structure that simulates hierarchical formation of states (nodes). For simplicity, we consider a binary tree, where smaller elements are inserted to the left of a root and larger elements are inserted to the right. To traverse to a target node, we follow a path defined by the intermediate states.

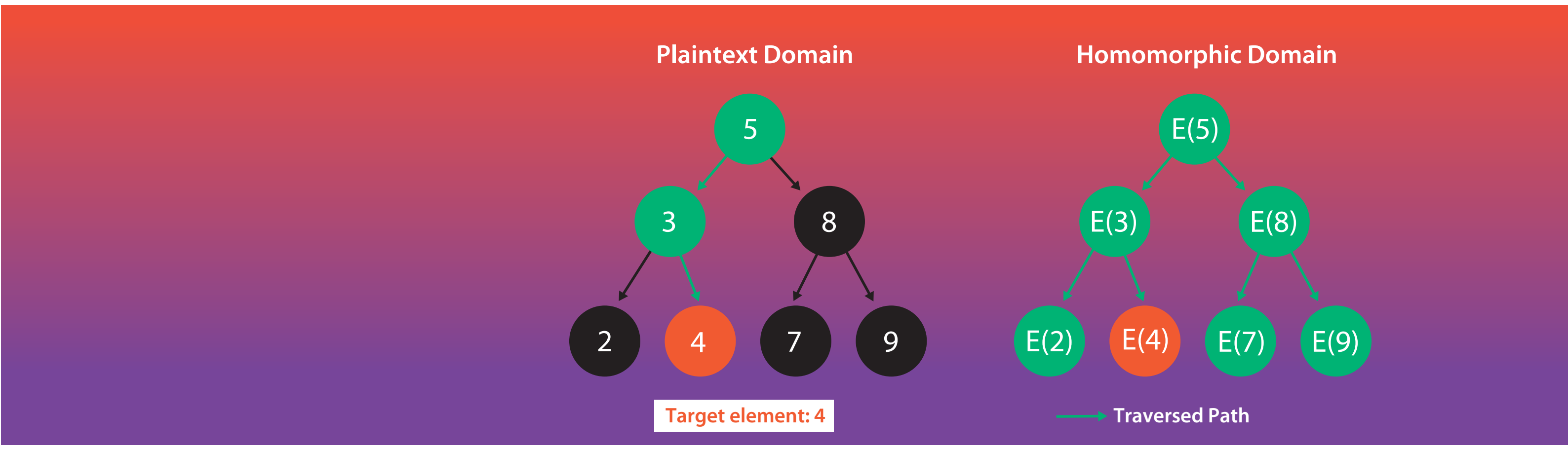

Figure 3: Traversing to a target element in a tree

For traversal in plaintext domain, only one path is followed according to the result of the conditions, whereas in homomorphic domain all possible paths must be executed and evaluated as the result of the condition is encrypted at any of the intermediate nodes (Figure 3). Therefore, in the homomorphic domain there is a significant computational overhead. Therefore, for the tree traversal, though the complexity in plain domain is O(log n), where n is size of the list, the complexity in homomorphic domain is O(n), thus yielding exponential difference between the plain and homomorphic domains.

Tree data structure is usually defined recursively, as a hierarchical structure of nodes with traversal paths defined from one node to the other. This analysis illustrates similar complexity overheads for recursion problems. In the case of recursion, the solution to a problem is based on iterating smaller instances of the same problem until a terminating condition is satisfied. In the homomorphic domain, one cannot determine the result of the terminating condition in the recursion algorithms. To mitigate this constraint, the program must be provided with some user defined thresholds for termination. Therefore, there is a limitation which inhibits the controlled execution of the program thereby increasing computational overhead.

We will now see few problems where the conditional execution in the plain and homomorphic domains exhibits significant differences in the performance of an application.

Search

Search is a well scrutinized problem in computer science. There are various search algorithms based on the mechanism of search, such as linear search, binary search and so on. For easier understanding, we consider the example of linear search understand the computation overhead involved.

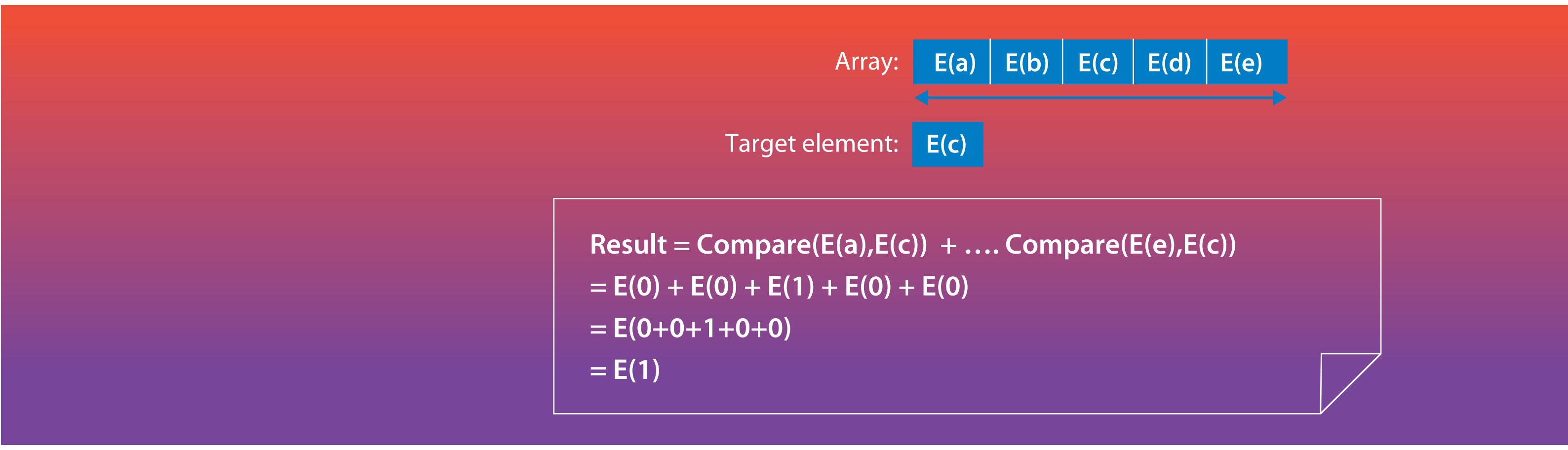
Array:
E(a)
E(b)
E(c)
E(d)
E(e)
Target element:
E(c)
Result = Compare(E(a),E(c)) + …. Compare(E(e),E(c))
= E(0) + E(0) + E(1) + E(0) + E(0)
= E(0+0+1+0+0)
= E(1)

Figure 4: Encrypted Linear Search

Linear search is an algorithm where a target element is searched in a given list by sequentially comparing the target to each element in the list. The search operation terminates when the target is found in the list. In plain domain, the linear search is straightforward and in the best case, the target element can be found at the start of the list. As soon as the element is found, the search operation can be terminated. However, in the homomorphic domain, as both the list and target element are encrypted, the program cannot determine whether an element is found at a particular position. Therefore, one needs to iterate through the entire list to find out if the target is in the list (Figure 4). The target is compared with each and every element in the list and the encrypted comparison results are aggregated homomorphically to compute the end result (Figure 2). Hence, the complexity of the computation is always O(n), which is the worst-case complexity for linear search in the plain domain. Similarly, we can generalize this overhead for other search algorithms as well that also perform with worst-case complexity in the homomorphic domain.

Sorting

Sorting is yet another important problem and many applications require frequent ordering of data to either find minimum, maximum or data in a certain range. The fundamental building blocks for any sorting algorithm are comparison and swap operations. Most known algorithms for sorting are the data dependent algorithms such as bubble sort, insertion sort, quick sort or merge sort. The complexity of these algorithms depends on the order of input. While sorting integers in the plaintext domain, the elements are swapped based on the comparison of the elements in certain indices. However, in the homomorphic domain, as the elements are encrypted, we cannot determine the comparison output, thus we do not know whether a swap operation is to be performed. Therefore, in the homomorphic domain, comparison and swap operations are combined into an oblivious multiplexer circuit to blindly operate on the encrypted data. The sorting algorithm cannot skip unnecessary swaps or comparisons, thus, increasing the complexity of the sorting (Figure 5).

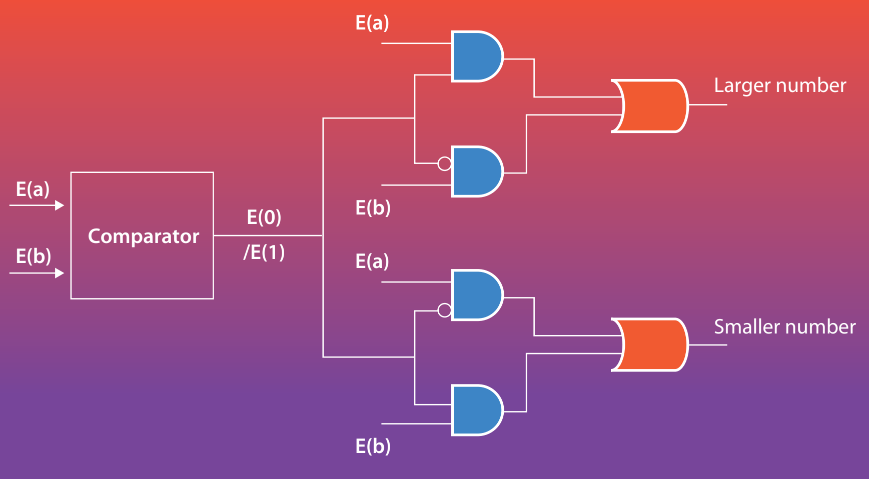

Figure 5: Comparison and Swap Circuit in FHE domain

To swap two encrypted numbers E(a) and E(b) using encrypted comparison result E(C), where E(C) ={E(0) if a>=b; E(1) else if a<b}, the compare and swap oblivious circuit can be described as follows

E(temp)= E(C) *E(a)+ E(C')*E(b);

E(b)=E(C')*E(a)+ E(C) *E(b);

E(a)=E(temp);

Note: E(C') is compliment of E(C) and can be computed from E(C) homomorphically.

Assuming that we are sorting in ascending order, if E(a) < E(b), E(a) and E(b) are homomorphically swapped. This swap operation is oblivious to the compiler, meaning the compiler does not know if the two inputs are swapped, which is very important to preserve security of the FHE schemes.

To analyze the complexity overheads induced in the FHE domain, let us consider the simplest sorting algorithm, the Bubble sort which works by making passes through the list. In each pass, adjacent elements are compared and swapped based on the comparison result; the smaller element moves to a lower index position. Since all the elements are encrypted, the algorithm is oblivious of the comparison result and ends up executing the conditional circuit described above for each pair of elements, which could have been avoided if the comparison results are known to the compiler. Therefore, it takes worst case complexity for the algorithms independent of the input. Similarly, for other data dependent sorting algorithms, average case complexity is same as the worst-case complexity in the encrypted domain[10].

[10] N. Emmadi, P. Gauravaram, H. Narumanchi and H. Syed, "Updates on Sorting of Fully Homomorphic Encrypted Data," 2015 International Conference on Cloud Computing Research and Innovation (ICCCRI), Singapore, 2015, pp. 19-24, doi: 10.1109/ICCCRI.2015.28.

In contrast to these classical sorting algorithms, there are some algorithms called sorting networks such as Bitonic sort and Odd-Even Merge sort which are data independent where the number of comparisons is constant and independent on the order of input. Hence, these algorithms have better performance in FHE domain than data dependent sorting algorithms.

The following table summarizes the complexities of sorting algorithms:

Algorithm	Plaintext domain (best case)	FHE domain (any case)
Bubble sort	O(n)	O(n^2)
Insertion sort	O(n)	O(n^2)
Quick sort	O(n log n)	O(n^2)
Merge sort	O(n log n)	O(n^2)
Bitonic sort	O(log^2 n)	O(n log^2 n)
Odd even merge sort	O(log^2 n)	O(n log^2 n)

Towards Realizing Machine Learning Applications based on FHE

In the recent years, Machine learning (ML) has revolutionized several industries such as health, finance, insurance, retail, telecom, etc. Machine Learning provides the ability for a computer to learn a task without explicit programming. This ability is imparted to the machine by training it with sample data. Once the system learns insights based on the input data, it can then make appropriate predictions on new queries. These machine learning algorithms depend on large datasets to train their models for accurate predictions. Hence, the two important components of any machine learning application can be broadly classified into Training and Inference.

The machine learning models are often deployed on cloud to cater to the needs of users such as predictions or classifications. Considering the magnitude of data involved in the machine learning applications, privacy of the data is critical, both for training as well as for inference. To address this, ML research has gained momentum towards developing privacy-preserving machine learning algorithms modelled using FHE. These types of applications are important amid growing concern of data privacy and increase in regulations such as GDPR, CCPA etc. These applications will be beneficial for both end-users as well the model owner, as they reduce data privacy related liabilities. The privacy preserving applications can employ either or both of private learning and private inference.

Privacy Preserving Learning

In the privacy preserving learning approach, the machine learning models are trained with the data encrypted with FHE. The model on the server can learn from the data without learning anything about the data. Consider the case of an enterprise that wishes to employ a third-party proprietary Machine Learning capability. The enterprise can utilize the third-party machine learning model by training with the encrypted data without revealing its data to that party, at the same time the server does not have to reveal any information about the Machine Learning algorithm to the client.

Privacy Preserving Inference

Privacy preserving inference based on FHE [10] can protect privacy of the queries in machine learning applications. An already trained model on the cloud can be queried privately by a user to obtain a prediction or classification. An example of such a case can be any health care application that can diagnose an ailment based on input health parameters. To preserve privacy of patients' health data, the model can be queried privately using FHE.

As in the case of other applications, machine learning applications also suffer from additional computational overheads in the homomorphic domain. FHE makes the computations expensive both in terms of time and memory. The most important component of machine learning applications are the cost functions. While training a machine learning model, a cost function is used to quantify error deviation from the predicted values to the expected values. In order to improve the accuracy of the model, the cost function must be minimized. This is done by evaluating the cost function iteratively and checking if the function converged to a minimum for certain input parameters. However, in the case of encrypted domain, it is not possible for the algorithm to check if the cost function converged. Hence, the algorithm is iterated up to a certain threshold and expected to converge. This may impact accuracy in certain cases if the thresholds are chosen inappropriately.

Another important component of a machine learning algorithm is an activation function in artificial neural networks (ANN). A neural network is a computing system that simulates biological neural networks of a brain. The neural networks are composed of nodes called neurons that are connected to each other. An activation function determines the output of a neuron in a neural network. These functions induce non-linearity in the neural network which is important for the network to learn complex patterns about the data and accurately predict and distinguish the data

[11] Kim, A., Song, Y., Kim, M., Lee, K., Cheon, J.H.: Logistic regression model training based on the approximate homomorphic encryption. BMC Med. Genom.11(4),83 (2018)

from other data points. In the plaintext domain, these activation functions can be computed using their corresponding high degree polynomials. However, using these polynomials as is in the FHE domain is computationally expensive. Therefore, in the FHE domain, these functions are realized using lower degree approximation polynomials that demand lower computational resources. These approximation polynomials give activation output with certain amount of loss in accuracy but with better performance.

Considering the current state of FHE schemes and the computational costs involved, it is not practical to realize arbitrary machine learning models with FHE. Certain class of applications such as neural networks that have fewer number of layers can still achieve practical performance and hence more suitable for FHE domain. The challenge lies in realizing these applications using approximation functions with minimal loss in accuracy as well as performance.

Conclusion

Enterprises are increasingly wary of the privacy risks and are inclined towards solutions that preserve privacy of the data with provable security guarantees. Although Fully Homomorphic Encryption is one of the most promising solution to mitigate these risks, it still requires significant advancements in terms of performance. In this paper, we described the fundamental aspects behind huge complexity for applications in the homomorphic domain. When compared to plaintext domain, the computations in the FHE domain are expensive due to obliviousness of the computational model. We illustrated the complexity overheads using searching and sorting techniques in the FHE domain. We also described the impact of encrypted computations in the privacy preserving machine learning applications.

Although the computational costs incurred due to encrypted computations are significant, the returns in terms of privacy guarantees are enormous. With more advances in FHE research, the performance of applications can be significantly improved. Since the first FHE scheme proposed in 2009, that is 100 trillion times slower than the plain domain, there has been significant progress in the past decade, reaching million times level currently. Though the progress is quite optimistic and there will be further improvements to the schemes, the fundamental computational model in the encrypted domain will always induce significant additional computational costs. We expect that the understanding from this paper gives insights into envisioning applications suitable for encrypted domain.

About The Authors

Harika Narumanchi

Harika Narumanchi is a researcher in the cybersecurity and privacy research area at TCS Research and Innovation (R&I). She joined TCS R&I in 2014, with research broadly focusing on applying cryptography, machine learning and blockchain solutions to business-oriented scenarios. She graduated from the Jawaharlal Nehru Technological University, Hyderabad, India, with a Master's in information technology, specialized in information security.

Nitesh Emmadi

Nitesh Emmadi is a researcher in the CyberSecurity and Privacy Research Group at TCS Innovation Labs, India. His areas of research mostly include computations on encrypted data, with a broader interest in application security, applied cryptography, machine learning and blockchains. He looks closely into the practical side of novel systems and provides consulting services to evaluate and build products for his organization. Prior to joining TCS, Nitesh received his Master's degree in Information Technology, specialized in Information Security, from International Institute of Information Technology, Hyderabad, India.

Dr Praveen Gauravaram

Dr Praveen Gauravaram is a Senior Scientist at Tata Consultancy Services (TCS) Australia leading TCS's Research & Innovation partnership with Cyber Security CRC. Praveen has a PhD in Cryptology from Queensland University of Technology and has held scientific positions in India, Europe and Australia. In 2010, Praveen was a recipient of young elite researcher award from the Danish Agency for Science, Technology and Innovation. Praveen has published more than fifty scientific and consulting articles in cryptology and cyber security. Praveen has honorary academic titles of Adjunct Associate Professor with the University of New South Wales and Adjunct Professor with Deakin University respectively.

Corporate Marketing | Design Services | M | 12 | 20