Journal of King Saud University – Computer and Information Sciences 35 (2023) 101718

HOSTED BY

Contents lists available at ScienceDirect

Journal of King Saud University – Computer and Information Sciences

journal homepage: www.sciencedirect.com

Highly efficient secure linear algebra for private machine learning classifications over malicious clients in the post-quantum world

Artrim Kjamilji [a,*], Osman Berke Güney [b,c]

[a] Faculty of Engineering, Istanbul Commerce University, Istanbul, Turkey
[b] Faculty of Engineering and Natural Sciences, Sabancı University, Istanbul, Turkey
[c] Orta Mahalle, Üniversite Caddesi No:27 Tuzla, 34956 Istanbul, Turkey

ARTICLE INFO

Article history:
Received 7 March 2023
Revised 24 July 2023
Accepted 17 August 2023
Available online 25 August 2023

Keywords:
Novel secure linear algebra
Privacy preserving algorithms
Machine learning classification
Malicious clients
Post-quantum cryptography

ABSTRACT

Over the last decade there has a been widespread usage of Machine Learning (ML) classifiers in cases such accurate disease diagnosis at clinics, credit card fraud detection in banks, cyber-attacks prevention of computer systems in different industries, etc. However, privacy and security concerns and law regulations have been an obstacle to the usage of ML classifiers. To this end, this paper addresses the scenario where a server has a private trained ML model, and one or more clients have private queries that they wish to classify using the server's model. During the process, the server learns nothing, while the clients learn only their final classifications and nothing else. Several ML classification algorithms, such as Deep Neural Networks, Support Vector Machines, Logistic Regression, different flavors of Naïve Bayes, etc., can be expressed in terms of linear algebra operations. To this end, initially, as building blocks, several novel secure linear algebra operations are proposed. On top of them novel secure ML classification algorithms are proposed for the aforementioned classifiers under strict security, privacy and efficiency constraints and their security is proven under the semi-honest model. Since the used underlying cryptographic primitives are shown to be resilient to quantum computer attacks, the proposed algorithms are also suitable for the post-quantum world. Furthermore, the proposed algorithms are non-interactive and, based on where the bulk of the operations are done, they have the flexibility to be server or client centric. Theoretical analysis and extensive experimental evaluations over benchmark datasets show that the proposed secure linear algebra operations, hence the secure ML algorithms build on top of them, outperform the state-of-the-art schemes in terms of computation and communication costs as well as on security and privacy characteristics. Moreover, and to the best of the authors' knowledge, for the first time in literature the security of the proposed algorithms is proven when dealing with multiple malicious clients during classifications.

© 2023 The Author(s). Published by Elsevier B.V. on behalf of King Saud University. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

1. Introduction and problem statement

Machine Learning (ML) classification (inference) has seen a wide range of applications in many areas of the modern life. Namely, different hospitals/clinics own already trained ML models and use/offer them to their patients/clients for accurate diagnosis of many diseases (Kjamilji et al., 2021b; Liu et al., 2017; Kjamilji et al., 2020; Chen et al., 2019). Or, cyber-security companies own already trained ML models which are used by their customers/-clients to detect or prevent cyber-attacks in customers' network and computer systems (Shaukat et al., 2020). Similarly, banks also own ML trained models and offer them to their customers/clients to be used for credit card fraud detection, fraudulent transactions (Leo et al., 2019), etc. However, security and privacy concerns from both the ML model owner and the client's side, as well as law regulations, have been an obstacle in the widespread usage of ML classifications. From the perspective of the ML model owner, the trained ML model should be kept private since it is a source of revenue for the clinic/bank/company. Besides, revealing the trained ML model can leak some information about the private dataset(s) over which the ML model was trained. From the perspective of the client, the query which is send to the ML model for classification (labeling) purposes should be kept private since it contains sensitive information such private patients health records and symptoms, credit card information, log data of computer and network systems, etc.

* Corresponding author at: Küçükyalı, İnönü Cd. No:4, 34854, 34854 Maltepe/Istanbul, Turkey.
E-mail addresses: akjamilji@ticaret.edu.tr (A. Kjamilji), osmanberke@sabanciuniv.edu (O.B. Güney).

https://doi.org/10.1016/j.jksuci.2023.101718
1319-1578/© 2023 The Author(s). Published by Elsevier B.V. on behalf of King Saud University.
This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

The first to address the issue of data privacy while dealing with Machine Learning (ML) and Data Mining algorithms where (Agrawal and Srikant, 2000) and (Lindell and Pinkas, 2000). Several other schemes soon followed up. A survey of them is given in (Aggarwal and Philip, 2008). However, most of these early schemes exclusively concentrated on the privacy of the datasets, with little or no privacy concerns about the classification (inference) stage. Those fall into the **Privacy Preserving (PP) ML training** category. Realizing these shortcomings, starting from a few years ago, several schemes were proposed that deal with the case where a server owns a trained model, and one or more client(s)/user(s) has/have an unclassified query that he/they wish(es) to classify using the trained model. In those schemes, besides several others, usually the trained model's privacy and/or query privacy and/or the final query classification are the most common privacy goals. Such schemes fall into the **Privacy Preserving (PP) ML classification** category. Over the years several such schemes were proposed that deal with different PP ML classifiers such as Naive Bayes (NB) (Kjamilji et al., 2021b; Bost et al., 2015; Gao et al., 2018; Li et al., 2018; Liu et al., 2015; Brakerski et al., 2014; Fan and Vercauteren, 2012; Smart and Vercauteren, 2014; Liu et al., 2017; Sun et al., 2018; Park et al., 2018; Kjamilji et al., 2020; Pereira, 2020; Gupta et al., 2017; Kjamilji et al., 2021a; Kjamilji et al., 2021c; Vu et al., 2022; Yang et al., 2005; Hao et al., 2014; Vu et al., 2020; Resende et al., 2022; Vu, 2022; Zhao and Xia, 2023; Duy, 2022; Chen et al., 2021); Multinomial Naive Bayes (MNB) (Kjamilji et al., 2021b; Gupta et al., 2017; Kjamilji, 2021; Kjamilji et al., 2021c; Liu et al., 2018; Khedr et al., 2015; Costantino et al., 2017); Support Vector Machines (SVM) (Kjamilji et al., 2021b; Bost et al., 2015; Gao et al., 2018; Li et al., 2018; Sun et al., 2018; De Cock et al., 2017; Liu et al., 2018; Rahulamathavan et al., 2013a; Rahulamathavan et al., 2013b; Smahi et al., 2020; Liang et al., 2021; Liang et al., 2020; Zhu et al., 2016; Zhang et al., 2020; Farokhi, 2020); Logistic Regression (LR) (Kjamilji et al, 2021; Gupta et al., 2017; De Cock et al., 2017; Guo et al., 2018; Zhou et al., 2023; Fan et al., 2020; Edemacu and Kim, 2021; Cheon et al., 2018; Ghavamipour et al., 2022); Deep Neural Networks (DNN) (Juvekar et al., 2018; Jiang et al., 2018), etc. However, almost all of them have one or more drawback(s), such as a total or partial leakage of the trained ML model, of the client query and/or of the final label of the query, or they use computationally costly cryptographic techniques, or they introduce a high number of communication rounds which increases the communication cost, etc. Another drawback that should be emphasized is the fact that most of the state-of-the-art PP ML classification schemes use cryptographic techniques that can be broken by quantum computers, which are around the corner, thus they are not applicable to the post-quantum world. A more detailed elaboration and comparisons of the related state-of-the-art PP ML classifiers is given in Section 2.

Paper's contributions: To these ends, having in mind the drawbacks of the state-of-the-art schemes elaborated in Section 2 and knowing that, among others, all the aforementioned ML classifiers can be expressed in terms of linear algebra operations:

1. **A set of novel secure linear algebra** operations **are proposed**, such as **secure dot (inner) product**, different flavors of **secure matrix–vector product**, a few algorithms of **secure matrix–matrix product**, different flavors of **secure matrix transpose**, etc.
2. On top of the proposed novel secure linear algebra algorithms, for the afore-mentioned ML classifiers, **novel ML classifiers are built under strict security, privacy as well as computation and communication efficiency requirements**, which are elaborated in Section 2 (Table 1).
3. **Theoretical analysis and extensive experimental evaluations show that the proposed set of novel secure linear algebra**, hence the **PP ML classifiers** build on top of them as well, **outperform the related state-of-the-art ones in terms of computation and communication costs**, **as well as on security and privacy properties** (characteristics).
4. To the best of our knowledge, this paper is **the only one to deal with multiple malicious clients** during PP ML classifications.
5. **All the proposed algorithms are resistant to quantum computers attacks; thus** are **applicable to the post-quantum world**.

2. Related research and state-of-the-art

Related to this paper are considered the state-of-the-art schemes which exclusively deal with PP ML classifications of classifiers that can be expresses in terms of linear algebra (especially matrix) operations, such as NB, MNB, SVM, LR and DNN.

Among the first schemes in literature to deal with PP NB is (Yi and Zhang, 2009), which is has a few shortcomings. One of them is that in (Yi and Zhang, 2009) they partially retrain the ML NB model for each client query, instead of doing it once and using it in the subsequent queries. Also, in (Yi and Zhang, 2009) they use a couple of servers (called mixers) which are assumed to be non-colluding and the underlying scheme is not resistant to attacks from quantum computers. Furthermore, (Yi and Zhang, 2009) reveals the final classification result, does not allow simultaneous multi-query classifications, and does not deal with malicious users. A few years ago, Bost et al. (2015) proposed a few secure building blocks on top of which they build their PP ML classifiers. However, schemes in (Bost et al., 2015) suffer from high communication costs due to the high number of interaction rounds, which are due to the highly interactive secure building blocks. Besides, they don't allow multiple queries at once, have multiple pairs of public keys based on different computationally expensive cryptosystems, are vulnerable to the Substitute-Than-Comparison (STC) attack introduced in (Gao et al., 2018), etc. While in (Gao et al., 2018) and (Li et al., 2018) they fix the STC attack issues from (Bost et al., 2015), yet both (Gao et al., 2018) and (Li et al., 2018), among other drawbacks, mainly inherit most of the drawback in (Bost et al., 2015) since their schemes are built on top of the same secure building blocks proposed in (Bost et al., 2015). The scheme in (Liu et al., 2015), among other setbacks, suffers from using several public key pairs of cryptosystems which are computationally expensive, and their schemes require high number of communication rounds. Besides, among others, in (Liu et al., 2015) they utilize an extra server which is assumed to be non-colluding, do not support multiple queries, reveal some intermediate results, etc.

The following papers are built on crypto schemes which are based on the hardness of Decision-RLWE (Ring Learning With Errors) given in Section 9, which in turned is assumed to be safe from quantum computer attacks (Brakerski et al., 2014; Fan and Vercauteren, 2012; Smart and Vercauteren, 2014). While in (Liu et al., 2017) they introduce a single round non-interactive protocol, yet, among others, the proposed schemes suffer from high computation costs due to proxy re-encryptions and other issues of the underlying cryptographic scheme. Also, (Liu et al., 2017) does not keep the trained model fully private, and partially leaks the final classification result.

Sun et al. (2018) does not offer client query privacy, leaks some of the intermediate results, the proposed secure building blocks are not proven to be secure and is interactive. In Park et al. (2018) they remove some of the drawback introduced in (Sun et al., 2018), yet (Park et al., 2018) suffers from high computation costs, doesn't deal with multiple queries or malicious users, and is not protected from

Table 1
Properties and privacy characteristics among different secure and private ML classification schemes.

Properties	Schemes																						
	(Bost, 2015)	(Gao et al., 2018)	(Li et al., 2018)	(Liu et al., 2015)	(Liu et al., 2017)	(Sun et al., 2018)	(Park et al., 2018)	(Kjamilji et al., 2020)	(Pereira, 2020)	(Gupta et al., 2017)	(Juvekar et al., 2018)	(De Cock et al., 2017)	(Liu et al., 2018)	(Rahulamathavan, 2013a)	(Vu et al., 2022)	(Yi and Zhang, 2009)	(Guo et al., 2018)	(Rahulamathavan, et al. 2013b)	(Liang et al., 2020)	(Zhao and Xia, 2023)	(Chen et al., 2021)	(Zhou et al., 2023)	(Zhang et al., 2020)	Thispaper
ML model used for PP classifications	SVM, NB, DT	NB	SVM, NB	NB	NB	SVM, NB, DT	NB	NB	NB	(M)NB, SVM, LR	DNN	SVM, LR	SVM	SVM	NB	NB	LR	SVM	SVM	NB	NB	LR	SVM	DNN, LR, SVM, (M) NB
Total trained model privacy	✓	✓	✓	✓	✓×	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓×	✓	✓	✓	✓	✓	✓	✓	✓
Total query privacy	✓	✓	✓	✓	✓	×	✓	✓	✓	✓	✓	✓×	✓	✓	✓	✓	✓	✓	✓	✓×	✓	✓	✓	✓
Final classification privacy	✓	✓	✓	✓	×	✓×	✓	✓	✓×	✓	✓	✓×	✓	✓	✓	×	×	✓	✓	✓	✓	✓	✓	✓
Intermediate results privacy	✓	✓	✓	×	×	✓×	✓	✓	✓	✓	✓	✓	×	✓	✓	✓×	×	✓	×	✓×	✓	✓×	✓	✓
Non-interactive (exactly one-round of communic.)	×	×	×	×	✓	×	✓	✓	✓	×	✓	×	✓	×	×	✓	✓×	×	✓	✓	✓	✓×	✓	✓
Simultaneous multi-query classification	×	×	×	×	×	✓	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	✓
Avoids one or more non-colluding semi-honest servers	✓	✓	×	×	×	✓	✓	✓	✓	✓	✓	✓	✓	✓	×	×	×	✓	✓	×	✓	×	✓	✓
Resistant to quantum computer attacks	×	×	×	×	✓	✓	✓	✓	✓	✓	✓	×	×	×	×	×	×	×	✓	✓	✓	×	×	✓
No loss of accuracy due to PP classification	✓	✓	✓	NR	×	NR	✓	✓	NR	NR	NR	NR	NR	NR	NR	NR	×	✓	✓	✓	×	NR	NR	✓
Deals with PP multi-class classification	✓	×	✓	✓	×	✓	✓	✓	✓	✓	✓	✓	×	✓	✓	✓	×	×	×	✓	✓	×	✓	✓
Avoids resource demanding schemes (FHE, re-encryption, garbled circuits, expon., etc.)	×	×	×	×	×	✓	×	×	✓	×	✓	✓×	×	✓	✓×	✓	×	×	✓	✓×	×	✓	×	✓
Deals with malicious clients/users	×	×	×	×	×	×	×	✓	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	✓
Flexibility in terms of where the bulk of operations are done (server vs client centric)	×	×	×	×	×	×	✓	✓	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	✓
Avoids Retraining the trained model for each query(ies)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	×	✓	✓	✓	✓	✓	✓	✓	✓
Resistant to eavesdropping	✓	✓	✓	✓	✓	✓	✓	✓	✓×	✓	✓	×	✓	✓	✓	✓	✓×	✓	✓	×	✓	✓	✓	✓
Resistant to the Substitute Then Comparison (STC) attack (Gao et al., 2018)	×	✓	×	×	×	×	×	✓	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	×	NA	×	✓
Avoids having multiple pair of keys or access token (permission) from the same or different public key protocols	×	✓	×	×	×	✓	✓	✓	✓	✓	✓	✓	×	✓	×	×	×	✓	×	✓	✓	×	×	✓
Resistant to collusion attacks between two or up to q − 1 out of q clients	NA	NA	NA	NA	NA	✓	NA	✓	NA	NA	NA	×	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	✓
Follows the parameter standards of the cryptographic schemes on top of which the PP classifications are done	✓	✓	✓	✓	✓	×	✓	✓	✓	✓×	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Legend: ✓ - presence; × - absence; ✓× - partial presence of the property/characteristic; **NR** – **N**ot **R**eported; **NA** = **N**ot **A**pplicable.
DNN = **D**eep **N**eural **N**etworks; **SVM** = **S**upport **V**ector **M**achines; **DT** = **D**ecision **T**ree; **LR** = Logistic regression; **NB** = Naïve Bayes; **MNB** = **M**ultinomial **N**aïve **B**ayes; **(M)NB** = both NB and MNB.

the STC-attack. The scheme in (Kjamilji et al., 2020) has 2 communication rounds and uses computationally costly cryptosystems, while the scheme in (Pereira, 2020) does not fully protect the privacy of the final classification result. Both (Kjamilji et al., 2020) and (Pereira, 2020) do not deal with multi-query classification. While the scheme in (Gupta et al., 2017) has good privacy characteristics, its main drawback is the costly computation cost due to the usage of expensive garbled circuits, which become costlier as the number of classes in multi-class classification becomes large.

More recent schemes, such as (Juvekar et al., 2018; De Cock et al., 2017; Liu et al., 2018; Rahulamathavan et al., 2013a; Vu et al., 2022; Guo et al., 2018; Rahulamathavan et al., 2013b) and (Liang et al., 2020), in general, don't deal with simultaneous multi-query classifications which would increase the throughput and the performances of the scheme, nor do they offer the flexibility on where the bulk of the operations will be done (server or client side), neither do they deal with malicious clients that don't follow the protocol. Some of them, such as (De Cock et al., 2017; Liu et al., 2018; Rahulamathavan et al., 2013a; Vu et al., 2022; Yi and Zhang, 2009; Guo et al., 2018) and (Rahulamathavan et al., 2013b), are prone to quantum computer attacks, while (Li et al., 2018; Liu et al., 2015; Liu et al., 2017; Vu et al., 2022; Yi and Zhang, 2009) and (Guo et al., 2018) use one or more extra server (s) besides the server that owns the trained mode and the client (s). Schemes in (Gao et al., 2018; Liu et al., 2017; Liu et al., 2018; Guo et al., 2018) and (Rahulamathavan et al., 2013b) do not deal with multi-class classifications and their performances suffer from resource demanding schemes such as FHE (Fully Homomorphic Encryption, (Brakerski et al., 2014; Fan and Vercauteren, 2012; Smart and Vercauteren, 2014) with bootstrapping, re-encryption, garbled circuit, expensive exponentiations, etc., to name a few. Furthermore, a few of the mentioned schemes do not hide some of the intermediate results which can be used as a trapdoor to retrieve the trained model fully or partially. Moreover, not all those few schemes that deal with multi-client query classifications enable the non-colliding clients to preserve the query privacy or get the accurate classification result due to the malicious or semi-honest colluding clients.

More recent PP ML classification schemes, such as those in (Zhao and Xia, 2023; Chen et al., 2021; Zhou et al., 2023; Fan et al., 2020; Cheon et al., 2018; Zhang et al., 2020), do not deal with malicious clients, do not deal with multiple query classification which would increases the performances and (depending on where the bulk of the computationally costlier operations are done) do not offer the flexibility of server or client-based classifications. Furthermore, (Chen et al., 2021; Fan et al., 2020) and (Cheon et al., 2018) report loss of classification accuracy due to dealing with PP ML classifications, whereas (Duy, 2022; Chen et al., 2021; Fan et al., 2020) and (Zhang et al., 2020) use computationally costly cryptographic primitives or multiple public key-pairs in their system. Moreover, schemes in (Vu, 2022; Zhou et al., 2023; Edemacu and Kim, 2021) and (Zhang et al., 2020) are vulnerable to quantum computer attacks; schemes in (Zhao and Xia, 2023; Duy, 2022) and (Ghavamipour et al., 2022) use secret sharing which are vulnerable to eavesdroppers that can listen to all of the communication channels; schemes in (Chen et al., 2021) and (Zhang et al., 2020) are susceptible to the STC-attack in (Gao et al., 2018); while (Zhao and Xia, 2023) and (Zhou et al., 2023) need two non-colluding semi-honest cloud servers to execute their PP ML classification.

To conclude, as summarized in Table 1 the majority (if not all) of the state-of-the-art schemes have one or more drawbacks which are not acceptable in the post-quantum world. To these ends, this paper aims to address and eliminate all of them by having as implementation goals the strict security, privacy and efficiency characteristics elaborated in Table 1.

3. Partıcıpants' model and algorıthm flows

Participant and their adversary model. Schemes in this have two types of participants, server, and client. The server owns a trained ML model, whereas one or more client(s) have unclassified query(ies) that they wish to classify using the server's trained model having in mind the goals set in Table 1 and elaborated in Section 2. Both the server and the client(s) are in the semi-honest (honest-but-curious mode), which is technically and formally introduced in Section 9. Briefly and informally, participant belonging to the semi-honest model follow the protocol, but, while executing it, they try to infer some information and intermediate results, which they are not supposed to.

Moreover, a randomly chosen ***Main-client*** which does the bulk of the aggregation operations for all the clients, and a random ***Decr-Client*** which has the decryption key for the server centric classifications (Section 6) are introduced. When dealing with malicious clients (Section 8) all the clients have malicious tendencies and might collude, except for the *Main-Client* and the server, which always remain in the semi-honest model and don't collude with each other or with any of the clients.

Secure ML classification – client centric algorithm flow at-a-glance (Fig. 1): the trained model *M*resides at the client encrypted by the server's public key. All encryptions are done using the server's public key, which also has the decryption key. The client(s) construct(s) his/their queries $X^{(i)}$❶, encode and (depending on the circumstances, might also) encrypt and send them to the *Main-Client* which packs all the queries in a single plaintext (S_p) or single ciphertext (S_c) ❷. For each feature, the client query contains the client's feature values upon which the query will be classified (labeled) in secure and private fashion. E.g. for the case of disease prediction in clinics, the client query contains the patient's symptoms upon which an accurate disease diagnosis will be done in secure and private manner. More details on the query content and construction for different ML classifiers is given in Section 4.1. Using the encrypted trained model *M,* the *Main-Client* does the secure ML classifications over the encrypted trained model to get the classification results $C_M(S)_c$ in encrypted form ❸. After randomizing the result, the *Main-client* gets the encrypted $rndC_M(S)_c$, sends it to the server and send the randomization parameters to the other clients ❹. After decrypting and decoding it, the server obtains the randomized classification results in plain $rndC_M(S)_v$, and sends to each client its randomized classification result ❺. Finally, each client de-randomizes $rndC_M(S)_v$ to get the final classification for the query(ies), $C_M(S)_v$ ❻.

Secure ML classification – server centric algorithm flow at-a-glance (Fig. 2): the trained model *M*resides at the server in plain (un-encrypted). All encryptions are done using *Decr-client*'s public key, which also has the decryption key. The client(s) construct(s) his/their queries $X^{(i)}$❼, encode and (depending on the circumstances) encrypt and send them to the *Main-Client,* which packs all the queries in a single ciphertext (S_c) which is send to the server ❽. The server does the secure ML classifications over the encrypted trained model to get the classification results $C_M(S_c)$❾, randomizes them to get $rndC_M(S)_c$, sends it to the *Decr-Client* and sends the randomization parameters to all the clients ❿. The *Decr-Client* decrypts the ciphertext send by the server to get $rndC_M(S)_v$ in plain and sends to each client its portion of

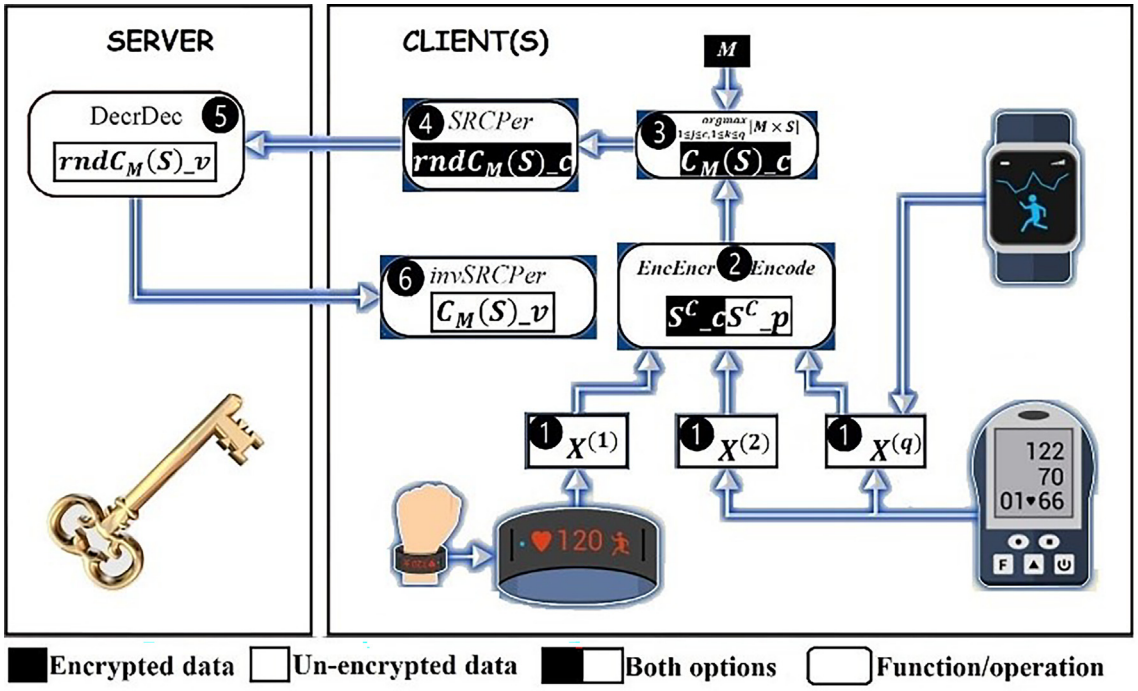

Fig. 1. Secure ML client centric classification algorithm flow.

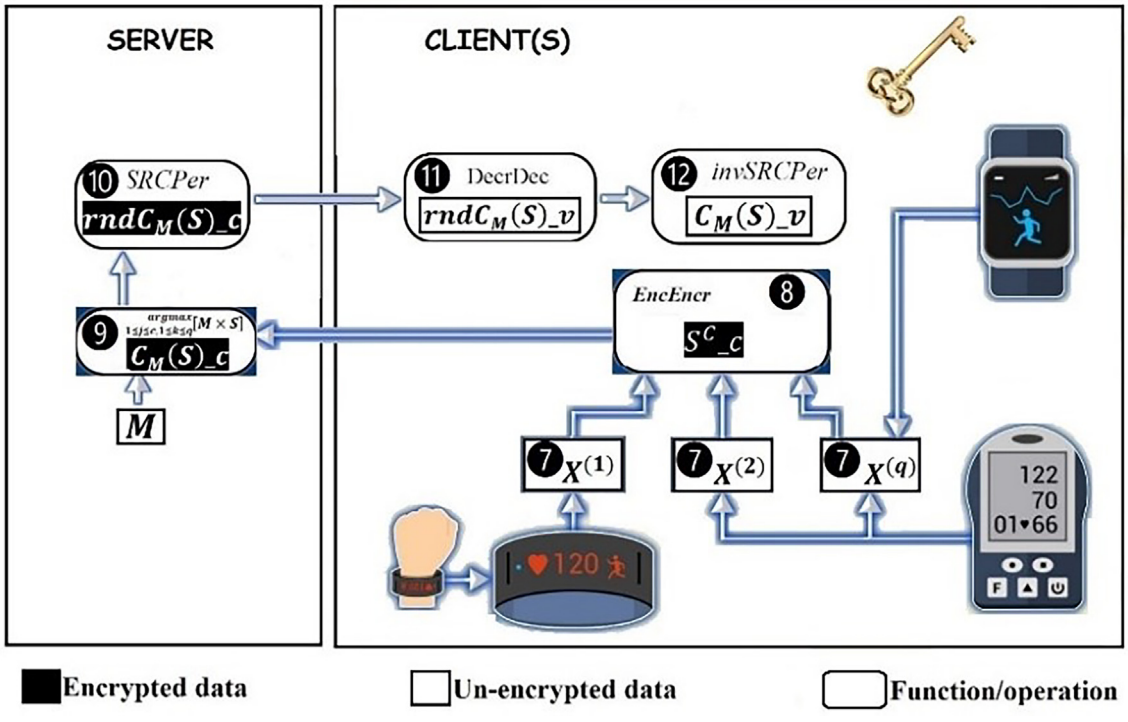

Fig. 2. Secure ML server centric classification algorithm flow.

the randomized classification result ⓫. Clients de-randomize this to get the final classification result $C_M(S)_v$ ⓬.

The detailed flow of both algorithms is given in Section 6.

4. Preleminaries

This section briefly introduces some of the ML classification algorithms which can be expressed in terms of matrix operations (NB,

MNB, SVM, LR and DNN) over already trained ML models. This sections also introduces the cryptographic techniques which will be used to build the proposed PP ML classifications in Section 6. The notations introduced here will be valid throughput the rest of the paper.

4.1. ML multi-class classifications

Definition 1. Classifying an unclassified query (record) X, denoted as $C_M(X)$, according to a trained model M obtained by a certain training algorithm $\mathscr{T}$ means assigning a label Λ to X according to the corresponding classification algorithm (function) $\mathscr{C}$ that as an input takes X and M and output the corresponding label, thus $C_M(X) = f(M,X) = \Lambda$.

4.1.1. Naïve Bayes (NB)

Let's have n sets of discrete features denoted as $F_i = \{V_{1,F_i}, V_{2,F_i}, \ldots, V_{|F_i|,F_i}\}$, where $1 \le i \le n$. Here $|F_i|$ is the number of elements (cardinality) of feature set F_i and V_{m,F_i} is the m-th value the same feature set. If the features are not discrete, they can be discretized (Bost et al., 2015; Gao et al., 2018; Li et al., 2018; Liu et al., 2015; Liu et al., 2017; Sun et al., 2018; Park et al., 2018; Kjamilji et al., 2020; Pereira, 2020; Gupta et al., 2017) . Let's have a set of c classes, denoted as $C = \{C_1, C_2, \ldots, C_c\}$. Let $P(C_j)$ and $P(V_{m,F_i}|C_j)$ be the class probability and conditional feature-value class probability, respectfully, for $1 \le j \le c$, $1 \le i \le n$, and $1 \le m \le |F_i|$. Those probabilities represent the trained NB model. An un-classified query vector $X\prime = \{X\prime_1, X\prime_2, \ldots, X\prime_n\}$, where $X\prime_i \in F_i$, according to the NB classifier is classified by (1):

$$C_M(X') = \underset{1 \le j \le c}{argmax} \left[K \cdot logP(C_j) + \sum_{i=1}^{n} K \cdot logP(X'_i|C_j) \right] \quad (1)$$

i.e. by labeling the query with the class which has the highest (maximum) posterior probability $P(C_j|X') \approx logP(C_j) + \sum_{i=1}^{n} logP(X\prime_i|C_j)$ (Bost et al., 2015; Gao et al., 2018; Li et al., 2018; Liu et al., 2015; Liu et al., 2017; Sun et al., 2018; Park et al., 2018; Kjamilji et al., 2020; Pereira, 2020; Gupta et al., 2017). In (1) K is a constant, whereas $\lfloor \cdot \rfloor$ is the floor function. The constant K and the floor function are used to convert decimal number probabilities into integers, which are needed for the encryption purpose of the scheme given in Section 4.2. K is usually a 8–10 bits constant, which is enough to avoid the loss of classification accuracy when dealing with PP ML classification algorithms that work over encrypted integers (Park et al., 2018; Kjamilji et al., 2020; Kjamilji et al., 2021a).

Let f be the sum of the cardinalities of all the features sets, i.e., $f = \sum_{i=1}^{n} |F_i|$. Let's define an $f+1$ dimensional row-vector belonging to class C_j as

$$C^{(j)} = \left\{ K\log P(C_j), \underset{i=1}{\overset{f}{,}} \left(\overset{|F_i|}{\underset{m=1}{,}} K\log P(V_{m,F_i}|C_j), \right) \right\}$$
$$= \{K\log P(C_j), K\log(V_{1,F_1}|C_j), K\log(V_{2,F_1}|C_j), \ldots, K\log(V_{F_n,F_n}|C_j)\},$$

for $1 \le i \le f, 1 \le m \le |F_i|$, $1 \le j \le c$. Hence, as its first element $C^{(j)}$ has class C_j's probability proceeded by all the feature values of all the feature sets in order. Let the rows of a $c \times (f+1)$ matrix M consist of the row vectors of all the c classes in order, i.e., $M = \left[\left\{ C^{(j)} \right\}_{j=1}^{c} \right]_{c \times (f+1)}$. M, basically, represents the trained model matrix. Similarly, let's redefine the query $X\prime$ as a $f+1$ dimensional column-vector

$$X = \left\{ 1, \underset{i=1}{\overset{f}{,}} \left(\overset{|F_i|}{\underset{m=1}{,}} V_{m,F_i} \right) \right\} = \{1, V_{1,F_1}, V_{2,F_1}, \ldots, V_{|F_n|,F_n}\}$$
$$= \{1, X_1, \ldots X_f\}.$$

Hence, X in its first index has 1 proceeded by a sequence of 0 s and 1 s, s.t. if the corresponding index belonging to a certain feature-value appears in the original query $X\prime$, then its value is 1, otherwise all the remining indexes have values of 0.

Then Eq. (1), i.e., the classification of column-vector query X according to the trained NB model matrix M, in terms of matrix operations, can be expressed as it is shown in (2):

$$C_M(X) = \underset{1 \le j \le c}{argmax} [M \times X] \quad (2)$$

The results of the $M \times X$ multiplication is a c-dimensional column-vector which in its indexes, in order, stores the posteriors probabilities of query X to belong to the corresponding class. The maximum value of those indexes corresponds to the term $\underset{1 \le j \le c}{argmax}$ in (2), which is the class with the highest posterior probability and the label for query X.

4.1.2. Multinomial Naïve Bayes (MNB)

Let's have a set of f selected features (bag-of-words) denoted as $W = \{w_1, w_2, \ldots, w_f\}$ and a set of c classes deonted as $C = \{C_1, C_2, \ldots, C_c\}$. Let the class probabilities be $P(C_j)$, while the conditional feature-value class probabilities be $P(w_i|C_j)$, where $1 \le i \le f$, $1 \le j \le c$. Let's define the rows of the $c \times (f+1)$ dimensional training model matrix as follows $C^{(j)} = \{P(C_j), P(w_1|C_j), P(w_2|C_j), \ldots, P(w_f|C_j)\}$. Then, similarly as with NB, the trained model matrix is denoted as $M = \left[\left\{ C^{(j)} \right\}_{j=1}^{c} \right]_{c \times (f+1)}$. Let $X = \left\{ f_{w_1}, f_{w_2}, \ldots, f_{w_f} \right\}$ be the query where f_{w_i} is the frequecy of appereance of word w_i in the query X, s.t. $1 \le i \le f$. Then the classification of X according to M, thus $C_M(X)$, is done using Eq. (2).

4.1.3. Support Vector Machines (SVM)

Let's have f sets of features, denoted as $F_1, \ldots, F_f$, and c classes forming the set of classes $C = \{C_1, C_2, \ldots, C_c\}$. Following the one-vs-all reasoning, each class C_j has an $f+1$ dimensional decision hyperplane, denoted as $W^{(j)} = \left\{ b^{(j)}, w_1^{(j)}, \ldots, w_f^{(j)} \right\}$, which divides the instances labeled to belong to that class with the ones that do not belong to the class. Here $b^{(j)}$ is the bias and $w_i^{(j)}$ is the i-th weight of of the C_j-th class, for $1 \le i \le f, 1 \le j \le c$. Let's stack the hyperplanes in order to form a $c \times (f+1)$ matrix expressed as $M = \left[\left\{ W^{(j)} \right\}_{j=1}^{c} \right]_{c \times (f+1)}$, which is basically the trained SVM model. Let the query X be an $(f+1)$ dimensional column vector, expressed as $X = \{1, X_1, \ldots, X_f\}$, where $X_i \in F_i$, $1 \le i \le f$. Similarly to the NB case, query Xcan be classified according to M by using Eq. (2) (Bost et al., 2015; Gao et al., 2018; Li et al., 2018; Sun et al., 2018; De Cock et al., 2017; Liu et al., 2018; Rahulamathavan et al., 2013a; Vu et al., 2022).

4.1.4. Logistic regression (LR) for binary classes

Let's have $c = 2$ classes (class c_1 and class c_2) and $f$features, denoted as $F_1, \ldots, F_f$. The trained model M is an $f+1$ dimensional row vector, whereas the query vector is an $f+1$ dimensional column-vector denoted as $X = \{1, X_1, \ldots, X_f\}$ where $X_i \in F_i$ *for* $1 \le i \le f$. The final classification of query X is calculated using Eq. (2), which in this case is a mere dot (inner) product of M and X, and the result is an integer. If this result is positive than the query is labeled as belonging to class c_1, otherwise it is labeled as class c_2.

Improvement 1: for the binary class case ($c = 2$), the trained model M for NB, MNB and SVM has 2 rows. They can be merged into a single row by subtracting index-wise the first and the second row. This converts the classification of query X according to M into a simple dot (inner) product s.t. if $C_M(X) > 0$ then X is labeled to belong to class c_1, otherwise it is labeled as c_2.

4.1.5. Deep Neural Networks (DNN)

A processing unit (neuron, perceptron) takes an input and on this input it performs a so called non-linear activation function (sigmoid, leaky ReLU, tanh, squaring, etc.) to get the corresponding output. A layer of neurons has a certain number of neurons, usually with the same actvation function. Let's have $L+1$ layers of neurons, where Layer 0 is the input layer which has $d_0 = f$ neurons and the remaining L layers are the hidden ones and the ouput one. Each of them has a dimension (number) of d_l neurons per layer and a corresponding layer activation function $f^l(\cdot)$, where $1 \le l \le L$ (Fig. 3). Let the output of neuron i of layer $l-1$ be connected with neuron j of its neighbouring layer l through a link with weight w_{ij}, where i and j are the dimensions of layer $l-1$ and layer l, respectfully, thus $1 \le i \le d_{l-1}$ and $1 \le j \le d_l$. Let the weights that go from neuron i of layer $l-1$ to all the neurons j of the neighbourng layer l form a row vector denoted as $W_i^l = \left\{w_{i1}^l, w_{i2}^l, \ldots, w_{id_l}^l\right\}$. In this way the weights' matrix $M^l = \left[\left\{W_i^l\right\}_{i=1}^{d_{l-1}}\right]_{d_{l-1}\times d_l}$ is defined. The trained model M of a DNN consists of the set of all weights' matrices $M = \left\{M^l\right\}_{l=1}^{L}$ (Fig. 3). Let's have a column-vector query with f features denoted as $X = X^1 = \{X_1^1, X_2^1, \ldots, X_f^1\}$. Then the classification of X according to M can be expressed as:

$$C_M(X) = argmax \quad \prod_{l=1}^{L} f^l\left(M^l \times X^l\right) \tag{3}$$

where $X^{l+1} = f^l\left(M^l \times X^l\right)$ for $1 \le l \le L$.

4.1.6. Multi-query classification

In Sections 4.1.1–4.1.5 a single query column-vector X was classified by multiplying it with the trained matrix model M according to (2), which is basically a matrix–vector multiplication. Intuitively, if instead of one query column vector, q such column queries can be stacked to form a $(f+1)\times q$ query matrix S, then, with a single matrix–matrix multiplication, q queries can be classified. To this end, let's define the query matrix $S = \left[\left\{X^{(k)}\right\}_{k=1}^{q}\right]_{(f+1)\times q}$, for $1 \le k \le q$, where the column-vector $X^{(k)} = \left\{1, X_1^{(k)}, \ldots, X_f^{(k)}\right\}$ is the k-th query and $X_i^{(k)} \in F_i$. Then the simultaneous multi-query classification of q such queries is done with the multiplication of the $c \times (f+1)$dimensional trained matrix model M with the $(f+1)\times q$ dimensional query matrix S, as shown in (4) and illustrated in Fig. 4:

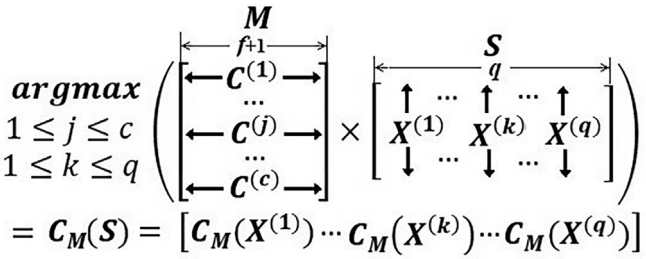

Fig. 4. Illustration of the proposed multi-query ML classifications.

$$C_M(S) = \underset{1\le j\le c, 1\le k\le q}{argmax}[M \times S] \tag{4}$$

1) Similarly, for the DNN case, Eq. (4) can be expressed as:

$$C_M(S) = argmax \quad \prod_{l=1}^{L} f^l\left(M^l \times S^l\right) \tag{5}$$

where $S^{l+1} = f^l\left(M^l \times X^l\right)$ for $1 \le l \le L$, and $S^1 = S$.

4.2. Somewhat homomorphic encryption (SWHE)

Somewhat Homomorphic Encryption (SWHE) public cryptosystems allow for additions and a limited number of multiplications to be done over data (ciphertexts) while they are in encrypted form (Fan and Vercauteren, 2012). These operations are known as homomorphic additions and multiplications. The number of the allowed homomorphic multiplications without decrypting the ciphertext is known as the depth of the circuit. SWHE schemes are based on the hardness of the Decision-RLWE problem, formally given in Section 9, which is assumed to be resilient to attacks from quantum computer (Brakerski et al., 2014; Fan and Vercauteren, 2012; Smart and Vercauteren, 2014; Kjamilji et al., 2021a). In SWHE schemes the plaintexts are polynomial rings with modulus X^{N+1} and integer coefficients with modulus t, and their corresponding ciphertext are polynomial rings with modulus X^{N+1} and integer coefficients with modulus q, such that q is much greater than t, thus $q >> t$ (Brakerski et al., 2014; Fan and Vercauteren, 2012). If the polynomial degree N and the coefficients t and q are properly chosen, then in a single plaintext and ciphertext up to N integers can be encoded/encrypted (Smart and Vercauteren, 2014). Consequently, this allows for simultaneous SIMD (Single Instruction Multiple Data) and slot (index) wise homomorphic additions and multiplications of N integers without extra costs (Fig. 5). Throughout the paper, it is assumed that parameters N, q and t are chosen

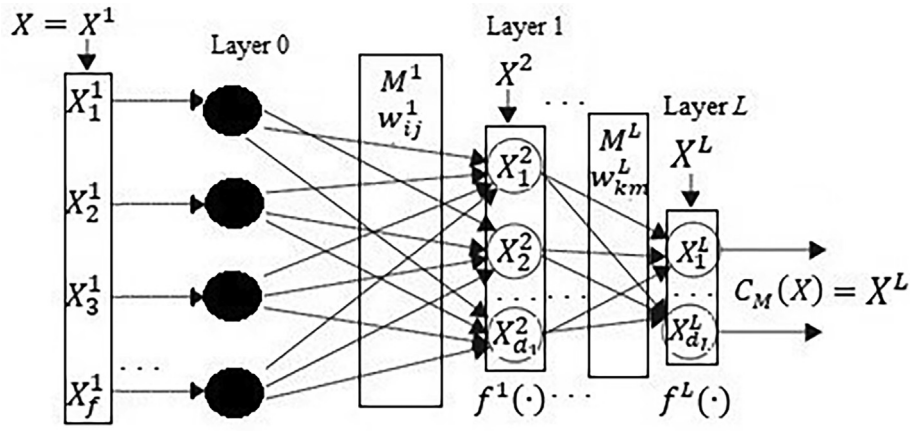

Fig. 3. Illustration of a multi layered Deep Neural Network (DNN).

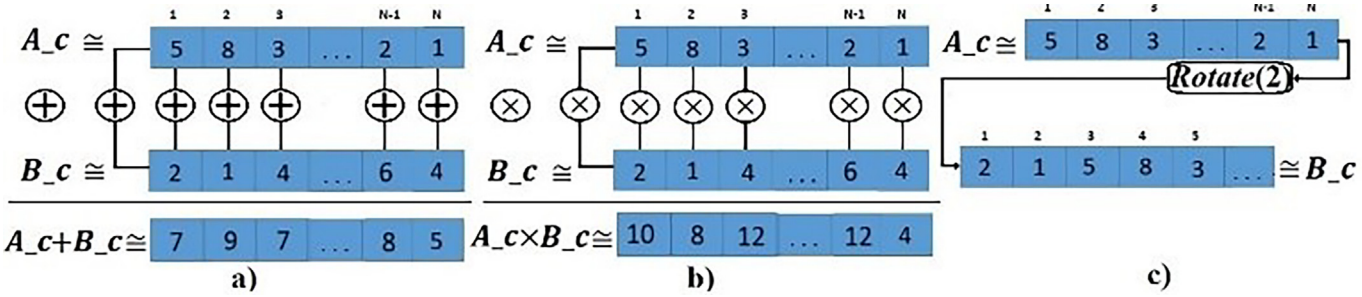

Fig. 5. SWHE SIMD homomorphic a) addition; b) multiplication. c) rotation.

in a way that enables homomorphic SIMD operations over encoded/encrypted plaintexts/ciphertexts.

Throughout the paper integer vectors are denoted by having "_*v*" at the end of their names, integer vectors encoded into plaintexts are denoted by having "_*p*" at the end of their names, and integer vector encoded and encrypted into ciphertexts are denoted by having "_*c*" at the end of their names. The main operations of an SWHE public key cryptosystem scheme are as follows (Fan and Vercauteren, 2012):

- $\boldsymbol{m_p} = \boldsymbol{Encode}(\boldsymbol{m_v})$: Encoding an integer vector into a plaintext.
- $\boldsymbol{m_p} = \boldsymbol{Encode}(\boldsymbol{m_v})$: Encoding and encrypting and integer vector into a ciphertext using one of the keys.
- $\boldsymbol{C_c} = \boldsymbol{A_c} + \boldsymbol{B_p}$: SIMD addition of two ciphertexts. The result is a ciphertext (Fig. 5a).
- $\boldsymbol{C_c} = \boldsymbol{A_c} + \boldsymbol{B_p}$: SIMD addition of a ciphertext with a plaintext. The result is a ciphertext (Fig. 5a).
- $\boldsymbol{m_c} = \boldsymbol{AddMany}(\boldsymbol{m_c}[])$: Takes a vector of ciphertexts as an input and returns their sum.
- $\boldsymbol{C_c} = \boldsymbol{A_c} \times \boldsymbol{B_p}$: SIMD multiplication of two ciphertexts. The result is always a ciphertext (Fig. 5b).
- $\boldsymbol{C_c} = \boldsymbol{A_c} \times \boldsymbol{B_c}$:. SIMD multiplication of a ciphertext with another. The result is always a ciphertext (Fig. 5b).
- $\boldsymbol{B_c} = \boldsymbol{Rotate}(\boldsymbol{A_c}, \boldsymbol{R})$: Rotation of the indexes (coefficients) of a ciphertext by R (Fig. 5c). If $R > 0$ the rotation is done to the right for R slots, otherwise if $R < 0$ the rotation is done to the right for R slots.
- $\boldsymbol{m_v} = \boldsymbol{Decode}(\boldsymbol{m_p})$; Decoding a plaintext into an integer vector.
- $\boldsymbol{m_v} = \boldsymbol{DecrDec}(\boldsymbol{m_c})$ Decrypting and decoding a ciphertext into an integer vector using the other key.

5. Novel Secure Linear Algebra and its homomorphic computation complexity

A row or a column vector can be trivially encoded in a SWHE plaintext or a ciphertext. As illustrated in Fig. 6, two ways of encoding matrices into SWHE plaintexts or ciphertexts are proposed. In a row-wise encoding all the rows are arranged (put) in a SWHE plaintext/ciphertext in ordered fashion. Similarly, in column-wise encoding all the columns are arranged (put) in a SWHE plaintext/-ciphertext in ordered fashion. A row-wise encoded matrices is denoted by putting an R as a superscript in their name, e.g., mat^R, while for the column-wise encoded matrices a C as a superscript is used in their name, e.g., mat^C (Fig. 6). In this way, in a plaintext/-ciphertext of polynomial degree N (see Section 4.2), $\frac{N}{d}$ vectors of dimension d and $\frac{N}{(d_1 \times d_2)}$ matrices of dimension $d_1 \times d_2$ can be

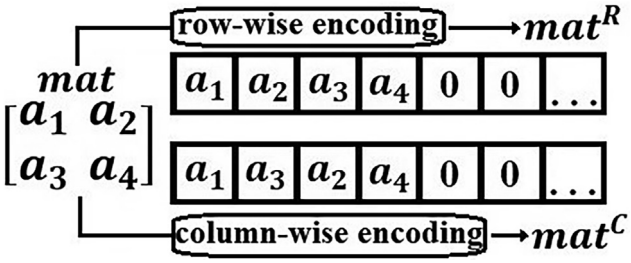

Fig. 6. Row and column-wise encoding of a 2×2 integer matrix into a SWHE plaintext/ciphertext.

packed (encoded). Furthermore, it is assumed that all the dimensions of the vectors and matrices are powers of 2. If it is not the case, then the remaining slots can be filled up with zeros up until the dimensions are a power of two, as it is done in (Kjamilji et al., 2021a; Juvekar et al., 2018; Jiang et al., 2018).

5.1. Novel secure linear algebra

This section introduces over a dozen of secure linear algebra operations, which, among others, can be also used to provide secure and private ML classification algorithms, as it is shown in Section 6.

5.1.1. Secure sums of blocks of d slots (secSum)

Firstly proposed in (Kjamilji et al., 2021a; Juvekar et al., 2018; Jiang et al., 2018), it is shown in Algorithm 1. As an input it takes a ciphertext *input_c* and an integer d, which is the number of consequtive slots (indexes) per block. As an output it provides the sum of all of the blocks of d slots in a way that the sum of the block of the first d slots is stored in the first slot (index), the sum of the block of the second d slots is stored in the d-th slot, and in general the sum of blocks of the i-th block of d slots is stored in the $(i-1) \times d$-th slot. The sums are found in $log_2(d)$ iterations, such that at the i-th iteration the result of the previous iteration is added with itself rotated by 2^i slots to the left (lines 1–3). In line 4 a mask plaintext named *mask_p* is constructed, which, starting from the first slot (index), it has ones (1) after each d slots and has zeros (0) in all the other remaining slots. In line 5 this mask is then multiplied with the result found in lines 1–3 to hide the intermediate results (sums). If Algorithm 1 is proceeded by another algorithm and if it is feasible to do so, then line 5 can be skipped for performance reasons. For a ciphertext polynomial degree of N, in total Algorithm 1 calculates the sum of $\frac{N}{d}$ blocks of d slots

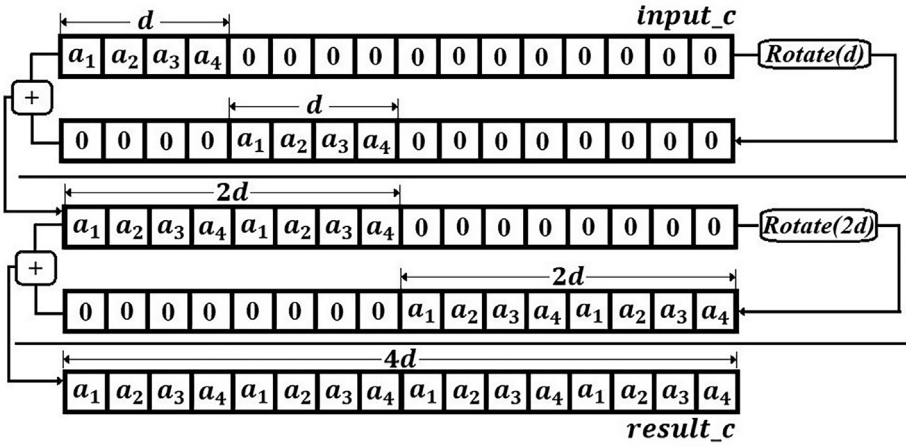

Fig. 7. Illustration of $CRep(\cdot)$ for $d = 4$ *and* $r = 4$.

ALGORITHM 1: *secSum* **- secure Sums of blocks of slots**

INPUT: $\boldsymbol{input_c, d}$
$\boldsymbol{d}$: the number of slots per block for which the sum is found

OUTPUT: $\boldsymbol{result_c}$
$\boldsymbol{result_c}$: has the sum of each of the N/d blocks of d slots at the first slot of the corresponding block

1 $\boldsymbol{result_c = input_c}$
2 **for** $\boldsymbol{i = 0}$ **to** $\boldsymbol{\log_2(d)}$
3 $\quad \boldsymbol{result_c = result_c + Rotate\ (result_c, -2^i)}$
4 $\boldsymbol{mask_p = secSumMask(d)}$
5 $\boldsymbol{result_c = result_c \times mask_p}$

5.1.2. Ciphertext Replication (CRep)

It is given in Algorithm 2. As an input it takes the replication rate r, and a ciphertext *input_c* which has some integer data in its first d slots and at least $(r-1)\cdot d$ zeros in the subsequent slots (Fig. 7). As an output it gives a ciphertext which has the input's first d slots replicated by r times. This pattern (of d data slots proceeded by at least $(r-1)\cdot d$ subsequent zeros) might be repeated till the end of the remaining N slots, where N is the polynomial modulus, as it is shown in the last two vectors of Fig. 8. The replication is done in line 2–3 by $log_2(r)$ SIMD homomorphic additions and rotations. *CRep* is illustrated in Fig. 7 and in the last two vectors of Fig. 8. If the polynomial modulus is N, and the replication rate is r, then simultaneously $\frac{N}{(d\times r)}$ciphertexts with d data slots can be replicated.

ALGORITHM 2: *CRep* **- Ciphertext Replication**

INPUT: $\boldsymbol{input_c, d, r}$
$\boldsymbol{input_c}$: a ciphertext that will be homomorphically replicated
$\boldsymbol{d}$: number of data slots in *input_c* (starting from the first slot)
$\boldsymbol{r}$: the replication rate

OUTPUT: $\boldsymbol{result_c}$
$\boldsymbol{result_c}$: *input_c*'s d data slots replicated for r times

1 $\boldsymbol{result_c = input_c}$
2 **for** $\boldsymbol{i = 0}$ **to** $\boldsymbol{\log_2(r)}$
3 $\quad \boldsymbol{result_c = result_c + Rotate\ (result_c,\ 2^i \times d)}$

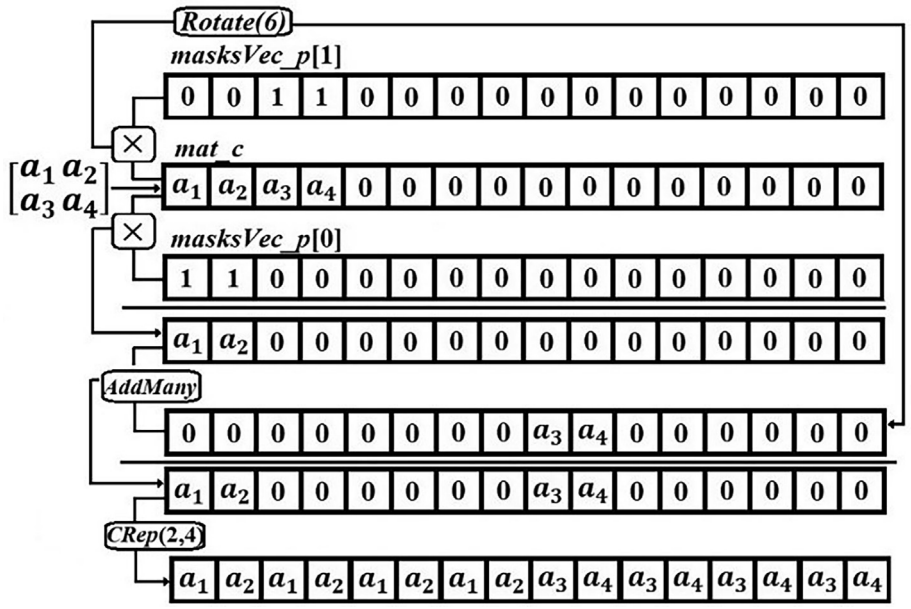

Fig. 8. Illustration of *secDRep*$(\cdot)$ for $d_1 = 2, d_2 = 2$and $r = 4$.

5.1.3. Secure Dimension Replication (secDRep)

It is given in Algorithm 3 and illustrated in Fig. 8. As an input it takes an encoded and encrypted matrix ciphertext *mat_c* of dimensions $d_1 \times d_2$ as well as a replication rate r. As an output it gives a ciphertext in which each of the d_1 dimensions of d_2 slots are replicated by r times. In line 1 a vector of plaintext masks is constructed for each of the d_1 dimensions. In this sense, for $0 \leq i < d_1$ the mask of dimension i is *masksVec*[i], which has ones starting from the $(i \times d_2)$-th slot till the $((i+1) \times d_2)$-th slot, and everywhere else is zeros. The bulk of the algorithm is done in d_1 iterations in lines 3–5. In line 4, at the i-th iteration, for $1 \leq i < d_1$, after multiplying the input ciphertext with the i-th mask *masksVec*[i], the resulting ciphertext *tmp_c*[i] is rotated to the right by $i \times d_2 \times (r-1))$ slots so as to pave the way for replications of the dimensions before it. In line 6 all the d_1 *tmp_c*[i]'s are added up and in the resulting ciphertext all the dimensions are simultaneously replicated by r times in line 7 by calling Algorithm 2 (CRep) (Fig. 8). If the polynomial modulus is N, and the replication rate is r, in parallel a total of $\frac{N}{(d_1 \times d_2 \times r)}$ matrices of dimensions $d_1 \times d_2$ can be replicated.

ALGORITHM 3: ***secDRep*** **- secure Dimension Replication**

INPUT: $\boldsymbol{mat_c, d_1, d_2, r}$
$\boldsymbol{mat_c}$: an encoded and encrypted matrix
$\boldsymbol{d_1, d_2}$: the first&second dimensions of *mat_c*, respectively
$\boldsymbol{r}$: the replication rate

OUTPUT: $\boldsymbol{matDRep_c}$
$\boldsymbol{matDRep_c}$: *input_c*'s d_1 dimension replicated by r times

1 $\boldsymbol{masksVec_p[] = secDRepMasks(d_1, d_2)}$ //constructs the masks
2 $\boldsymbol{tmp_c[0] = mat_c \times masksVec_p[0]}$
3 **for** $\boldsymbol{i = 1}$ **to** $\boldsymbol{d_1}$
4 $\quad \boldsymbol{tmp_c[i] = mat_c \times masksVec_p[i]}$
5 $\quad \boldsymbol{tmp_c[i] = Rotate\ (tmp_c[i],\ i \times d_2 \times (r-1))}$
6 $\boldsymbol{matDRep_c = AddMany(tmp_c[])}$ //$log d$ addit.
7 $\boldsymbol{matDRep_c = CRep(matDRep_c, d_2, r)}$

5.1.4. Secure Dot (Inner) Product (secDotP)

It is given in Algorithm 4. As an input it takes two vector ciphertexts od dimension d. As an output it returns a ciphertext which in its first slot has the dot product of the input vectors. Initially, the inputs are multiplied (line 1) and then the sum of the resulting vector is found (line 2). If the polynomial modulus is N, in parallel the dot product of $\frac{N}{d}$ pairs of vectors of dimension d can be calculated.

ALGORITHM 4: ***secDotP*** **- secure Dot (Inner) Product**

INPUT: $\boldsymbol{vec1_c, vec2_c, d}$
$\boldsymbol{vec1_c, vec2_c}$: two (packed) encrypted integer vectors
$\boldsymbol{d}$: the dimension of the vectors

OUTPUT: $\boldsymbol{result_c}$
$\boldsymbol{result_c}$: the result of the dot product contained at the first slot (of each block)

1 $\boldsymbol{result_c = vec1_c \times vec2_c}$
2 $\boldsymbol{result_c = secSum(result_c, d)}$

5.1.5. Secure Matrix-Vector Multiplication (secMatVec)

It is given in Algorithm 5. As an input it takes a row-encoded matrix ciphertext mat^R_c of dimensions $d_1 \times d_2$ and a column encoded vector ciphertext *vec_c*of dimension d_2. As an output it returns a sparsely encoded column-*vector* ciphertext which is the result of the matrix–vector multiplication. In line 1 the input vector is replicated by d_1 times, and in line 2 Algorithm 4 (secDotP) is called to find the final matrix–vector multiplication result. The resulting column-vector ciphertext has its d_1 elements encoded by having $d_2 - 1$ empty slots in-between them. If the polynomial modulus is N, in parallel the matrix–vector product of $\frac{N}{(d_1 \times d_2)}$ matri-

ces of dimension $d_1 \times d_2$ and vectors of dimension d_2 can be calculated.

ALGORITHM 5: *secMatVec* – secure Matrix-Vector Multiplication

INPUT: $\boldsymbol{mat^R_c, vec_c, d_1, d_2}$
$\boldsymbol{mat^R_c}$: $d_1 \times d_2$ dimen. row-encoded (packed) matrix(es)
$\boldsymbol{vec_c}$: a (packed) column vector(s) with dimension d_2

OUTPUT: $\boldsymbol{result_c}$
$\boldsymbol{result_c}$: the result of the mat-vec. prod. after each d_2 slots

1 $\boldsymbol{vec_c = CRep(vec_c, d_2, d_1)}$
2 $\boldsymbol{result_c = secDotP(mat_c, vec_c, d_2)}$

ALGORITHM 6: *secMatMat* – secure Matrix-Matrix Product

INPUT: $\boldsymbol{mat^R_c, mat^C_c, d_1, d_2, d_3}$
$\boldsymbol{mat^R_c}$: row-encoded. (packed) matrix(es) with dim. $d_1 \times d_2$
$\boldsymbol{mat^C_c}$: col.-encoded. (packed) matrix(es) with dim $d_2 \times d_3$

OUTPUT: $\boldsymbol{result_c}$
$\boldsymbol{result^C_c}$: sparsely encoded column-vector of dim. $d_1 \times d_3$.

1 $\boldsymbol{mat^R_c = CRep(mat^R_c, d_1 \times d_2, d_3)}$
2 $\boldsymbol{mat^C_c = secDRep(mat^C_c, d_3, d_2, d_1)}$
3 $\boldsymbol{result^C_c = secDotP(mat^R_c, mat^C_c, d_2)}$

5.1.6. Secure Matrix-Matrix Multiplication (secMatMat)

It is given in Algorithm 6. As an input it takes a row-encoded matrix ciphertext mat^R_c with dimensions $d_1 \times d_2$ and a column-encoded matrix ciphertext mat^C_c with dimensions $d_2 \times d_3$. As an output it returns a sparsely column-encoded matrix ciphertext $result^C_c$ of dimensions $d_1 \times d_3$, which is the result of matrix–matrix mulitplication of the two input matrices. In line 1 the whole mat^R_c is replicated by d_3 times. In line 2 dimension d_3 of mat^C_c is replicated by d_1 times. In line 3 the dot product of the replicated matrices is applied to find the result. In this way, in line 3 all the $d_1 \times d_2 \times d_3$ plain matrix multiplications are done by a single SIMD homomorphic multiplication. The resulting column-encoded matrix ciphertext has its $d_1 \times d_3$ elements encoded by having $d_2 - 1$ empty slots in-between them. *secMatMat*$(\cdot)$ is illustrated in the upper part of Fig. 12. If the polynomial modulus is N, in parallel the matrix–matrix product of $\frac{N}{(d_1 \times d_2 \times d_3)}$ matrices of dimensions $d_1 \times d_2$ and dimensions $d_2 \times d_3$, respectively, can be calculated.

5.1.7. Secure Ciphertext Compress (CCompress)

Algorithm 7 (*CCompress*$(\cdot)$) as an input takes a ciphertext that has *nrOfBlocks* number of blocks to be compressed, each with size *blockSize* separated by d slots from each other. In the end it compresses (brings them closer) by reducing the distance between blocks, or by merging neighboring blocks in each iteration and when necessary, in certain iterations by multiplying with a corresponding plaintext mask to free up space for further mergers in the upcoming iteration (lines 3–11). Depending on the input parameters, the resulting ciphertext can have all of the data slots of all of the blocks compressed together without any gaps (slot distances) between them, as it is illustrated in Fig. 9a for input parameters $nrOfBlocks = 4$, $blockSize = 1$, $d = 1$. One way to interpret *CCompress*$(\cdot)$ is that as an input it takes a sparsely encoded matrix with dimensions $d_1 \times d_2 = nrOfBlocks$ (as it is the case with the resulting matrix at the end of *secMatMat*$(\cdot)$) or a sparsely encoded vector (as it is the case with the resulting vector at the end of *secMatVec*$(\cdot)$) with their elements having d empty slots in between, At the end *CCompress*$(\cdot)$ compresses the input so there are no empty slots in-between the encoded elements (Fig. 9a).

ALGORITHM 7: *CCompress* – Ciphertext Compress

INPUT: $\boldsymbol{input_c, blockSize, d, nrOfBlocks}$
$\boldsymbol{input_c}$: a sparsely encoded $d_1 \times d_2$ matrix ciphertext
$\boldsymbol{blockSize}$: number of slots per block
$\boldsymbol{d}$: empty slots between two neighboring data blocks
$\boldsymbol{nrOfBlocks = d_1 \times d_2}$: number of blocks (matrix elements) to be compressed

OUTPUT: $\boldsymbol{result_c}$
$\boldsymbol{result_c}$: the compressed ciphertext

1 $\boldsymbol{emptySlots = d}$
2 $\boldsymbol{result_c = input_c}$
3 $\boldsymbol{for\ i = 0\ to\ \log_2(nrOfBlocks)}$
4 $\quad \boldsymbol{tmp_c = Rotate\ (result_c, -d)}$
5 $\quad \boldsymbol{result_c = result_c + tmp_c}$
6 $\quad \boldsymbol{emptySlots = emptySlots - blockSize}$
7 $\quad \boldsymbol{d = 2 \times d}$
8 $\quad \boldsymbol{blockSize = 2 \times blockSize}$
9 $\quad \boldsymbol{if(emptySlots == 0)}$
10 $\quad\quad \boldsymbol{result_c = result_c \times maskCComp_p[blockSize, d]}$
11 $\quad\quad \boldsymbol{emptySlots = blockSize}$

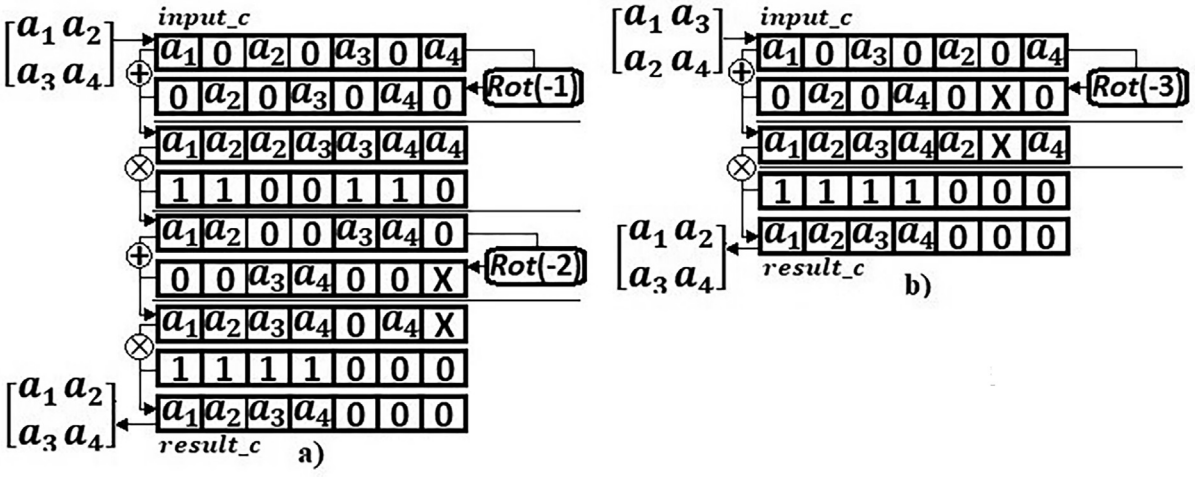

Fig. 9. Illustration of a) *CCompress*$(\cdot)$ for input parameters $nrOfBlocks = d_1 \times d_2 = 4$, $blockSize = 1$, $d = 1$, and b) *CCompress_v2*$(\cdot)$ for input parameters $d = 1$, $d_1 = 2, d_2 = 2$.

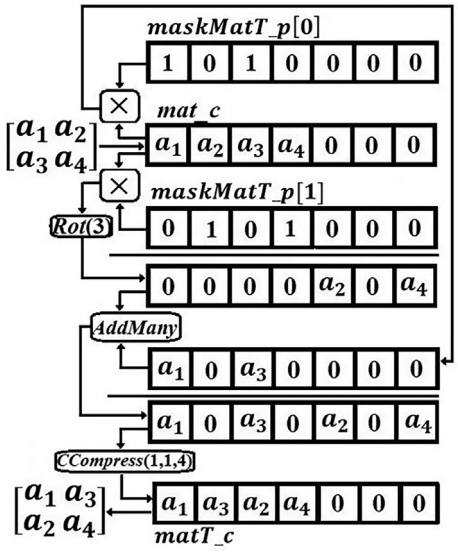

Fig. 10. Illustration of *secMatT*$(\cdot)$ for an input matrix *mat_c* with dimension $d_1 = 2$ and $d_2 = 2$ to get its transposed matrix *matT_c*.

5.1.8. Secure Ciphertext Compress – version 2 (CCompress_v2))

The way to interpret Algorithm 8 (*CCompress_v2*$(\cdot)$) is similar to *CCompress*$(\cdot)$, in a sense that *CCompress_v2*$(\cdot)$) takes a sparsely encoded input ciphertext matrix with dimesnions $d_1 \times d_2$ with d empty slots in between the elements. At the end of *CCompress_v2*$(\cdot)$ the resulting ciphertext is the compressed transponse matrix of the input with no empty slots in-between the encoded elements. *CCompress_v2*$(\cdot)$) is illustrated in Fig. 9b for input parameters $d = 1$, $d_1 = 2, d_2 = 2$.

ALGORITHM 8: ***CCompress_v2* – Ciphertext Compress version 2**

INPUT: $\boldsymbol{input_c, d, d_1, d_2}$
input_c: a sparsely encoded (packed) $d_1 \times d_2$ ciphertext matrix
$\boldsymbol{d}$: empty slots between two neighboring data slots, $d + 1 \geq d_1$
$\boldsymbol{d_1}$: the first matrix dimension (rows)
$\boldsymbol{d_2}$: the second matrix dimension (columns)

OUTPUT: ***result_c***
result_c: the compressed ciphertext

```
emptySlots = d
blockSize = 1
result_c = input_c
for i = 0 to log2(d1)
    tmp_c = Rotate(result_c, −(i + 1) × ((d + 1) × d2 − 1))
    result_c = result_c + tmp_c
    emptySlots = emptySlots − blockSize
    blockSize = 2 × blockSize
result_c = result_c × maskCCompres_v2(d, d1, d2)
if(emptySlots != 0)
    CCompress(result_c, blockSize, emptySlots, d2)
```

5.1.9. Secure Matrix Transponse (secMatT)

Given in Algorithm 9 and illustrated in Fig. 10, it securely transposes an input matrix with dimensions $d_1 \times d_2$. Using the corresponding d_2 number of plaintext masks, the input matrix is multiplied by them and each of them has the columns of the input

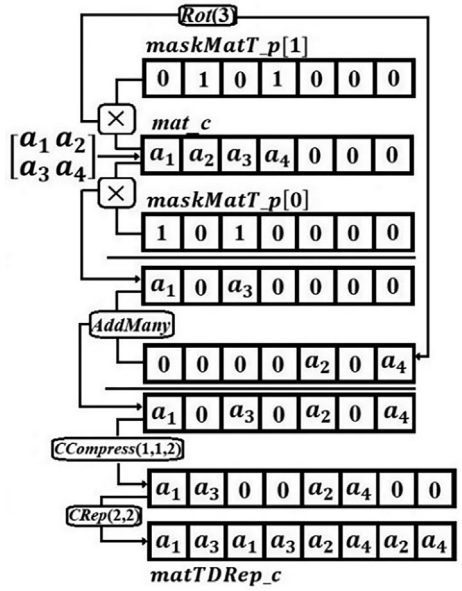

Fig. 11. Illustration of *secMatTDRep*$(\cdot)$ for $d_1 = 2, d_2 = 2$ *and* $r = 2$.

matrix (which are going to be the rows of the resulting output) whose elements are scattered in distance by d_2 slots from each other. In this way the input matrix is firstly converted into sparsely encoded transposed matrix ciphertext (lines 1–4), to which *CCompress*$(\cdot)$ is applied to get the final transposed matrix result. The illustration of *secMatT*$(\cdot)$ is given in Fig. 10 for $d_1 = 2$ and $d_2 = 2$.

ALGORITHM 9: ***secMatT*** **– Secure Matrix Transponse**

INPUT: $\boldsymbol{mat_c, d_1, d_2}$
$\boldsymbol{mat_c}$: a matrix with dimensions $d_1 \times d_2$

OUTPUT: $\boldsymbol{result_c}$
$\boldsymbol{matT_c}$: *mat_c* transposed

1 $\boldsymbol{for\ i = 0\ to\ d_2}$
2 $\quad \boldsymbol{tmp_c[i] = mat_c \times maskMatT_p(i, d_1, d_2)}$
3 $\quad \boldsymbol{tmp_c[i] = Rotate(tmp_c[i], i \times (d_1 \times d_2 - 1))}$
4 $\boldsymbol{tmpRes_c = AddMany(tmp_c[])}$
5 $\boldsymbol{matT_c = CCompress(result_c, 1, d_2 - 1, d_1 \times d_2)}$

5.1.10. Secure Matrix Transponse and Dimension Replication (secMatTDRep)

Given in Algorithm 10. Instead of transposing the matrix using *secMatT*$(\cdot)$ and then replicating its dimensions by using *secDRep*$(\cdot)$, for efficiency purposes, this algorithm on the fly does them both. Thus, it replicates the dimensions by r of the transposed input matrix. It is illustrated in Fig. 11.

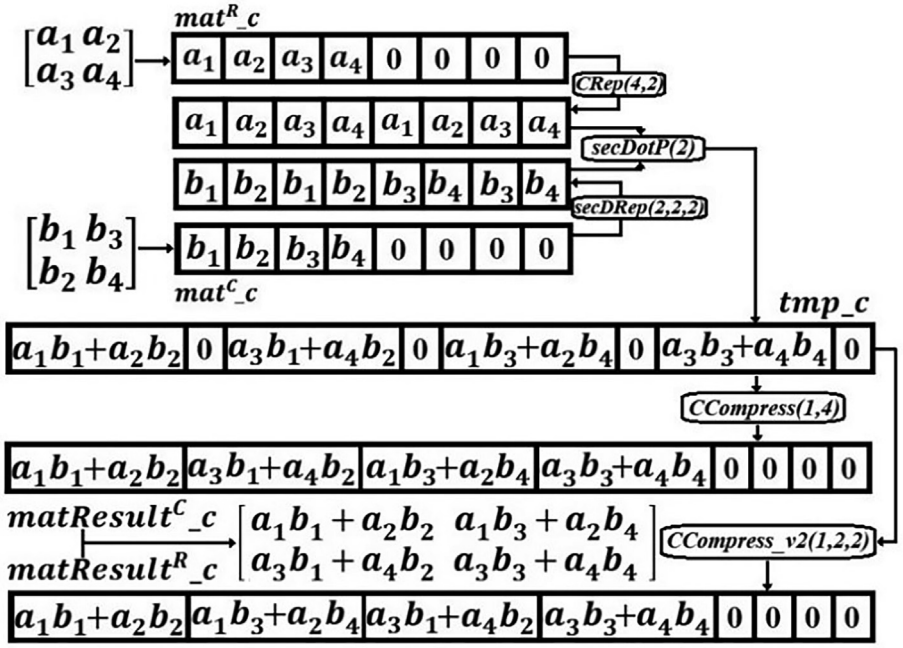

Fig. 12. Illustration of *secMatMatCas*$(\cdot)$: all of the steps (algorithms) that *secMatMat*$(\cdot)$ calls (i.e.*CRep*$(\cdot)$, *secDRep*$(\cdot)$and *secDotP*$(\cdot)$), proceeded with both *CCompress*$(\cdot)$ and *CCompress_v*2$(\cdot)$ options.

ALGORITHM 10: ***secMatTDRep*** **– secure Matrix Transponse and Dimension Replication**

INPUT: mat_c, d_1, d_2, r
mat_c: a matrix with dimensions $d_1 \times d_2$
r: the replication rate

OUTPUT: $result_c$
$matTDRep_c$: mat_c transposed and replicated by r

1 $for\ i = 0\ to\ d_2$
2 $\quad tmp_c[i] = mat_c \times maskMatT_p(i, d_1, d_2)$
3 $\quad if(r > d_2)$
4 $\quad\quad tmp_c[i] = Rotate(tmp_c[i], i \times (d_1 \times r - 1))$
5 $\quad else$
6 $\quad\quad tmp_c[i] = Rotate(tmp_c[i], i \times (d_1 \times d_2 - 1))$
6 $tmpRes_c = AddMany(tmp_c[])$
7 $tmpRes_c = CCompress(result_c, 1, d_2 - 1, d_1)$
8 $if(r > d_2)$
9 $\quad tmpRes_c = tmpRes_c \times maskMatTDRep(d_1, d_1 \times (r - 1))$
10 $\quad matTDRep_c = CRep(tmpRes_c, d_1, r)$
11 $else$
12 $\quad tmpRes_c = tmpRes_c \times maskMatTDRep(d_1, d_1 \times (d_2 - 1))$
13 $\quad tmpRes_c = CRep(tmpRes_c, d_1, r)$
14 $\quad matTDRep_c = CCompress(result_c, r \times d_1, d_1 \times (d_2 - r), d_2)$

5.1.11. Secure Cascading Matrix-Vector Product (secMatVecCas)

Given in Algorithm 11. After *secMatVec*(·), the resulting ciphertext vector has its data slots sparsely separated from each other and this makes it unsuitable for cascading (another round) of linear algebra multiplication. In order to make the resulting ciphertext vector suitable for another matrix–vector multiplication, after *secMatVec*(·), the data slots are compressed using the *CCompress*(·) (line 2).

ALGORITHM 11: ***secMatVecCas*** **- secure Matrix-Vector Cascading Product**

INPUT: mat^R_c, vec_c, d_1, d_2
mat^R_c: row-encoded (packed) matrix(es) with dimen. $d_1 \times d_2$
vec_c: a (packed) column vector(s) with dimension d_2

OUTPUT: $result_c$
$result_c$: the compressed result of the matrix-vector product

1 $tmp_c = secMatVec(mat^R_c, vec_c, d_1, d_2)$
2 $result_c = CCompress(tmp_c, 1, d_2 - 1, d_1)$

5.1.12. Secure Cascading Matrix-Matrix Product (secMatMatCas)

Given in Algorithm 12. After *secMatMat*(·) (line 1), the resulting ciphertext has its data slots sparsely separated from each other and this makes it unsuitable for cascading (another round) of linear algebra multiplication. In order to make the resulting ciphertext suitable for another round of multiplication, after *secMatMat*(·) the data slots are compress using the *CCompress*(·) Algorithm (line 2) to get the compressed and column-encoded result matrix. If as an output it is desired the compressed matrix to be in row-encoded form, instead of *CCompress*(·) in line 2 *CCompress_v2*(·) is called. Fig. 12 illustrates all of the steps (algorithms) that *secMatMat*(·) calls (i.e.*CRep*(·), *secDRep*(·)and *secDotP*(·)), proceeded with both *CCompress*(·) and *CCompress_v2*(·) options.

ALGORITHM 12: ***secMatMatCas*** **- secure Matrix-Matrix Cascading Product**

INPUT: $mat^R_c, mat^C_c, d_1, d_2, d_3$
mat^R_c: row-encoded (packed) matrix(es) with dimen. $d_1 \times d_2$
mat^C_c: column-encoded (packed) matrix(es) with dimen. $d_2 \times d_3$

OUTPUT: $result_c$
$result_c$: the compressed result of the mat-mat product

1 $tmp_c = secMatMat(mat^R_c, mat^C_c, d_1, d_2, d_3)$
2 $matResult^C_c = CCompress(tmp_c, 1, d_2 - 1, d_1 \times d_3)$
//2 $matResult^R_c = CCompress_v2(tmp_c, d_2 - 1, d_1, d_3)$

Table 2
Homomoprhic complexity of the proposed secure linear algebra algorithms.

Algorithm	ADD	PMUL	ROT	CMUL	DEPTH
secSum	logd	1	logd	0	1 PMUL
CRep	logd	0	logd	0	0
secDRep	2logd	d	d + logd	0	1 PMUL
secDotP	logd	1	logd	1	1 CMUL, 1 PMUL
secMatVec	2logd	1	2logd	1	1 CMUL, 1 PMUL
CCompress^{+}	2logd	2	2logd	0	2 PMUL
secMatT	3logd	d + 2	d + 2logd	0	3 PMUL
CCompress_v2^{+}	logd	1	logd	0	1 PMUL
secMatTDRep	3logd	d + 1	d + 2logd	0	2 PMUL
secMatVecCas	3logd	2	3logd	1	1 CMUL, 2 PMUL
secMatVec (C)*	logd	2	logd	0	2 PMUL
secMatVec (Juvekar et al., 2018)	d	d	d-1	0	2 PMUL
secMatMat	4logd	d + 1	d + 3logd	1	1 CMUL,2 PMUL
secMatMatCas, (with CCompres_v2)	6logd, (5logd)	d + 3, (d + 2)	d + 5logd, (d + 4logd)	1, (1)	1 CMUL, 4 PMUL (1 CMUL, 3 PMUL)
secMatMat_v2	5logd	d + 2	d + 4logd	1	1 CMUL, 3 PMUL
secMatMatCas_v2, (with CCompres_v2)	7logd, (6logd)	d + 4, (d + 3)	d + 6logd, (d + 5logd)	1, (1)	1 CMUL, 5 PMUL (1 CMUL, 4 PMUL)
secMatMat (Jiang et al., 2018)	6d	4d	$3d+5\sqrt{\mathbf{d}}$	d	1 CMUL. 2 PMUL

+ for *CCompress* it is assumed that.$nrOfBlocks = d \cdot d = d^2$.
*For benchmark purposes with (Juvekar et al., 2018) the vector is packed and in plain.
**All the matrices are squared with dimension *d*. All logs are in base 2.
***ADD = Ciphertext Addition, PMUL = Plain Multiplication,
ROT = Ciphertext Rotation, CMUL = Ciphertext Multiplication.

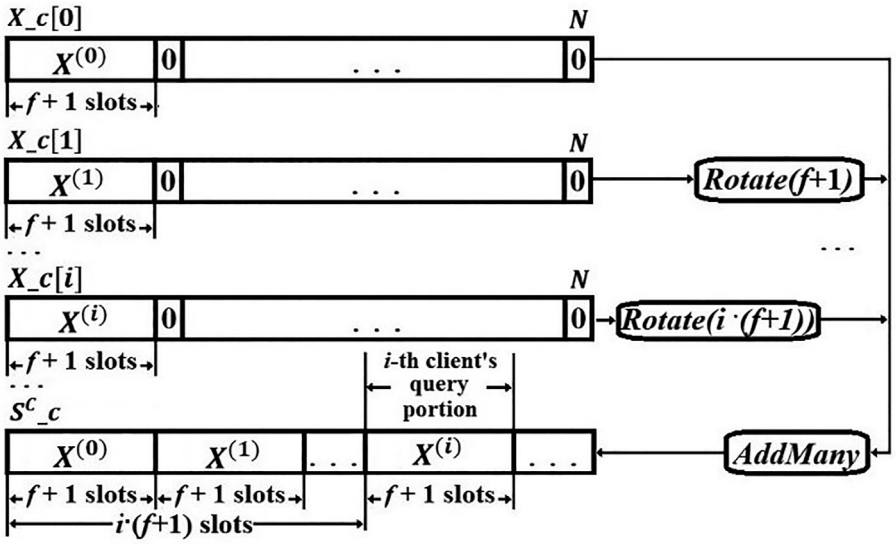

Fig. 13. Construction of the column-wise queries' matrix S^C_c where each of the q clients has its own query portion of $f+1$ slots in S^C_c.

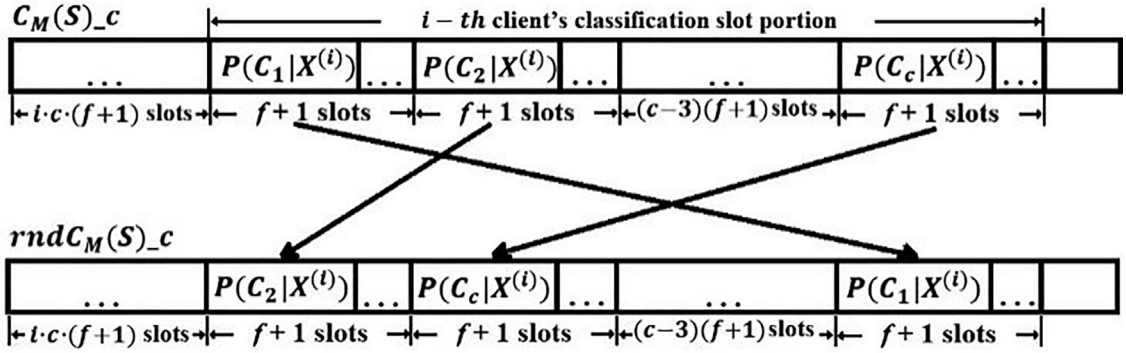

Fig. 14. Illustration of $SRCPer(\cdot)$ over client i's classification portion of $C_M(S)_c$ to get $rndC_M(S)_c$, c class values have f empty slots in between.

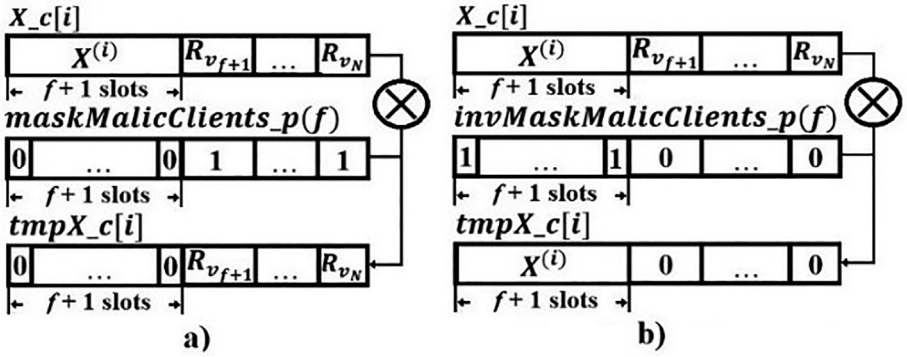

Fig. 15. Multiplying client queries with a) a plaintext mask with zeros in the first $f+1$ slots and ones elsewhere; b) a plaintext mask with ones in the first $f+1$ slots and zeros elsewhere (the inverse plaintext of a)).

5.1.13. Secure Matrix-Matrix Product – version 2 (secMatMat_v2)

Given in Algorithm 13. While *secMatMat*$(\cdot)$ takes one row-encoded and one column-encoded matrix as input, *secMatMat_v2*$(\cdot)$ takes two row-encoded matrices as its input. After replicating the dimensions of the first input matrix by using *CRep*$(\cdot)$ (line 1) and replicating the dimensions of the transposed second input matrix by using *secMatTDRep*$(\cdot)$ (line 2), it simply converts the matrix–matrix multiplication into a single dot product (line 3). All of the $d_1 \times d_2 \times d_3$ plain matrix multiplications are done by a single SIMD homomorphic multiplication in line 3. The resulting column-encoded matrix ciphertext has its $d_1 \times d_3$ elements encoded by having $d_2 - 1$ empty slots in-between them.

5.1.14. Secure Cascading Matrix-Matrix Product – version 2 (secMatMatCas_v2)

Given in Algorithm 14. After *secMatMat_v2*$(\cdot)$, the resulting ciphertext has its data slots sparsely separated from each other and this makes it unsuitable for cascading (another round) of linear algebra multiplication. To make the resulting ciphertext suitable for another multiplication, after *secMatMat_v2*$(\cdot)$, the data slots are compressed using the *CCompress*$(\cdot)$ Algorithm (line 2) to get the compressed and column-encoded result matrix. If as an output it is desired the compressed matrix to be in row-encoded form, instead of *CCompress*$(\cdot)$, in line 2 *CCompress_v2*$(\cdot)$ is called.

ALGORITHM 13: ***secMatMat_v2*** **- secure Matrix-Matrix – version 2 Product**

INPUT: $\boldsymbol{mat^R_c, mat^R_c, d_1, d_2, d_3}$
$\boldsymbol{mat1^R_c}$: row-encoded (packed) matrix(es) with dimen. $d_1 \times d_2$
$\boldsymbol{mat2^R_c}$: row-encoded (packed) matrix(es) with dimen .$d_2 \times d_3$

OUTPUT: $\boldsymbol{result_c}$
$\boldsymbol{result_c}$: the result of the mat.– mat. product after each d_2 slots

1 $\boldsymbol{mat1^R_c = CRep(mat1^R_c, d_1 \times d_2, d_3)}$
2 $\boldsymbol{mat2^R_c = secMatTDRep(mat2^R_c, d_2, d_3, d_1)}$
3 $\boldsymbol{result_c = secDotP(mat1^R_c, mat2^R_c, d_2)}$

ALGORITHM 14: ***secMatMatCas_v2*** **- secure Matrix-Matrix Cascading– version 2 Product**

INPUT: $\boldsymbol{mat^R_c, mat^R_c, d_1, d_2, d_3}$
$\boldsymbol{mat1^R_c}$: row-encoded (packed) matrix(es) with dimen. $d_1 \times d_2$
$\boldsymbol{mat2^R_c}$: row-encoded (packed) matrix(es) with dimen. $d_2 \times d_3$

OUTPUT: $\boldsymbol{result_c}$
$\boldsymbol{result_c}$: the compressed result of the matrix - matrix product

1 $\boldsymbol{tmp_c = secMatMat_v2(mat1^R_c, mat2^R_c, d_1, d_2, d_3)}$
2 $\boldsymbol{matResult^C_c = CCompress(tmp_c, 1, d_2 - 1, d_1 \times d_3)}$
//2 $\boldsymbol{matResult^R_c = CCompress_v2(tmp_c, d_2 - 1, d_1, d_3)}$

Table 3
Amortized Operations' Costs (In mılıseconds).

Dimension d	4	8	16	32	64
secSum	0.0009	0.002	0.008	0.025	0.06
CRep	0.0019	0.01	0.1	0.68	4
secDRep	0.56	2.9	60	56.25	166.25
secDotP	0.0051	0.013	0.034	0.074	0.156
secMatVec;	0.034	0.175	0.92	4.125	18
CCompress	0.625	3.1875	26.75	26.5	96.25
secMatT	2.05	11.56	46	200	1003
HE-MatTrans (Jiang et al., 2018)	1	NR	49	NR	2557
secMatTDRep	1.72	9.43	94.5	173.75	1020
secMatVecCas	0.068	0.468	2.875	15	69.5
secMatMat	1.4	6.875	125	125.5	462.5
secMatMatCas	2.96	12.5	205.25	235.3	855.75
secMatMat_v2	3.21	18.1	234	314.5	1568
secMatMatCas_v2	5.48	26.46	398	476.2	2301
secMatMat (HE-MatMult) of (Jiang et al., 2018)	3	NR	162	NR	10^4

5.2. Theoretical analysis and comparisons

Table 2 shows the homomorphic complexity and circuit depth of the proposed secure linear algebra operations compared with the best reported results in (Juvekar et al., 2018; Jiang et al., 2018), since they are known to be among the best state-of-the-art schemes. All the proposed algorithms have a $O(logd)$ (logarithmic) complexity w.r.t. to the matrix (vector) dimension(s), except for the different flavors of matrix–matrix multiplications which are linear since they use *secDRep*$(\cdot)$ and *secMatTDRep*$(\cdot)$ which are linear $O(d)$.

Note 1: for all the proposed vector and matrix operations, any of the two inputs can be a plaintext. Also, it is not necessary for matrices to be squares, they can be rectangles.

Note 2: not only for cascading (sequential) operations, rather several flavors of vector and matrix operations are provided for flexibility as well. E.g., in some literature, for DNN, instead of row-encoding, the weights are column-encoded, or instead of matrix–vector multiplication as it is done in this paper, they use vector–matrix multiplication, etc**.** In such cases the proper use of algorithms such as *secMatT*$(\cdot)$, *secMatMat_v2*$(\cdot)$, *secMatMatCas_v2*$(\cdot)$, etc. comes into play.

6. Secure ML multi-query classifications

On one hand, some servers get overloaded during PP ML classifications. On the other hand, some clients do not have the resources to deal with such classifications which can be costly. To this end, based on where the bulk of operations are done, the flexibility of both the server and the client centric PP ML classification scenarios are provided. Since the initial phase where the clients construct their queries is the same for both client and server centric PP ML classifications, in the beginning the initial phase named *queryConstr*$(\cdot)$ is provided and then it is proceeded with both PP classification flavors in the next two sub-sections of this section. As an input *queryConstr*$(\cdot)$ (Algorithm 15) takes the number of classes and features, c and f, respectively, and the queries of q clients, where $q = \frac{N}{c(f+1)}$, and N is the polynomial modulus (Section 4.2). The clients locally encode their queries in the first $f+1$ slots and let all other slots to be zero. Then they encrypt their queries using the server's public key (line 2) and send them to the *Main-Client* (line 3, Fig. 13).

ALGORITHM 15: ***queryConstr* – query Construct**

INPUT: $\boldsymbol{S = \{X^{(i)}\}_{i=1}^{q}, c, f}$

$\boldsymbol{S = \{X^{(i)}\}_{i=1}^{q}}$: the set of q user queries, $\boldsymbol{q = \frac{N}{c\cdot(f+1)}}$

$\boldsymbol{c, f}$: the number of **c**lasses and **f**eatures, respectively

OUTPUT: $\boldsymbol{\{X_c[i]\}_{i=1}^{q}}$

$\boldsymbol{\{X_c[i]\}_{i=1}^{q}}$: encryption of the q input user queries (Fig.13)

Client(s):

1 **for** $\boldsymbol{i = 0}$ **to** $\boldsymbol{q}$
2 $\quad \boldsymbol{X_c[i] = EncEncr(X^{(i)})}$
3 $\quad$ ***send X_c[i] to the Main − Client***

6.1. Secure ML client centric classifications

Algorithm 16 (*clientComp*$(\cdot)$), as an input takes the row-wise encrypted trained model M^R_c, q client encrypted queries $\{X_c[i]\}_{i=1}^{q}$ and the number of classes and features, c and f, respectively (Section 4.1). All encryptions here are done using the server's public key (Fig. 1). For the DNN case the trained model M is representd by the row-wise enrypted weights for each of the inner L layers (Section 4.1.5), hence $M = \left\{M^R_c[l]\right\}_{l=1}^{L}$, where $M^R_c[l] = EncEncr(M^l)$ for $1 \le l \le L$. As illustrated in Fig. 13, the *Main-Client* rotates each of the $X_c[i]$ encrypted q queries in a way that when they are homomorphically added after the rotation, they form the column-wise encrypted queries' matrix S^C_c (Section 4.1.6, Fig. 4).

In S^C_c, client i has its designated query portion encoded starting from slot $i \cdot (f+1)$ till before slot $(i+1)\cdot(f+1)$, for $0 \le i \le q$(-lines 1–3, Fig. 13). In a scenario which is the same as the one described in Section 4.1.6 (Fig. 4), the *Main-Client* proceeds with *secMatMat*$(\cdot)$ to multiply M^R_c with S^C_c to get the encrypted classifications' matrix $C_M(S)_c$ for all of the q users, as it is shown in Eq. (4) in Section 4 (line 4). For the DNN case, as required by Eq. (5), instead of line 4, lines 4–1 till 4–7 are executed, where d_l is the dimension (number of neurons) of Layer l, for $1 \le l \le L$. Algorithm

secMatMat$(\cdot)$ (Algorithm 6, Section 5.1.6), which in turn calls *secDRep*$(\cdot)$, replicates each client's slot portion for c times, as shown in Fig. 14. Now, client i has its classification slot portion encoded starting from slot $i \cdot c \cdot (f+1)$ till slot $(i+1) \cdot c \cdot (f+1)$ at $C_M(S)_c$, for $0 \leq i \leq q$. Furthermore, each client's portion in $C_M(S)_c$ is sparsely encoded with f empty slots in-between each of the portion's c data slots. Those represent the c class values of the corresponding classification (e.g. posterior class probabilities for NB and MNB). This is precisely how the input of *SRCPer*$(\cdot)$ should be encoded and encrypted. Proposed in (Kjamilji et al., 2021a);*SRCPer*$(\cdot)$ simultaneously (in SIMD fashion) and homomorphically permutes the sparsely encoded c data slots for each client within the clients's classifiaction portion of $C_M(S)_c$ using as an input the random permutation vector k to get $rndC_M(S)_c$, as it is illustrated in Fig. 14 (line 5). In line 6 the *Main-Client* calls Algorithm *SCADS*$(\cdot)$ (Secure Comparison of All Data Slots, proposed in (Kjamilji et al., 2021a), which securely compares all of the c permuted data slots with each other and sends the encrypted comparison results to the server, and sends the permutaton vector k to each of the $q$clients (lines 7, 8). The server decrypts&decodes the ciphertext containing the comparisons of the permuted data slots and to each client sends the index of the maximum of the permuted data slots of its query portion - $rndMaxC_M(S)_v[i]$, where (lines 9–11). Each client than uses the vector k to find the original position of the permuted maximum, which is actuallythe client's final classification.

ALGORITHM 16: ***clientComp* – client Computation**

INPUT: $\boldsymbol{M^R_c, \{X_c[i]\}_{i=1}^{q}, c, f}$ //$\boldsymbol{\{M^R_c[l]\}_{l=1}^{L}}$
$\boldsymbol{M^R_c}$: row-wise encrypted Trained Model for NB, MNB, SVM, LR
//$\boldsymbol{\{M^R_c[l]\}_{l=1}^{L}}$: L row-encoded weight matrices for a DNN
$\boldsymbol{\{X_c[i]\}_{i=1}^{q}}$,: encryption of the q input queries, $q = \frac{N}{c \cdot (f+1)}$
$\boldsymbol{c, f}$: the number of classes and features, respectively

OUTPUT: $\boldsymbol{C_M(S)_v}$
$\boldsymbol{C_M(S)_v}$: vector of the final classification of q queries

Main – Client:
1 **for** $i = 0$ **to** q
2 $\quad \boldsymbol{X_c[i] = Rotate\left(qVec_c[i], i \times (f+1)\right)}$
3 $\boldsymbol{S^C_c = AddMany\left(\{X_c[i]\}_{i=1}^{q}\right)}$
4 $\boldsymbol{C_M(S)_c = secMatMat(M^R_c, S^C_c, c, f+1, q)}$
5 $\boldsymbol{rndC_M(S)_c = SRCPer(C_M(S)_c, k, R_1, R_2, c, f+1)}$
6 $\boldsymbol{rndC_M(S)_c = SCADS(\, rndC_M(S)_c, c, f+1)}$
7 **send** $\boldsymbol{rndC_M(S)_c}$ **to Server**
8 **send** $\boldsymbol{k}$ **to all of the** $\boldsymbol{q}$ **clients**

Server:
9 $\boldsymbol{rndC_M(S)_v = DecrDec(rndC_M(S)_c)}$
10 **for** $i = 0$ **to** q
11 $\quad$ **send** $\boldsymbol{rndMaxC_M(S)_v[i]}$ **portion to the** $\boldsymbol{i-th}$ **Client**

Client(s):
12 **for** $i = 0$ **to** q
13 $\quad \boldsymbol{C_M(S)_v[i] = invSRCPer(rndMaxC_M(S)_v[i], k, R_1R_2, c, f+1)}$

---------for the DNN case line 4 is substituted with: ---------
4-1 $\boldsymbol{S^C_c[1] = S^C_c, d_1 = f}$
4-2 **for** $l = 1$ **to** L
4-3 $\quad \boldsymbol{S_c[l+1] = secMatMatCas(M^R_c[l], S^C_c[i], d_{l-1}, d_l, q)}$
4-4 $\quad \boldsymbol{S_c[l+1] = f^l(S_c[l+1])}$
4-5 $\boldsymbol{S_c[L] = secMatMat(M^R_c[L], S^C_c[L], d_{L-1}, d_L, q)}$
4-6 $\boldsymbol{S_c[L] = f^l(S_c[L])}$
4-7 $\boldsymbol{C_M(S)_c = S_c[L], f = d_L}$

Finally, the proposed client centric secure ML classification Algorithm - *secMLClass* $-$ *CC*$(\cdot)$ (Algorithm 17) simply calls queryConstr$(\cdot)$ and *clientComp*$(\cdot)$ sequentially in that order (lines 1–2).

Improvement 2: if feasible (clients trust each-other or the queries are from the same client), the column-wise packing and dimension replication of the queries' matrix can be done in plain (un-encrypted) to get S^C_p, thus avoiding the costly homomorphic replications. Similarly, the replication of the trained model can be done before encrypting it or only by executing *CRep*$(\cdot)$ over it once to get M^R_c. Hence, it can be amortized among several multi-query classifications. In this sense, *queryConstruct*$(\cdot)$ and lines 1–4 of *clientComp*$(\cdot)$ can be substituted by simply executing $secDot(M^R_c, S^C_p, f+1)$.

ALGORITHM 17: ***secMLClass-CC* - secure ML Classifications-Client Centric**

INPUT: $\boldsymbol{M^R_c, S = \{X^{(i)}\}_{i=1}^{q}, c, f}$
$\boldsymbol{M^R_c}$: row-wise encrypted Trained Model
$\boldsymbol{S = \{X^{(i)}\}_{i=1}^{q}}$,: the set of q user queries, $q = \frac{N}{c \cdot (f+1)}$
$\boldsymbol{c, f}$: the number of classes and features, respectively

OUTPUT: $\boldsymbol{C_M(S)_v}$
$\boldsymbol{C_M(S)_v}$: vector of the final classification of q queries

1 $\boldsymbol{\{X_c[i]\}_{i=1}^{q} = queryConstr\left(S = \{X^{(i)}\}_{i=1}^{q}, c, f\right)}$
2 $\boldsymbol{clientComp\left(M^R_c, \{X_c[i]\}_{i=1}^{q}, c, f\right)}$

6.2. Secure ML server centric classifications

In *serverComp*$(\cdot)$ (Algorithm 18) the trained model resides in plaintext at the server, while all the encryptions are done using the *Decr-Client*'s public key, which also has the decryption key (Fig. 2). The general idea and the executed operations of *serverComp*$(\cdot)$ are the same ones as in *serverComp*$(\cdot)$, with the main difference being that the bulk of the costly homomorphic operations are done at the server rather than at the client.

Table 4
Amortized Costs of *secMatVec* (In millisececonds).

Matrix dim.	Naive (Juvekar et al., 2018)	Diagonal (Juvekar et al., 2018)	Hybrid (Juvekar et al., 2018)	This*
1024×128	880.0	192.4	16.2	28.7
1024×16	110.3	192.4	7.8	2.79
128×16	77.4	25.4	5.3	1.9

*Multiplication of a plaintext matrix with a packed ciphertext vector.

ALGORITHM 18: *serverComp* – server Computation

INPUT: $M^R_p, \{X_c[i]\}_{i=1}^{q}, c, f$ //$\{M^R_p[l]\}_{l=1}^{L}$
M^R_p: row-wise encoded Trained Model for NB, MNB, SVM, LR
//$\{M^R_p[l]\}_{l=1}^{L}$: L row-encoded weight matrices for a DNN
$\{X_c[i]\}_{i=1}^{q}$,: encryption of the q input queries, $q = \frac{N}{c\cdot(f+1)}$
c, f: the number of classes and features, respectively

OUTPUT: $C_M(S)_v$
$C_M(S)_v$: vector of the final classification of q queries

Main – Client:
1 **for** $i = 0$ **to** q
2 $\quad X_c[i] = Rotate(qVec_c[i], i \times (f+1))$
3 $S^C_c = AddMany(\{X_c[i]\}_{i=1}^{q})$
4 **send** S^C_c **to Server**

Server:
5 $C_M(S)_c = secMatMat(M^R_p, S^C_c, c, f+1, q)$
6 $rndC_M(S)_c = SRCPer(C_M(S)_c, k, R_1, R_2, c, f+1)$
7 $rndC_M(S)_c = SCADS(rndC_M(S)_c, c, f+1)$
8 **send** $rndC_M(S)_c$ **to** Decr – Client
9 **send** $k, R_1, R_2,$ **to all of the** q **clients**

Decr – Client (not same as the Main – Client):
10 $rndC_M(S)_v = DecrDec(rndC_M(S)_c)$
11 **for** $i = 0$ **to** q
12 $\quad$ **send** $rndMaxC_M(S)_v[i]$ **portion to the i – th Client**

Client(s):
13 **for** $i = 0$ **to** q
14 $\quad C_M(S)_v[i] = invSRCPer(rndMaxC_M(S)_v[i], k, R_1R_2, c, f+1)$

---------for the DNN case line 5 is substituted with: ---------
5-1 $S^C_c[1] = S^C_c, d_1 = f$
5-2 **for** $l = 1$ **to** L
5-3 $\quad S_c[l+1] = secMatMatCas(M^R_p[l], S^C_c[i], d_{l-1}, d_l, q)$
5-4 $\quad S_c[l+1] = f^l(S_c[l+1])$
5-5 $S_c[L] = secMatMat(M^R_p[L], S^C_c[L], d_{L-1}, d_L, q)$
5-6 $S_c[L] = f^l(S_c[L])$
5-7 $C_M(S)_c = S_c[L], f = d_L$

ALGORITHM 19: *secMLClass-SC* - secure ML Classifications-Server Centric

INPUT: $M^R_p, S = \{X^{(i)}\}_{i=1}^{q}, c, f$
M^R_p: row-wise encoded Trained Model plaintext
$S = \{X^{(i)}\}_{i=1}^{q}$,: the set of q user queries, $q = \frac{N}{c\cdot(f+1)}$
c, f: the number of classes and features, respectively

OUTPUT: $C_M(S)_v$
$C_M(S)_v$: vector of the final classification of q queries

1 $\{X_c[i]\}_{i=1}^{q} = queryConstr\left(S = \{X^{(i)}\}_{i=1}^{q}, c, f\right)$
2 $serverComp(M^R_c, \{X_c[i]\}_{i=1}^{q}, c, f)$

The proposed server centric secure ML classification Algorithm - *secMLClass – SC*(·) (Algorithm 19) simply calls *queryConstr*(·) and *serverComp*(·) sequentially.

Improvement 2 also holds for *secMLClass – SC*(·).

Table 5
Accuracy and amortized (per query) costs for different ML classifiers over different datasets for algorithms 17 and 19.

Client centric, N = 8192		Server centric, N = 8192		Client centric, N = 16384		Server centric, N = 16384		Data-set	MLclassifier	Acc. (%)
Comp. (μs)	Comm. (KB)	Comp. (μs)	Comm. (KB)	Comp. (μs)	Comm. (KB)	Comp. (μs)	Comm. (KB)			
242	1.72	287	3.4	279	1.72	340	3.4	https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(original)	NB	97.5
19	0.2	21	0.4	26.8	0.2	25.4	0.4	https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(original)	SVM; LR	96.8
1559	3.4	1748	6.8	1871	3.4	1505	6.8	https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(original)	DNN	98.5
17.6	0.2	18	0.4	20.6	0.2	23.9	0.4	https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/	SVM; LR	78.5
3190	13.75	3650	27.5	3260	13.75	3470	27.5	(SMS Spam Collection, 2019)	NB	95.6
7793	27.5	9050	55	7935	27.5	8704	55	(Vangelis et al., 2006)	MNB	99.1

*A single processor core is used for the results of the proposed schemes given in Tables 5–10.

7. Dealing with Malıcous clients

It is assumed that the server, the *Main-Client* and the *Decr-Client* (for $secMLClass - SC(\cdot)$) always remain in the semi-honest model, but any other client can be malicious. A malicious client that deviates from the protocol might fill up its own query portion of $f+1$ slots (Fig. 13) in a way that it will enable the malicious client to retrieve some info fully or partially about the trained model. In (Kjamilji et al., 2021a) they have dealt with such clients for the NB case, and the same technique can be easily extended and adopted to deal with the SVM, LR and DNN classifiers as well.

This paper deals with the case when malicious clients deviate from the protocol by putting random numbers in slots outside of their designated query portion with the aim of disrupting the correctness of the final classification for the other clients in the system. This attack is easily executed due to packing all the q client queries in a single ciphertext in both $secMLClass - CC(\cdot)$ and $secMLClass - SC(\cdot)$, as it is illustrated in Fig. 13.

According to (Goldreich, 2009)**;** malicious behavior is detected using protocols proven to be secure under the semi-honest model as their building block in a scenario where a subset of participants is semi-honest. Also, while detecting malicious behavior, the query privacy of the malicious client(s) should not be jeopardized. To this end, in $malicClientDet - CC(\cdot)$ (Algorithm 20), in order to keep the query privacy of the malicious client(s), the *Main-Client* multiplies each received client query with a plaintext mask which has zeros at the first $f+1$ slots and ones elsewhere (Fig. 15a) and sends the result to the server (lines 1, 2). The server decrypts the results for each query and sends an abortion signal to the *Main-Client* for the clients whose decrypted result doesn't have zeros in all slots, to exclude that client from the protocol. Thus, the proposed client centric secure ML classification over malicious clients Algorithm - $secMLClass - CC - MU(\cdot)$ (Algorithm 21) simply calls queryConstr$(\cdot)$, $malicClientDet - CC(\cdot)$ and $clientComp(\cdot)$ sequentially.

Table 6
Amortized query costs for the PP NB classifier.

Scheme	Cost	
	Computation cost	Communication cost
Original Wisconsin Breast Cancer Dataset (https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(original))		
(Bost, 2015)	479 ms	72.5 KB
(Gao et al., 2018)	555 ms	19.3 KB
(Li et al., 2018)	2900 ms	800 KB
(Liu et al., 2017)	349 min	1.24 MB
(Sun et al., 2018)	48.79 ms	Not reported
(Park et al., 2018)	70 s	2 ciphertexts
(Kjamilji et al., 2020)	14 ms	109 KB
(Pereira, 2020)	35.74 ms	43.13 KB
(Zhao and Xia, 2023)	2.1. milliseconds	Not reported
(Chen et al., 2021)	34 ms	890 KB
This paper	0.24 ms	1.7 KB
SMS spam dataset (SMS Spam Collection, 2019)		
(Kjamilji et al., 2021a)	6.75 ms	109 KB
(Kjamilji et al., 2021c)	3.38 ms	13.75 KB
(Vu et al., 2022)	28.350 s	Not reported
(Yang et al., 2005)	4212 s	Not reported
(Hao et al., 2014)	99.77 s	Not reported
(Vu et al., 2020)	99.85 s	Not reported
(Resende et al., 2022)	21 ms	Not reported
This paper	3.19 ms	13.75 KB

Table 7
Amortized query Costs for the PP SVM classifier.

Scheme	Cost	
	Computation cost	Communication cost
Original Wisconsin Breast Cancer Dataset (https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(original))		
(Bost, 2015)	204 ms	35.84 KB
(Gao et al., 2018)	3100 ms	7.5 MB
(Sun et al., 2018)	2.41 ms	Not reported
(De Cock et al., 2017)	3.47 ms	0.92 KB
(Rahulamathavan et al., 2013a)	7710 ms	1.394 MB
(Rahulamathavan et al., 2013b)	41,000 ms	0.174 MB
(Smahi, 2020)	900 s	Not reported
(Liang et al., 2021)	2.09 ms	1.35 KB
(Liang et al., 2020)	7.5 ms	0.34 KB
(Zhu et al., 2016)	560 ms	0.59 KB
(Zhang et al., 2020)	5, 32 s	Not reported
This paper	0.02 ms	0.2 KB
UNSW-NB 15 cybersecurity dataset (https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/)		
This paper	0.02 ms	0.2 KB

Table 8
Amortized query Costs for the PP LR Classifier.

Scheme	Cost	
	Computation cost	Communication cost
Original Wisconsin Breast Cancer Dataset (https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(original))		
(Bost, 2015)	204 ms	35.84 KB
(De Cock et al., 2017)	3.55 ms	0.92 KB
(Guo et al., 2018)	2200 ms	2.5 KB
(Zhou et al., 2023)	2.48 s	Not reported
This paper	0.02 ms	0.2 KB
UNSW-NB 15 cybersecurity dataset (https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/)		
(Guo et al., 2018)	12,200 ms	4 KB
This paper	0.02 ms	0.2 KB

Table 9
Amortized query Costs for the PP MNB Classifier.

Scheme	Cost	
	Computation cost	Communication cost
Enron email dataset (Vangelis et al., 2006)		
(Gupta et al., 2017)	350 ms	110 KB
(Kjamilji et al., 2021c)	15.31 ms	27.5 KB
(Khedr et al., 2015)	≈ 7.9 s	Not reported
(Costantino et al., 2017)	78 min	Not reported
This paper	7.7 ms	27.5 KB

Table 10
Amortized query Costs for the PP DNN Classifier.

Scheme	Cost	
	Computation cost	Communication cost
Original Wisconsin Breast Cancer Dataset (https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(original))		
This paper	1.56 ms	3.4 KB

ALGORITHM 20: ***malicClientDet-CC*** **– Malicious Client Detections – Client Centric**

INPUT: $\{X_c[i]\}_{i=1}^{q}, c, f$

$\{X_c[i]\}_{i=1}^{q}$,: encryption of the q input queries, $q = \frac{N}{c\cdot(f+1)}$

c, f: the number of classes and features, respectively

OUTPUT: $\{X_c[i]\}_{i=1}^{q'}$:

$\{X_c[i]\}_{i=1}^{q'}$: encrypted q' queries of clients that follow the protocol

Main – Client:

for $i = 0$ **to** q

  $tmpX_c[i] = X_c[i] \times maskMalicClients_p(f)$

  send $tmpX_c[i]$ **to Server**

Server:

for $i = 0$ **to** q

  $tmpX_v[i] = DecrDec(tmpX_c[i])$

  if $notAllZeros(tmpX_v[i])$

    send $abort_query(i)$ **to** $Main - Client$

ALGORITHM 21: ***secMLClass-CC-MU***
secMLClass-CC with malicius users

INPUT: $M^R_c, S = \{X^{(i)}\}_{i=1}^{q}, c, f$

M^R_c: row-wise encrypted Trained Model ciphertext

$S = \{X^{(i)}\}_{i=1}^{q}$,: the set of q user queries, $q = \frac{N}{c\cdot(f+1)}$

c, f: the number of classes and features, respectively

OUTPUT: $C_M(S)_v$

$C_M(S)_v$: vector of the final classification of q queries

$\{X_c[i]\}_{i=1}^{q} = queryConstruct\left(S = \{X^{(i)}\}_{i=1}^{q}, c, f\right)$

$\{X_c[i]\}_{i=1}^{q'} = malicClientDet\left(\{X_c[i]\}_{i=1}^{q}, c, f\right)$

$clientComp\left(M^R_c, \{X_c[i]\}_{i=1}^{q'}, c, f\right)$

Improvement 3. The improved algorithm is given in $secMLClass - CC - MU_v2(\cdot)$ (Algorithm 22). To avoid the extra communication round introduced by $malicClientDet - CC(\cdot)$ while detecting the malicious clients, the *Main-Client* instead simply multiplies each client query with a plaintext mask which has ones at the first $f+1$ slots and zeros elsewhere (Fig. 15b)). In this way the potential malicious behavior of the clients is avoided, but without actually detecting the malicious clients.

ALGORITHM 22: ***secMLClass-CC-MU_v2 secMLClass-CC with malicius users – version 2***

INPUT: $M^R_c, S = \{X^{(i)}\}_{i=1}^{q}, c, f$

M^R_c: row-wise encrypted Trained Model ciphertext

$S = \{X^{(i)}\}_{i=1}^{q}$,: the set of q user queries, $q = \frac{N}{c\cdot(f+1)}$

c, f: the number of classes and features, respectively

OUTPUT: $C_M(S)_v$

$C_M(S)_v$: vector of the final classification of q queries

$\{X_c[i]\}_{i=1}^{q} = queryConstruct\left(S = \{X^{(i)}\}_{i=1}^{q}, c, f\right)$

for $i = 0$ **to** q

  $tmpX_c[i] = X_c[i] \times invMaskMalicClients_p(f)$

$clientComp\left(M^R_c, \{tmpX_c[i]\}_{i=1}^{q}, c, f\right)$

Note 3: Algorithms 21 and 22 deal with malicious users during the client centric classifications (Section 6.1). Using the same logic, the corresponding versions of Algorithms 21 and 22, named as $secMLClass-SC-MU(\cdot)$ and $secMLClass-SC-MU_v2(\cdot)$, respectively, can be constructed, which deal with malicious clients for the server centric classifications (Section 6.2).

8. Experimental Evaluations and comparisons

For experimental evaluations, a machine of Intel(R) Core (TM) i5-8265U CPU with 4 Cores of 1.60 GHz and 16 GB of RAM running Windows 10 as its operating system is used. Unless otherwise stated, only a single processor core is used. For the SWHE operations the C++ based Microsoft's SEAL 3.2 library (https://github.com/microsoft/SEAL) is used, which implements a version of the FV scheme proposed in (Brakerski et al., 2014). For the polynomial degree N, plaintext coefficient modulus t and ciphertext coefficient modulus q, the values $N = 16384, t = 24bits, q = 438bits$, respectively, are used, which provide a security of 128-bits (Section 4.2). Roughly, this makes the security parameters and the used hardware resources similar to the state-of-the-art ones with which the results of this paper are compared with. All the used codes are uploaded to GitHub. C++ codes related to the novel secure linear algebra algorithms (Section 5) can be found in (https://github.com/artrimk/secLinAlg), while the C++ codes for the proposed PP ML classification algorithms (Section 6) can be found in (https://github.com/artrimk/secMLClass). Although when dealing with matrix–vector and matrix–matrix multiplication, in the proposed secure linear algebra algorithms one of the inputs can be a plaintext, which leads to faster results, to be fair with (Jiang et al., 2018), where both inputs are ciphertexts, it is also done the same while conducting the experiments for the corresponding algorithms of this paper. Furthermore, since in (Jiang et al., 2018) they deal with square matrices, in the experiments related to the secure linear algebra operations (Section 5.1), all the matrices are assumed to be square matrices with dimensions $d \times d$. Having this in mind, and knowing that in one ciphertext several matrices, vectors, etc. can be packed, as it was explained in the corresponding sections while introducing these algorithms, for the $secSum(\cdot)$ and $secDotP(\cdot)$ the number of packed units (vectors) is $q = \frac{N}{d}$. A single $d \times d$ matrix and a single ciphertext of length d that will be replicated for d times needs d^2 slots, thus for $secRep(\cdot)$, $secMatVec(\cdot)$ and $secMatVecCas(\cdot)$ in total $q = \frac{N}{d^2}$ units (vectors, matrices) can be packed. Finally, a $d \times d$ matrix whose dimension will be replicated for d times needs d^3 slots, thus for $secDRep(\cdot)$ and $secMatTDRep(\cdot)$, hence for $secMatMat(\cdot)$, $secMatMat_v2(\cdot)$, $secMatMatCas(\cdot)$ and $secMatMatCas_v2(\cdot)$ (which all call either $secDRep(\cdot)$ or $secMatTDRep(\cdot)$) $q = \frac{N}{d^3}$ units (matrices) can be packed. All the costs in Table 3 are amortized according to their corresponding number of packed units q that were just calculated. An important remark is that most of the time only a single processor core is used. Only at the cases when the matrix ciphertexts need to expand beyond the provided $N = 16384$ slots that a single ciphertext offers the other processor cores are utilized to deal with the newly created ciphertexts. In Table 3 such cases happen for $secDRep(\cdot)$ (hence $secMatMat(\cdot)$ also, which calls $secDRep(\cdot)$) when $d = 32 \text{and} d = 64$, since $d^3 = 32768$ and 262144, respectively, thus the provided $N = 16384$ slots are not enough.

In (Juvekar et al., 2018) they provide results only for the matrix–vector multiplication case with a plaintext matrix and an already packed ciphertext vector. Translated into the proposed secure operations proposed in Section 5, this can be reduced to a secure dot product ($secDotP(\cdot)$) between a ciphertext and a packed plaintext. Characteristic of the hybrid scheme in (Juvekar et al., 2018) is that it has a lot of offline efforts to set the scheme so it can utilize all the processor cores simultaneously, thus shows good results. To counter for that, the following is used: **1.** When using large enough N that fits all the data in a single ciphertext, instead of using one core it can be assumed that the same calculation can be done do at the other cores simultaneously and amortize the result by basically dividing with the number of cores. Since one might object to this approach, then **2.** Just like (Juvekar et al., 2018), the parameter N is set so to encode the data in multiple ciphertexts that can be run simultaneously over all the available cores. E.g. for the 1024×16 dimensions 16,384 slots are needed that can be fit in a single ciphertext with $N = 16384$, or in 4 ciphertexts with $N = 4096$ that are run on 4 cores simultaneously. At the first case an amortized cost of 3.01 ms is obtained, while for the second it is 2.79 ms. The same logic is followed for the other dimensions in Table 4.

To evaluate and compare the proposed secure and PP ML classification algorithms with the state-of-the-art ones the following benchmark datasets are used:

Wisconsin Breast Cancer Dataset (WBCD) (https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(original)) which has $c^{WBSC} = 2$ classes and 9 features, each with 10 distinct values. Thus, for SVM and LR, the number of features is $f^{WBCD}_{SVM,LR} = 9$ (Section 4.1.3 and 4.1.4), while for NB the number of features is $f^{WBCD}_{NB} = \sum_i |F_i| = 9 \cdot 10 = 90$ (Section 4.1.1). However, due to the proposed secure linear algebra operations, which require the matrix dimensions to be powers of two (Section 5), and also knowing that the trained model column dimension and the query matrix row dimension is $f + 1$ (Section 4, Fig. 4), the numbers of features used in the proposed algorithms is $f^{WBCD}_{SVM,LR} = 15$ and $f^{WBCD}_{NB} = 127$. The extra added slots are padded with zeros.

SMS Spam Collection Dataset (SMS Spam Collection, 2019). It has $c^{SMS} = 2$. The bigram approach proposed in (Kjamilji et al., 2021a) is used, which gives 441 features, each with 2 values, thus, for the reasons explained above, for NB the proposed PP ML schemes use $f^{WBCS}_{NB} = 1023$

Enron email dataset (Vangelis et al., 2006). It has $c^{Enron} = 2$ classes. Using the technique given in (Vangelis et al., 2006); $f^{Enron}_{MNB} = 2047$ features (bag-of-words) can be extracted for the MNB classifier.

UNSW-NB 15 cybersecurity dataset (https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/). It also has $c^{UNSW} = 2$ classes. Out of its 42 original features, for the SVM and LR classifier $f^{UNSW}_{SVM,LR} = 15$ features are chosen.

For all the datasets and all the proposed secure and PP ML classifier schemes no loss of classification accuracy was observed because of PP classification due to selecting the security parameters N, t and q properly (Section 4.2).

Table 5 gives the accuracies and the amortized (per query) computation and communication costs for both the PP ML client and the PP ML server centric classifications (Algorithms 17 and 19) for different ML classifiers and different security parameters (when $N = 8192$, $t = 17$ bits and $q = 218$ bits) over the above-mentioned datasets. Tables 6–10 compare the amortized (per query) computation and communication costs of the proposed schemes and the related state-of-the-art PP classifications for NB, MNB, SVM, LR and DNN among different datasets, respectively. If a certain scheme provides several results for the same secure classifier due to different security parameters or improved scenarios, Tables 6–10 provide the best reported results of the corresponding schemes. In all of them the implementation results which use Improvement

1 and 2 for both of the proposed secure ML classifiers (Algorithm 17 and 19) are reported. For all the datasets, the trained models for NB and MNB in plain were obtained using C++, while for SVM, LR and DNN the trained models were obtained by WEKA (https://old-www.cms.waikato.ac.nz/~ml/weka/). For the dataset in (https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(original)), the proposed DNN architecture has $L = 3$ layers, where the dimensions of the input (layer 0) and the two hidden layers are $d_0 = 16$, $d_1 = 16$ and $d_2 = 1$, respectively (Fig. 3). The activation function for both hidden layers is a simple squaring.

Note 4: A single ciphertext can encode and encrypt up to N integers (Section 4.2). For the conducted experiments and chosen datasets, as shown above, the total number of feature values over all feature ranges from a dozens(s) to a few hundreds. This is enables the encoding of not one, but multiple queries in a single ciphertext. Since a single ciphertext is not divisible, it cannot be executed in parallel among multiple cores or processors. Parallel execution is rather intrinsic for a single ciphertext due to the SIMD way of processing (Section 4.2). However, for high dimensional datasets with thousands of features, a single client query is encoded in multiple ciphertexts which can be processed in parallel among multiple processors or processor cores or GPUs and FGPAs. As a matter of fact, as mentioned above, while conducting experiments for $secDRep(\cdot)$ (hence $secMatMat(\cdot)$ also, which calls $secDRep(\cdot)$) when dimensions of the input square matrix are $d = 32$ and $d = 64$, multiple processor cores were used for parallel execution.

Note 5: While computationally not expensive, SWHE schemes allow for a limited number of homomorphic multiplications, which is known as the depth of the circuit (Section 4.2). While this is not a problem for the other PP ML classifiers which can be done in a single homomorphic matrix multiplication, there are limitations for PP DNN classifiers with high number of layers. This is due to the high number of cascading homomorphic matrix multiplication, one per each layer to be more precise. This can be countered by adding some random values to the ciphertext which reaches its circuit depth, decrypting it, then freshly re-encrypting it, removing (subtracting) the random added values, and continuing with a new series of cascading homomorphic multiplications. Or it also can be countered by using the costlier FHE (Brakerski et al., 2014; Fan and Vercauteren, 2012; Smart and Vercauteren, 2014) which offers unlimited number of homomorphic multiplications.

Note 6: While it is non-linear, due to loss of accuracy, squaring does not have the same effect on DNN as other commonly used activation functions do (such as sigmoid, leaky ReLU, etc.). Having also in mind the remarks in Note 5, the conducted experiments show that a DNN with up to 10 hidden layers with squaring activation functions can be used for accurate PP DNN classifications. To use a DNN with more than 10 hidden layers, then the commonly used activation functions should be used either by homomorphically evaluating their Taylor series or by utilizing other approaches which can be found in literature (Farokhi, 2020; Obla et al., 2020; Ishiyama et al., 2020).

9. Security analysis and proofs

Definition 2. Decision-RLWE (Ring Learning With Errors): for a security parameter λ, let $f(x)$ be a cyclotomic irreducible polynomial of degree N which depends on λ. Let $R_q = Z_q[x]/f(x)$ be a polynomial ring with integer coefficients modulo q, s.t. $q = q(\lambda) \geq 2$. Let a, c and s be uniform randomly chosen from R_q, i.e. $a \leftarrow R_q$, $c \leftarrow R_q$ and $s \leftarrow R_q$. Let e be a randomly chosen noise from a normal distribution χ, i.e., $e \leftarrow \chi$. Decision RLWE is the problem to distinguish between $(a,\ a \cdot s + e)$ and (a, c) (Brakerski et al., 2014; Fan and Vercauteren, 2012; Smart and Vercauteren, 2014; Kjamilji et al., 2021a).

Assumption 1. Hardness of Decision-RLWE: Decision-RLWE is assumed to be a hard and resilient problem even for an adversary with a quantum computer (Brakerski et al., 2014; Fan and Vercauteren, 2012; Smart and Vercauteren, 2014).

Semantic security of the RLWE schemes: Due to its probabilistic encryption, RLWE based schemes offer semantic security, i.e., for ciphertexts $m_0_c$and m_1_c that encrypt plaintexts $m_0_p$and m_1_p, an adversary cannot distinguish which ciphertext belongs to which plaintext (Goldreich, 2009).

Definition 3. semi-honest Secure Multi-Party Computation (SMPC) for deterministic functions (simplified version of the one given in (Goldreich, 2009)**:** There have p parties (entities) denoted as $P_1, P_2, \ldots, P_p$. Let the view of P_i, where $i = 1, 2, \ldots p$, while executing protocol (algorithms) Π be $V_{P_i}(\lambda, I_i) = (m_1^{P_i}, \ldots, m_m^{P_i},)$, where the inputs of the views are λ which is a security parameter and I_i which are the private inputs of P_i, while $m_1^{P_i}, \ldots, m_m^{P_i}$ are the messages that P_i gets while executing Π. Protocol Π is secure under the semi-honest model if there exists a simulator (function) $S_{P_i}(\lambda, O_i)$ that as an input takes λ and O_i such that $S_{P_i}(\lambda, O_i) \cong_c V_{P_i}(\lambda, I_i)$, for any I_i, where O_i is the output that P_i receives when Π is finished. The $\cong_c$ stands for computational indistinguishability against a probabilistic polynomial time adversary (Bost et al., 2015; Goldreich, 2009).

Theorem 1. ***Modular Sequential Composition Theorem (MSCT) (simplified):*** *Let $\Pi_1, \Pi_2, \ldots, \Pi_n$ be secure SMPC protocols as per* Definition 3. *Let Π be a protocol which sequentially calls $\Pi_1, \Pi_2, \ldots, \Pi_n$ Then Π is also a secure SMPC protocol.*

Proof: given in (Goldreich, 2009). ■

Theorem 2. *queryConstr (Algorithm 15) is an SMPC protocol under the semi-honest model.*

Proof: Throughout all the proofs, c and f are global public inputs and are not private. In this algorithm there are q clients and one *Main-Client*. The view of client i, where $i = 1, 2, \ldots, q$, is $V_i(\lambda, I_i) = \{\phi\}$, where $\lambda = \{N, q, t\}$ (as explained in Section 4.2) throughput the rest of the proofs, $I_i = X^{(i)}$ is the private input of client i and ϕ is the empty set, since none of the q clients get any message while executing the protocol. The simulator $S_i(\lambda, O_i) = \{\phi\}$ for client i is rather trivial, where $O_i = \phi$ is the output of client i (since none of the clients get any output) and $i = 1, 2, \ldots, q$. Thus, $S_i(\lambda, O_i) \cong_c V_i(\lambda, I_i)$.

The view of the *Main-Client* is $V_{MC}(\lambda, I_{MC}) = \left\{\{X_{c[i]}\}_{i=1}^{q}\right\}$, where $I_{MC} = \phi$. Let's construct q random ciphertexts $\left\{\tilde{X}_c[i]\right\}_{i=1}^{q}$. Due to the semantic security of the RLWE schemes, $\{X_{c[i]}\}_{i=1}^{q}$ cannot be distinguished from the corresponding $\left\{\tilde{X}_c[i]\right\}_{i=1}^{q}$. Let the simulator of the *Main-Client* be $S_{MC}(\lambda, O_{MC}) = \left\{\left\{\tilde{X}_c[i]\right\}_{i=1}^{q}\right\}$, where $O_{MC} = \left\{\left\{\tilde{X}_c[i]\right\}_{i=1}^{q}\right\}$ is the output of the *Main-Client*. Thus, $S_{MC}(\lambda, O_{MC}) \cong_c V_{MC}(\lambda, I_{MC})$ ■

Theorem 3. clientComp (Algorithm 16) is an SMPC protocol under the semi-honest model.

Proof: The view of q clients is $V_i(\lambda, I_i) = \{rndC_M(S)_v[i], k\}$, where their private input is $I_i = \phi$ and the output is $O_i = \{C_M(S)_v[i]\}$, for $i = 1, 2, \ldots, q$. Let their corresponding simulators be $S_i(\lambda, O_i) = \left\{\widetilde{rndC_M}(S)_v[i]\right\}$, where $\widetilde{rndC_M}(S)_v[i] = SRCPer(C_M(S)_v[i], k, 0, 0, c, f+1)$, since $C_M(S)_v[i] = invSRCPer(\widetilde{rndC_M}(S)_v[i], k, 0, 0, c, f+1)$, thus $S_i(\lambda, O_i) \cong_c V_i(\lambda, I_i)$.

The view of the *Main-Client* is $V_{MC}(\lambda, I_{MC}) = \{\phi\}$, where the private input of the *Main-Client* is $I_{MC} = \left\{\{X_c[i]\}_{i=1}^{q}, M^R_c\right\}$. Let the Main-Client's simulator be $S_{MC}(\lambda, O_{MC}) = \{\phi\}$, where $O_{MC} = \phi$ is the output of the *Main-Client*. $S_{MC}(\lambda, O_{MC}) \cong_c V_{MC}(\lambda, I_{MC})$.

The view of the Server is $V_S(\lambda, I_S) = \{rndC_M(S)_c\}$, where the private input is $I_S = \phi$ and the output is $O_S = \phi$. For the server's simulator $S_s(\lambda, O_S)$ a matrix of q random queries $\widetilde{S} = \left\{\widetilde{X^{(i)}}\right\}_{i=1}^{q}$ is constructed and used as an input to proceed with *queryConstr* (Algorithm 14) and then with lines 1–5 of *clientComp* (Algorithm 15) to finally get $\widetilde{rndC_M}(S)_c$. Since the server cannot distinguish between $rndC_M(S)_c$ and $\widetilde{rndC_M}(S)_c$ due to the semantic security of the underlying RLWE scheme (he can't even distinguish between the decrypted and decoded $rndC_M(S)_v$ and $\widetilde{rndC_M}(S)_v$), by having $S_s(\lambda, O_S) = \left\{\widetilde{rndC_M}(S)_c\right\}$ it is shown that $S_S(\lambda, O_S) \cong_c V_S(\lambda, I_S)$ ■

Theorem 4. *secMLClass-CC (Algorithm 17) is an SMPC protocol under the semi-honest model.*

Proof: *secMLClass-CC* sequentially calls *queryConstr* (Algorithm 15) and *clientComp* (Algorithm 16), which are both shown to be SMPC protocols under the semi-honest model. The MSCT is invoked to proof the security of *secMLClass-CC* ■

Theorem 5. *servComp (Algorithm 18) is an SMPC protocol under the semi-honest model.*

Proof: The view of q clients is $V_i(\lambda, I_i) = \{rndC_M(S)_v[i], k\}$, where their private input is $I_i = \phi$ and their output is $O_i = \{C_M(S)_v[i]\}$, for $i = 1, 2, \ldots, q$. Let their simulator be $S_i(\lambda, O_i) = \{rndC_M(S)_v[i]\}$, where $rndC_M(S)_v[i] = SRCPer(C_M(S)_v[i], k, 0, 0, c, f+1)$, since $C_M(S)_v[i] = invSRCPer\left(\widetilde{rndC_M}(S)_v[i], k, 0, 0, c, f+1\right)$, thus $S_i(\lambda, O_i) \cong_c V_i(\lambda, I_i)$.

The view of the *Main-Client* is $V_{MC}(\lambda, I_{MC}) = \{\phi\}$, where the private input of the Main-Client is $I_{MC} = \{\{X_c[i]\}_{i=1}^{q}\}$. Let the *Main-Client*'s simulator be $S_{MC}(\lambda, O_{MC}) = \{\phi\}$, where $O_{MC} = \phi$ is the output of the *Main-Client*. $S_{MC}(\lambda, O_{MC}) \cong_c V_{MC}(\lambda, I_{MC})$.

The view of the Server is $V_S(\lambda, I_S) = \left\{S^C_c\right\}$, where the private input is $I_S = M^R_p$ and the output is $O_S = \phi$. For the server's simulator $S_s(\lambda, O_S)$ a random ciphertext $\widetilde{S^C_c}$ is constructed which cannot be distinguished by S^C_c due to the semantic security of the RLWE schemes. By having $S_s(\lambda, O_S) = \left\{\widetilde{S^C_c}\right\}$ it is shown that $S_S(\lambda, O_S) \cong_c V_S(\lambda, I_S)$.

The view of the *Decr-Client* is $V_{DC}(\lambda, I_{DC}) = \{rndC_M(S)_c\}$, where the private input is $I_{DC} = \phi$ and the output is $O_{DC} = \phi$. Let the *Decr-Client*'s simulator $S_{DC}(\lambda, O_{DC})$ be a matrix of q random queries $\widetilde{S} = \left\{\widetilde{X^{(i)}}\right\}_{i=1}^{q}$ which is used as an input in *queryConstr* (Algorithm 14). Then it is proceeded with lines 1–7 of *servComp* (Algorithm 17) to finally get $\widetilde{rndC_M}(S)_c$. Since *Decr-Client* cannot distinguish between $rndC_M(S)_c$ and $\widetilde{rndC_M}(S)_c$ due to the semantic security of the underlying RLWE scheme (he can't even distinguish between the decrypted and decoded $rndC_M(S)_v$ and $\widetilde{rndC_M}(S)_v$), by having the *Decr-Client* simulator be $S_{DC}(\lambda, O_{DC}) = \widetilde{rndC_M}(S)_c$, it is shown that $S_{DC}(\lambda, O_{DC}) \cong_c V_{DC}(\lambda, I_{DC})$. ■

Theorem 6. *secMLClass-SC (Algorithm 19) is an SMPC protocol under the semi-honest model.*

Proof: *secMLClass-SC* sequentially calls *queryConstr* (Algorithm 15) and *servComp* (Algorithm 18), which are both shown to be SMPC under the semi-honest model. The MSCT (Theorem 1) is invoked to proof the security of *secMLClass-SC*. ■

Theorem 7. *malicClientDet-CC (Algorithm 20) is an SMPC protocol under the semi-honest model.*

Proof: The input and the output of the *Main-Client* are $I_{MC} = \{X_c[i]\}_{i=1}^{q}$ and $O_{MC} = \{X_c[i]\}_{i=1}^{q'}$, respectively. The view is $V_{MC}(\lambda, I_{MC}) = \{\{abort_query(i)\}_{i=1}^{q'}\}$, which are the q' aborted queries. For the simulator for the *Main-Client* the same aborted queries can be given, thus $S_{MC}(\lambda, O_{MC}) = \{\{abort_query(i)\}_{i=1}^{q'}\}$, so $S_{MC}(\lambda, O_{MC}) \cong_c V_{MC}(\lambda, I_{MC})$.

The input and the output of the Server are $I_S = \phi$ and $O_S = \{\{abort_query(i)\}_{i=1}^{q'}\}$, respectively. The view is $V_S(\lambda, I_S) = \{\{tmpX_c[i]\}_{i=1}^{q}\}$. Based on the output O_S, $\left\{\widetilde{tmpX_c[i]}\right\}_{i=1}^{q}$ can be constructed such as for the input queries that the Server gets that are aborted, the corresponding ciphertexts that have zeros on the portion of the slots that belong to the corresponding user is constructed, and the remaining slots are filled up with random values (as shown in Fig. 15a)). For the queries that are not aborted, ciphertexts with zeros on all slots are constructed. In this way, if $\left\{\widetilde{tmpX_c[i]}\right\}_{i=1}^{q}$ is used instead of $\{tmpX_c[i]\}_{i=1}^{q}$ while executing lines 4–7 of *malicClientDet-CC*, the same queries will be aborted. Thus, the Server simulator is $S_s(\lambda, O_S) = \left\{\left\{\widetilde{tmpX_c[i]}\right\}_{i=1}^{q}\right\}$, hence $S_s(\lambda, O_S) \cong_c V_S(\lambda, I_S)$. ■

Theorem 8. ***secMLClass-CC-MU*** *(Algorithm 21) is an SMPC protocol under the semi-honest model.*

Proof: *secMLClass-CC* sequentially calls *queryConstr* (Algorithm 15), *malicClientDet* (Algorithm 20) and *clientComp* (Algorithm 16), which are all shown to be SMPC protocol under the semi-honest model. The MSCT (Theorem 1) is invoked to proof the security of *secMLClass-CC-MU* ■

Corollary 1. If two till up to $q-1$ out of q clients (except the Main-client) collude in Algorithm 21, they cannot retrieve any data about the non-colluding clients or the trained model.

Corollary 2. The secMLClass-SC-MU and secMLClass-SC-MU_v2 as described in *Section 8*, are proven to be SMPC protocols under the semi-honest model by following the same lines as in Theorem 8 ■

Theorem 9. *secMLClass-CC-MU_v2 (Algorithm 22) is an SMPC protocol under the semi-honest model.*

Proof: *secMLClass-CC-MU_v2* sequentially calls *queryConstr* (Algorithm 15) and *clientComp* (Algorithm 16), which are both shown to be SMPC protocols under the semi-honest model. The MSCT (Theorem 1) is invoked to proof the security of *secMLClass-CC-MU_v2* ■

10. Discussions and future directions

This paper proposes a novel set of secure linear algebra operations, such as inner (dot product), a couple of flavors of matrix–vector product, a few versions of matrix–matrix product, matrix transpose etc. On top of them, under strict security, privacy and efficiency constraints, secure and PP ML classifiers which can be expressed in terms of linear algebra operations such as DNN, NB, MNB, SVM and LR are built. Theoretical analysis and extensive experimental evaluations over benchmark datasets show that the proposed secure linear algebra operations, hence the secure and PP ML classifiers build on top of them, outperform the state-of-the-art schemes in terms of computation and communication costs as well as on security and privacy characteristics. Among others, the proposed secure and PP ML classifiers deal with malicious clients (one of the rare schemes to do so), are non-interactive, are flexible in terms of where the bulk of operations are executed (server vs. client centric), etc. Since the underlying cryptographic primitives are shown to be resilient to quantum computer attacks, the proposed secure and PP ML classifiers are also suitable for the post-quantum world. It should be noted that the proposed secure linear algebra operations can be used in any scenario where linear algebra operations are required to be done in secure and private fashion. Thus, the usage of the proposed secure linear algebra operations is not limited to PP ML classifications.

In the future the authors plan to extend those efficiency, security, and privacy characteristics for other ML classifiers such as KNN, decision trees (DT), random forests (RF), ensemble of classifiers, etc., which cannot be explicitly expressed by the proposed linear algebra operations.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Aggarwal, C.C., Philip, S.Y., 2008. A general survey of privacy-preserving data mining models and algorithms. Privacy-Preserv. Data Mining: Models Algorithms, 11–52.

Agrawal, Rakesh, Ramakrishnan Srikant, 2000. Privacy-preserving data mining. In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data.

Bost, R. et al., 2015. Machine learning classification over encrypted data. NDSS. 4324.

Brakerski, Z., Gentry, C., Vaikuntanathan, V., 2014. (Leveled) fully homomorphic encryption without bootstrapping. ACM Trans. Comput. Theory (TOCT) 6 (3), 1–36.

Chen, J. et al., 2021. Non-interactive Privacy-Preserving Naïve Bayes Classifier Using Homomorphic Encryption. International Conference on Security and Privacy in New Computing Environments. Springer International Publishing, Cham.

Chen, P.-H., Liu, Y., Peng, L., 2019. How to develop machine learning models for healthcare. Nat. Mater. 18 (5), 410–414.

Cheon, J.H. et al., 2018. Ensemble method for privacy-preserving logistic regression based on homomorphic encryption. IEEE Access 6, 46938–46948.

Costantino, G. et al., 2017. Privacy-preserving text mining as a service. 2017 IEEE Symposium on Computers and Communications (ISCC). IEEE.

De Cock, M. et al., 2017. Efficient and private scoring of decision trees, support vector machines and logistic regression models based on pre-computation. IEEE Trans. Dependable Secure Comput. 16 (2), 217–230.

Duy, H.V., 2022. An efficient solution for privacy-preserving naïve bayes classification in fully distributed data model. J. Sci. Technol. Informat. Sec. 1 (15), 56–61.

Edemacu, K., Kim, J.W., 2021. Multi-party privacy-preserving logistic regression with poor quality data filtering for IoT contributors. Electronics 10 (17), 2049.

Fan, Y. et al., 2020. Privacy preserving based logistic regression on big data. J. Network Comput. Appl. 171, 102769.

Fan, J., Frederik V., 2012. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive

Farokhi, F., 2020. Privacy-preserving public release of datasets for support vector machine classification. IEEE Trans. Big Data 7 (5), 893–899.

Gao, C.-Z. et al., 2018. Privacy-preserving Naive Bayes classifiers secure against the substitution-then-comparison attack. Infr. Sci. 444, 72–88.

Ghavamipour, A.R., Turkmen, F., Jiang, X., 2022. Privacy-preserving logistic regression with secret sharing. BMC Med. Informa. Decis. Mak. 22 (1), 1–11.

Goldreich, O., 2009. Foundations of Cryptography: Volume 2, Basic Applications. Cambridge University Press.

Guo, W. et al., 2018. A privacy-preserving online medical prediagnosis scheme for cloud environment. IEEE Access 6, 48946–48957.

Gupta, T., et al. 20107. Pretzel: Email encryption and provider-supplied functions are compatible. In: Proceedings of the Conference of the ACM Special Interest Group on Data Communication.

Hao, F, et al., 2014. Every Vote Counts: Ensuring Integrity in {Large-Scale} Electronic Voting. In: 2014 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (EVT/WOTE 14).

Ishiyama, T., Suzuki, T., Yamana, H., 2020. Highly accurate CNN inference using approximate activation functions over homomorphic encryption. 2020 IEEE International Conference on Big Data (Big Data). IEEE.

Jiang, X., et al., 2018. Secure outsourced matrix computation and application to neural networks. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security

Juvekar, Chiraag, Vinod Vaikuntanathan, Anantha Chandrakasan, 2018. {GAZELLE}: A low latency framework for secure neural network inference.In: 27th USENIX Security Symposium (USENIX Security 18).

Khedr, A., Gulak, G., Vaikuntanathan, V., 2015. SHIELD: scalable homomorphic implementation of encrypted data-classifiers. IEEE Tran. Comput. 65 (9), 2848–2858.

Kjamilji, A., et al., 2020. Secure Naïve Bayes classification without loss of accuracy with application to breast cancer prediction. In: Proceeding International Conference on Science and Engineering. vol. 3

Kjamilji, A., et al., 2021. Secure matrix operations for machine learning classifications over encrypted data in post quantum industrial IoT. In: 2021 International Symposium on Networks, Computers and Communications (ISNCC). IEEE.

Kjamilji, Artrim, Albert Levi, Erkay Savaş. 2021. Blockchain assisted secure feature selection, training and classifications in cloud and distributed edge IoT environments.

Kjamilji, A., Savaş, E., Levi, A., 2021a. Efficient secure building blocks with application to privacy preserving machine learning algorithms. IEEE Access 9, 8324–8353.

Kjamilji, A. 2021. Blockchain driven secure and private machine learning algorithms for post-quantum 5G/6G enabled industrial IoT with applications to cyber-security and health. Diss. 2021.

Leo, M., Sharma, S., Maddulety, K., 2019. Machine learning in banking risk management: A literature review. Risks 7 (1), 29.

Li, T. et al., 2018. Outsourced privacy-preserving classification service over encrypted data. J. Network Comput. Appl. 106, 100–110.

Liang, J. et al., 2020. Practical and secure SVM classification for cloud-based remote clinical decision services. IEEE Trans. Comput. 70 (10), 1612–1625.

Liang, J. et al., 2021. Verifiable and secure svm classification for cloud-based health monitoring services. IEEE Internet Things J. 8 (23), 17029–17042.

Lindell, Y., Benny, P., 2000. Privacy preserving data mining. Annual International Cryptology Conference. Springer Berlin Heidelberg, Berlin, Heidelberg.

Liu, X. et al., 2015. Privacy-preserving patient-centric clinical decision support system on naive Bayesian classification. IEEE J. Biomed. Health Informat. 20 (2), 655–668.

Liu, X. et al., 2017. Privacy-preserving outsourced clinical decision support system in the cloud. IEEE Trans. Serv. Comput. 14 (1), 222–234.

Liu, X. et al., 2018. Privacy-preserving outsourced support vector machine design for secure drug discovery. IEEE Trans. Cloud Comput. 8 (2), 610–622.

Obla, S. et al., 2020. Effective activation functions for homomorphic evaluation of deep neural networks. IEEE Access 8, 153098–153112.

Park, H. et al., 2018. Efficient machine learning over encrypted data with non-interactive communication. Computer Standards & Interfaces 58, 87–108.

Pereira, H.V.L., 2020. Efficient AGCD-based homomorphic encryption for matrix and vector arithmetic. International Conference on Applied Cryptography and Network Security. Springer International Publishing, Cham.

Rahulamathavan, Y. et al., 2013a. Privacy-preserving multi-class support vector machine for outsourcing the data classification in cloud. IEEE Trans. Dependable Secure Comput. 11 (5), 467–479.

Rahulamathavan, Y. et al., 2013b. Privacy-preserving clinical decision support system using gaussian kernel-based classification. IEEE J. Biomed. Health Informat. 18 (1), 56–66.

Resende, A. et al., 2022. Fast privacy-preserving text classification based on secure multiparty computation. IEEE Trans. Inf. Forensics Secur. 17, 428–442.
Shaukat, K. et al., 2020. A survey on machine learning techniques for cyber security in the last decade. IEEE Access 8, 222310–222354.
Smahi, A. et al., 2020. A blockchainized privacy-preserving support vector machine classification on mobile crowd sensed data. Pervasive Mob. Comput. 66, 101195.
Smart, N.P., Vercauteren, F., 2014. Fully homomorphic SIMD operations. Des. Codes Crypt. 71, 57–81.
SMS Spam Collection v.1 Data Set. Accessed: Jun. 2019. [Online]. Available: http://www.dt.fee.unicamp.br/~tiago/smsspamcollection.
Sun, X. et al., 2018. "Private machine learning classification based on fully homomorphic encryption." IEEE Transactions on Emerging Topics. Computing 8 (2), 352–364.
Vangelis, M., Ion, A., Geogios, P., 2006. Spam filtering with naive bayes-which naive bayes? Third Conference on Email and Anti-spam (CEAS).
Vu, D.-H., 2022. Privacy-preserving Naive Bayes classification in semi-fully distributed data model. Comput. Secur. 115, 102630.
Vu, D.-H., Luong, T.-D., Ho, T.-B., 2020. An efficient approach for secure multi-party computation without authenticated channel. Inf. Sci. 527, 356–368.
Vu, D.-H., Trong-Sinh, V., Luong, T.-D., 2022. An efficient and practical approach for privacy-preserving Naive Bayes classification. J. Infr. Sec. Appl. 68, 103215.
Yang, Z., Zhong, S., Wright, R.N., 2005. Privacy-preserving classification of customer data without loss of accuracy. Proceedings of the 2005 SIAM International Conference on Data Mining. Society for Industrial and Applied Mathematics.
Yi, X., Zhang, Y., 2009. Privacy-preserving naive Bayes classification on distributed data via semi-trusted mixers. Inf. Syst. 34 (3), 371–380.
Zhang, M., Song, W., Zhang, J., 2020. A secure clinical diagnosis with privacy-preserving multiclass support vector machine in clouds. IEEE Syst. J. 16 (1), 67–78.
Zhao, X., Xia, Z., 2023. Secure outsourced NB: Accurate and efficient privacy-preserving Naive Bayes classification. Comput. Secur. 124, 103011.
Zhou, Y. et al., 2023. A privacy-preserving logistic regression-based diagnosis scheme for digital healthcare. Futur. Gener. Comput. Syst. 144, 63–73.
Zhu, H. et al., 2016. Efficient and privacy-preserving online medical prediagnosis framework using nonlinear SVM. IEEE J. Biomed. Health Informat. 21 (3), 838–850.