Homomorphic SVM Inference for Fraud Detection

A. Vázquez-Saavedra, G. Jiménez-Balsa, J. Loureiro-Acuña
Gradiant
Carretera de Vilar, 56, Vigo, Pontevedra, 36214, Spain
{avsaavedra, gjimenez, jloureiro}@gradiant.org

M. Fernández-Veiga, A. Pedrouzo-Ulloa
atlanTTic Research Center, Universidade de Vigo
E.E. Telecomunicación, Vigo 36310, Spain
mveiga@det.uvigo.es, apedrouzo@gts.uvigo.es

***Abstract*—Nowadays, cloud computing has become a very promising solution for almost all companies, as it offers the possibility of saving costs by outsourcing computation on-demand. However, some companies deal with private information, which must be protected before outsourcing. Banks, whose financial information is highly sensitive, are one remarkable example of this problem. Their typical processes must be run on their systems for security and regulation reasons, which impedes to take advantage of the scalability and flexibility benefits introduced by the cloud. A relevant example on which we focus in this work is the case of fraud detection systems, for which we propose the use of modern lattice-based homomorphic encryption for its secure execution. To this end, we implement and validate the performance of a homomorphic SVM (Support Vector Machine) classifier for secure fraud detection, showing the feasibility of securely outsourcing fraud detection inference.**

***Index Terms*—Support Vector Machines, Fraud Detection, Homomorphic Encryption, Lattice-based Cryptography**

Type of contribution: *Ongoing research*

I. Introduction

The impact of cloud computing on industry and also end users is difficult to estimate: many aspects of everyday life have been transformed by the omnipresence of software running on cloud networks. On the one hand, cloud computing allows companies to optimise costs and increase their offerings, without having the need of purchasing and managing all the hardware and software. This allows them to launch globally-available apps and online services without having to spend resources on the platform on which they will run. On the other hand, end users can access these applications immediately and without any specific requirements, as all these services are easily available on the Internet. Unfortunately, not all companies can benefit from the cloud computing approach.

A major disadvantage which is present in cloud-based applications is their underlying security. Actually, the use of cloud-based services always leads to the storage of information in third party systems, which could be often considered as untrusted environments. Although, in general, this is not a problem for many applications, the situation is considerably aggravated in those use cases which deal with especially privacy-sensitive information, on which security is a priority and must be maximised.

The list of related use cases encompasses companies that must handle sensitive information, such as financial and medical data, religious information, etc. This type of data is protected by law and, as companies could be exposed to different sanctions, they are forced to guarantee a certain level of protection. Even so, although conventional cryptographic techniques allow for secure storage in the cloud, data protection is not so easy if some sort of computation has to be applied on the data. Thus, in view of all these shortcomings, companies which handle sensitive data should seemingly avoid cloud-based solutions.

Precisely, the field of privacy-preserving machine learning (PPML) [1] deals with the different security and privacy threats appearing in machine learning (ML). Actually, paying attention to the scenario of cloud computing, and among the broad set of available tools inside PPML, homomorphic encryption techniques, which enable secure processing on encrypted data, seem to be a perfect fit for secure outsourcing.

A. Our Contributions

Our scenario is set inside a banking fraud context [2], on which banks make use of fraud detection systems that require a large amount of resources to operate. Bank transactions do not follow a uniform time distribution, so a good solution for banks is to outsource their fraud detection systems, which allows them to take advantage of the flexibility and scalability provided by the cloud.

Our main contribution is to *implement and showcase the feasibility of a solution based on homomorphic cryptography* (see Section II), which *enables to securely outsource the fraud detection systems*. In particular, we have implemented a SVM (Support Vector Machine) inference algorithm in the encrypted domain (see Sections III and IV); being this classifier tailored for bank fraud detection. *Our homomorphic SVM classifier allows a bank to make secure encrypted predictions in an untrusted environment*.

In order to test its feasibility: (1) we have designed a proof of concept based on a client-server model (see Section IV) and, (2) we have evaluated the runtime and classification performance of the system (see Section V).

Notation and Structure: We represent vectors and matrices by boldface lowercase and uppercase letters, respectively. Polynomials are denoted with regular lowercase letters, ignoring the polynomial variable (e.g., a instead of $a(z)$). Finally, the Hadamard product of two vectors is $\boldsymbol{a} \circ \boldsymbol{b}$.

The rest of the paper is organized as follows: Section II briefly reviews the used homomorphic encryption scheme. Section III details the bank fraud detection scenario, the dataset and the SVM classifier. Section IV introduces the design of our system and the homomorphic classifier. Finally,

This work was funded by the Ayudas Cervera para Centros Tecnológicos, grant of the Spanish Centre for the Development of Industrial Technology (CDTI) under the project EGIDA (CER-20191012). Also funded by the Agencia Estatal de Investigación (Spain) and the European Regional Development Fund (ERDF) under project RODIN (PID2019-105717RB-C21), and by the Xunta de Galicia and ERDF under project ED431G2019/08.

TABLE I
HIGH LEVEL DESCRIPTION OF THE CKKS SCHEME [9]

Parameters: Let $R_q[z]$ be the quotient polynomial ring $\mathbb{Z}_q[z]/(1+z^n)$. Then, ciphertexts from the $\mathsf{E}_{\mathsf{CKKS}}$ scheme are composed of two polynomial elements from $R_q[z]$, and plaintexts belong to the set $\mathbb{C}^P$ such that $P = n/2$. Finally, q and n (for simplicity we omit some internal parameters) are chosen in terms of the security parameter λ.	
$\mathsf{E}_{\mathsf{CKKS}}$.SecKeyGen	**Input**: Security parameter λ **Output**: Secret key sk
$\mathsf{E}_{\mathsf{CKKS}}$.PubKeyGen	**Input**: Secret key sk **Output**: Public key pk, evaluation key evk and rotation key rtk
$\mathsf{E}_{\mathsf{CKKS}}$.Enc	**Input**: Plaintext $\boldsymbol{m} \in \mathbb{C}^P$ **Output**: A ciphertext $c = (c_0, c_1) \in R_q^2[z]$
$\mathsf{E}_{\mathsf{CKKS}}$.Dec	**Input**: A ciphertext $c' = (c'_0, c'_1) \in R_q^2[z]$ encrypting a plaintext $\boldsymbol{m}' \in \mathbb{C}^P$ **Output**: A plaintext $\boldsymbol{m}' \in \mathbb{C}^P$
$\mathsf{E}_{\mathsf{CKKS}}$.Add	**Input**: Ciphertexts c and c' encrypting, respectively, $\boldsymbol{m}$ and $\boldsymbol{m}'$ **Output**: A ciphertext $c'' = (c''_0, c''_1) \in R_q^2[z]$ encrypting $\boldsymbol{m} + \boldsymbol{m}' = \boldsymbol{m}'' \in \mathbb{C}^P$
$\mathsf{E}_{\mathsf{CKKS}}$.LinHadMult	**Input**: A plaintext $\boldsymbol{m} \in \mathbb{C}^P$ and a ciphertext c' encrypting $\boldsymbol{m}'$ **Output**: A ciphertext $c'' = (c''_0, c''_1) \in R_q^2[z]$ encrypting $\boldsymbol{m} \circ \boldsymbol{m}' = \boldsymbol{m}'' \in \mathbb{C}^P$
$\mathsf{E}_{\mathsf{CKKS}}$.HadMult	**Input**: Evaluation key evk, and ciphertexts c and c' encrypting, respectively, $\boldsymbol{m}$ and $\boldsymbol{m}'$ **Output**: A ciphertext $c'' = (c''_0, c''_1) \in R_q^2[z]$ encrypting $\boldsymbol{m} \circ \boldsymbol{m}' = \boldsymbol{m}'' \in \mathbb{C}^P$
$\mathsf{E}_{\mathsf{CKKS}}$.Rot	**Input**: A ciphertext c encrypting $\boldsymbol{m} \in \mathbb{C}^P$ **Output**: A ciphertext $c' = (c'_0, c'_1) \in R_q^2[z]$ encrypting $\boldsymbol{m}' \in \mathbb{C}^P$, which is the result of applying a rotation over the components of $\boldsymbol{m}$

Section V evaluates the performance of our system, and Section VI discusses some future work lines.

II. PRELIMINARIES: HOMOMORPHIC ENCRYPTION

Homomorphic encryption appears as a very promising tool for secure processing [3]. One relevant example is the Paillier cryptosystem [4], which has been used in a broad range of different applications [5]. It presents a group homomorphism which enables additions between two encrypted values, and multiplications between a ciphertext and a plaintext.

Consequently, Paillier could be a perfect candidate to implement our proposed encrypted detector for bank fraud detection. However, modern lattice-based cryptosystems outperform Paillier in several aspects: (1) *more complex applications are possible* as they present a ring homomorphism which enables multiplication between two ciphertexts (e.g., the training of a SVM [6]), and (2) *better performance*, as Paillier is slower even when considering only linear operations [7], [8].

In view of the above points, we make use of the CKKS scheme [9], which is a lattice-based cryptosystem especially adapted to work with approximate arithmetic operations.

A. A concrete homomorphic encryption scheme

We give a brief description of the CKKS scheme $\mathsf{E}_{\mathsf{CKKS}}$ in Table I. It is defined as a conventional public-key encryption scheme $\mathsf{E} = \{\mathsf{SecKeyGen}, \mathsf{PubKeyGen}, \mathsf{Enc}, \mathsf{Dec}\}$, but extended with Add, HadMult, LinHadMult and Rot procedures. For further details of the scheme we refer the reader to [9].

III. USE CASE

Banks use fraud detection systems that require a large amount of computational resources. This results in significant costs for banks to keep their systems up and running. One possible approach for banks is to outsource these systems to the cloud where, in addition to saving costs, they would obtain more flexible systems.

However, banking information is sensitive and traditional techniques only allow to work with data in clear, what can be a security problem. Consequently, we propose to implement the evaluation phase of an important machine learning algorithm such as Support Vector Machines (SVM) in the encrypted domain. Therefore, by directly working with encrypted data, banks could outsource this information securely and take advantage of the benefits provided by the cloud.

A. Dataset

The dataset used for this paper is called IEEE-CIS Fraud Detection [10]. It is composed of real-world e-commerce transactions provided by Vesta, a leader payment service company. The dataset is divided into 4 files train_transaction.csv, train_identity.csv, test_transaction.csv and test_identity.csv. However, we only use the files train_transaction.csv and train_identity.csv, as the other two are unlabeled and we could not use the samples they contain to test the results of the inference. Consequently, these two training files are combined, finally obtaining a dataset with 590540 rows and 434 features.

B. Dataset pre-processing

Given the large dimensionality of the training dataset, an adequate pre-processing is required before training. As our main objective in this work focuses on implementing the SVM inference function in the encrypted domain, we have followed the guidelines for data pre-processing and feature selection proposed in [10]. According to their recommendations:

- The number of features is reduced from 434 to 115.
- The columns with missing values are filled with the mean, mode or the most frequent category among the present values in the corresponding column.

C. SVM linear kernel

We have chosen a Support Vector Machine (SVM) classifier due to its good performance for the fraud detection scenario [10]. SVMs correspond to a type of binary classifiers which, given a set of training samples, maximize the gap between the two considered groups (e.g., 1 fraud vs -1 non-fraud in our scenario).

Specifically, in this work we are interested in the expression of the SVM classifier:

$$\mathsf{sign}\left(\underbrace{\sum_{i=1}^{L} y_i \alpha_i \mathsf{K}(\boldsymbol{x}_i, \boldsymbol{x}) + b}_{\mathsf{score}}\right), \tag{1}$$

where $\boldsymbol{x}$ is the input sample, and the rest of parameters are obtained through the SVM training: b is the bias, α_i are the dual coefficients, and $\boldsymbol{x}_i$ and y_i are, respectively, the support vectors together with their associated label (1 or -1). We refer to [11] for more details on the training of SVMs.

However, due to the limitations of homomorphic encryption schemes, two additional points must be considered:

- The $\mathsf{sign}(\cdot)$ function can be very costly to be homomorphically computed. As it does not introduce a relevant overhead, we have moved it to the client side, which calculates it after decryption.
- Among the possible choices for the kernel $\mathsf{K}(\cdot,\cdot)$ function, we make use of a linear kernel $\mathsf{K}(\boldsymbol{u}, \boldsymbol{v}) = \boldsymbol{u} \cdot \boldsymbol{v}$; mainly due to its simplicity and good behaviour for the fraud detection problem [10]. Additionally, it brings about an important efficiency advantage thanks to the fact that the inner product is distributive over vector addition, which enables to considerably simplify the classifier.

Consequently, taking into account the above changes, our homomorphic classifier turns out to be:

$$\mathsf{score} = \underbrace{\left(\sum_i y_i \alpha_i \boldsymbol{x}_i\right)}_{\text{linear SVM model}} \cdot \underbrace{\boldsymbol{x}}_{\text{input sample}} + b. \qquad (2)$$

For more details on its homomorphic computation and the different packing methods, we refer the reader to Section IV-B.

IV. SVM Homomorphic Inference

So as to showcase the potential and practicability of homomorphic encryption for PPML, and more specifically, for the encrypted execution of SVM inference, we have implemented a client-server prototype. Starting from the use case, bank fraud detection with a SVM, the needed homomorphic operations for both encryption/decryption and for the encrypted inference using a linear kernel have been implemented.

A. Design

The design of the prototype follows a client-server approach to emulate the client and cloud side for an scenario of outsourced computation in an untrusted environment. The client is responsible of encryption and decryption, so the input data is protected from the moment it leaves the client. The server is responsible of processing the encrypted data, and of finally sending back to the client the encrypted result. Figure 1 depicts the main components of the secure SVM prototype.

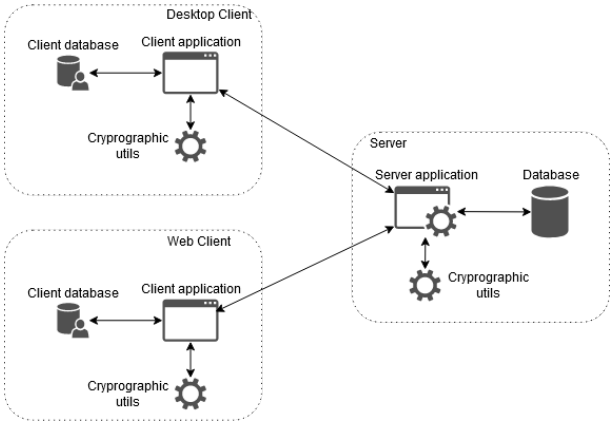

Fig. 1. System architecture

As shown in Figure 1, the prototype is composed of two different clients (one is web based and the other one command line based) and one server. All 3 components make use of a database and of a cryptographic library that implements the needed cryptographic primitives for each component. The clients database stores the user homomorphic cryptographic keys and the transactions data samples that are classified with the Homomorphic SVM Inference. The server database stores users authentication details, users public homomorphic cryptographic keys and parameters, and the SVM model. The clients cryptographic library implements the generation of cryptographic keys and parameters, and also the data packing/unpacking and encryption/decryption primitives, while the server cryptographic library implements the encrypted kernel evaluation primitives. In order to implement the primitives for encrypted processing, the cryptographic libraries make use of Microsoft Seal [12] and Lattigo [13] libraries. For running the homomorphic inference, the following steps must be followed:

1) The *Client* authenticates to the server
2) The *Client* creates a key pair and sends its public part to the server
3) The *Client* packs and encrypts the data samples that wants to infer and sends them to the server
4) The *Server* retrieves the SVM model and public keys from database
5) The *Server* does the Homomorphic SVM Inference and sends the result to the client
6) The *Client* decrypts the result and shows it to the user

B. Data packing

As the CKKS scheme [9] allows for homomorphic approximate additions and Hadamard products between encrypted vectors (see Section II), finding the best way of packing the data before encryption is fundamental to optimize the performance of our encrypted SVM classifier.

With this aim, we have explored two different packing methods [2], which we denote in this work as *column packing* and *flattened packing* respectively. Let $\boldsymbol{X}$ be a matrix of size $N \times M$ where N represents the number of input samples we want to query for detection, and M corresponds to the number of features for each input sample. We also assume that the instantiated CKKS scheme has a packing capacity of P complex slots. We briefly describe these two methods next:

- Column packing: It encodes the columns of $\boldsymbol{X}$ separately in different ciphertexts. This packing naturally fits into the SIMD (Single Instruction, Multiple Data) paradigm, which makes it convenient for those cases where the number of input samples N in the client query is large.
- Flattened packing: We represent the matrix $\boldsymbol{X}$ as a reshape on which all the rows are concatenated into a "flattened" vector of length $N \cdot M$ such as $[\mathsf{row}_1(\boldsymbol{X}), \ldots, \mathsf{row}_N(\boldsymbol{X})]$. Then, we encrypt all the row_i blocks using the minimal possible number of ciphertexts, i.e., each ciphertext has $\lfloor \frac{P}{M} \rfloor$ different rows from $\boldsymbol{X}$.

Contrarily to the former, the use of a *flattened packing* optimizes the storage in those cases on which N is small in relation to the ciphertext packing capacity P.

Homomorphic inference: Regarding the concrete algorithms in the encrypted domain, there are also some important differences for each packing:

- Inference with column packing: It follows the same structure as its counterpart in clear. We homomorphically compute Eq. (2) by means of both $\mathsf{E}_{\mathsf{CKKS}}$.LinHadMult and $\mathsf{E}_{\mathsf{CKKS}}$.Add primitives. It is worth noting that, as a different score is calculated by default in parallel for P complex slots, we directly obtain the inference for P different input samples in each output ciphertext.
- Inference with flattened packing: It requires a more cumbersome algorithm on which not only $\mathsf{E}_{\mathsf{CKKS}}$.LinHadMult and $\mathsf{E}_{\mathsf{CKKS}}$.Add primitives are needed, but also $\mathsf{E}_{\mathsf{CKKS}}$.Rot must be applied to relocate the partial results of the inner product. In this case each output ciphertext contains $\lfloor \frac{P}{M} \rfloor$ different score values.

Tables II and III include a summary of the existing trade-offs between both packing methods. Note that the flattened

TABLE II
COMPUTATIONAL COST FOR EACH PACKING METHOD

Method	Ciphertext operations
Column packing	$\lceil \frac{N}{P} \rceil (M \text{ mult.} + M \text{ add.})$
Flattened packing	$\lceil \frac{N}{\lfloor \frac{P}{M} \rfloor} \rceil (1 \text{ mult.} + \log_2 \lceil M \rceil \text{ add.} + \log_2 \lceil M \rceil \text{ rot.})$

TABLE III
STORAGE REQUIREMENTS FOR EACH PACKING METHOD

Method	# of ciphertexts
Column packing	$M \lceil \frac{N}{P} \rceil$
Flattened packing	$\lceil \frac{N}{\lfloor \frac{P}{M} \rfloor} \rceil$

packing also requires to generate the rotation key matrices rtk which are used for the homomorphic slot rotations.

V. IMPLEMENTATION RESULTS

In this section we show the results obtained in our implementation. All the experiments were conducted using a testbed composed of a physical device with a CPU i7-4710HQ, 12GB of RAM and Ubuntu Desktop 20.04.

In the first place, we evaluate the quality of the inference of our implementation against the inference in the clear. These results are shown in Table IV. In order to test the classification performance, we have considered three different metrics:

- Accuracy: % of correct predictions.
- Precision: % of correctly detected positives among all detected positives.
- Recall: % of correctly detected positives among all positives.

TABLE IV
INFERENCE RESULTS: MACHINE LEARNING CONTEXT

Library	Packing	Accuracy	Precision	Recall
Plain	-	96.6%	5.44%	46.8%
SEAL	Flattened	96.4%	4.39%	32.5%
SEAL	Column	96.6%	3.72%	70.4%
Lattigo	Flattened	96.6%	3.72%	70.4%
Lattigo	Column	96.6%	3.72%	70.4%

TABLE V
INFERENCE RESULTS: PERFORMANCE CONTEXT
($N = 147635, M = 114, P = 4096$)

Library	Packing	$\mathcal{P}_\mathsf{S}$ time (ms)	$\mathcal{P}_\mathsf{C}$ time (ms)	Size (KB)
SEAL	Flattened	4.03	3.66	31.3
SEAL	Column	0.12	1.69	14.0
Lattigo	Flattened	2.34	16.8	41.9
Lattigo	Column	0.03	8.29	18.7

Another important aspect that we have assessed is the runtime performance. In this case, Table V includes:

- Server ($\mathcal{P}_\mathsf{S}$) runtime: the required time to perform the inference function in the server.
- Client ($\mathcal{P}_\mathsf{C}$) runtime: total time taken by the client to encode/decode + encrypt/decrypt the different samples.
- Size: size of the information sent from client to server.

Note that our results consider the execution of several inferences in parallel, so *the included runtimes are normalized to measure the estimated runtime for a single inference of each type of packing*. As the transmission times have not been taken into account, we include *the size of transmitted information per inference*.

VI. CONCLUSIONS AND FUTURE WORK

This work shows that homomorphic encryption is a key enabler technology for migrating highly sensitive process to the cloud. Our proof of concept prototype demonstrates the feasibility of running the fraud detection inference in the untrusted domain, while maintaining the prediction performance and also with a feasible execution performance. As future work, we envision several extensions for our prototype:

- A more complete comparison considering other alternative cryptographic schemes/techniques as Paillier [4], BFV [14], and functional encryption [15]
- Deploy the server in a real cloud environment in order to measure the instantiation costs in a realistic scenario.
- Protect the model in the execution environment. This corresponds to the case on which the server infrastructure provider and the model owner are not the same entity.
- Extend the prototype considering more complex kernels; e.g., polynomial and RBF (Radial Basis Function).
- The implementation in the untrusted domain of the $\mathsf{sign}(\cdot)$ function. Currently, there exist mechanisms for its homomorphic approximation with CKKS [16]. Note this adds an additional degree of protection to the model, as it is harder for the client reverse engineering the model.

REFERENCES

[1] M. Al-Rubaie and J. M. Chang, "Privacy-preserving machine learning: Threats and solutions," *IEEE Secur. Priv.*, vol. 17, no. 2, pp. 49–58, 2019.

[2] A. Vázquez-Saavedra, "Study and applications of homomorphic encryption algorithms to privacy preserving svm inference for a bank fraud detection context," Master's thesis, Telecommunication Engineering School, University of Vigo, 2021. [Online]. Available: http://castor.det.uvigo.es:8080/xmlui/handle/123456789/533

[3] Homomorphic encryption standardization. [Online]. Available: https://homomorphicencryption.org/

[4] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *EUROCRYPT*, ser. LNCS, vol. 1592. Springer, 1999, pp. 223–238.

[5] R. L. Lagendijk, Z. Erkin, and M. Barni, "Encrypted signal processing for privacy protection: Conveying the utility of homomorphic encryption and multiparty computation," *IEEE Signal Process. Mag.*, vol. 30, no. 1, pp. 82–105, 2013.

[6] S. Park, J. Byun, J. Lee, J. H. Cheon, and J. Lee, "He-friendly algorithm for privacy-preserving SVM training," *IEEE Access*, vol. 8, pp. 57 414–57 425, 2020.

[7] A. Pedrouzo-Ulloa, J. R. Troncoso-Pastoriza, and F. Pérez-González, "Number theoretic transforms for secure signal processing," *IEEE Trans. Inf. Forensics Secur.*, vol. 12, no. 5, pp. 1125–1140, 2017.

[8] J. R. Troncoso-Pastoriza, A. Pedrouzo-Ulloa, and F. Pérez-González, "Secure genomic susceptibility testing based on lattice encryption," in *IEEE ICASSP*. IEEE, 2017, pp. 2067–2071.

[9] J. H. Cheon, A. Kim, M. Kim, and Y. S. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *ASIACRYPT*, ser. LNCS, vol. 10624. Springer, 2017, pp. 409–437.

[10] Z. Wu. (2020) Kaggle — IEEE Fraud Detection (EDA). [Online]. Available: https://blog.csdn.net/Tinky2013/article/details/104215953

[11] I. Steinwart and A. Christmann, "Support vector machines," in *Information science and statistics*, 2008.

[12] "Microsoft SEAL (release 3.0)," http://sealcrypto.org, Oct. 2018, microsoft Research, Redmond, WA.

[13] C. Mouchet, J.-P. Bossuat, J. Troncoso-Pastoriza, and J. Hubaux, "Lattigo: a multiparty homomorphic encryption library in go," 2020.

[14] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *IACR Cryptol. ePrint Arch.*, vol. 2012, p. 144, 2012.

[15] T. Marc, M. Stopar, J. Hartman, M. Bizjak, and J. Modic, "Privacy-enhanced machine learning with functional encryption," in *ESORICS*, ser. LNCS, vol. 11735. Springer, 2019, pp. 3–21.

[16] E. Lee, J. Lee, J. No, and Y. Kim, "Minimax approximation of sign function by composite polynomial for homomorphic comparison," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 834, 2020.