BLOCKCHAIN DRIVEN SECURE AND PRIVATE MACHINE LEARNING ALGORITHMS FOR POST-QUANTUM 5G/6G ENABLED INDUSTRIAL IoT WITH APPLICATIONS TO CYBER-SECURITY AND HEALTH

by

ARTRIM KJAMILJI

Submitted to the Institute of Engineering and Natural Sciences
in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

SABANCI UNIVERSITY
JULY 2021

BLOCKCHAIN DRIVEN SECURE AND PRIVATE MACHINE LEARNING ALGORITHMS FOR POST-QUANTUM 5G/6G ENABLED INDUSTRIAL IoT WITH APPLICATIONS TO CYBER-SECURITY AND HEALTH

APPROVED BY:

[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]

DATE OF APPROVAL: ....

ABSTRACT

BLOCKCHAIN DRIVEN SECURE AND PRIVATE MACHINE LEARNING ALGORITHMS FOR POST-QUANTUM 5G/6G ENABLED INDUSTRIAL IoT WITH APPLICATIONS TO CYBER-SECURITY AND HEALTH

ARTRIM KJAMILJI
Ph.D. Dissertation, July 2021
Supervisor: Prof. Albert Levi
Co-Supervisor: Prof. Erkay Savaş

Keywords: blockchain; multi-label multi-output machine learning algorithms; secure IoT; privacy preserving; feature selection; training; classification; homomorphic encryption; collusion attacks; distributed environments; cyber-security; Internet of Medical Things

We provide a general framework for secure and private multi-label multi-output machine learning (ML) algorithms for the semi-honest model in distributed edge IoT (Internet of Things) environments enabled by 5G/6G networks. The proposed framework includes the special cases of binary, multi-class and multi-label ML algorithms. We deal with both horizontally and vertically partitioned datasets. Initially, (i) we propose novel secure feature selection protocols by homomorphically evaluating features' information gains in distributed environments, we proceed with (ii) novel secure training protocols over the set of selected features, then (iii) we propose novel secure building blocks which are commonly used on ML algorithms (e.g. secure sum, comparison, argmax, top-K, sorting, permutation, etc.), as well as on secure linear algebra (e.g. secure inner product, cascading matrix-vector and matrix-matrix multiplications, matrix transpose, etc.), and finally (iv) on top of proposed secure building blocks we build our novel secure ML classification protocols for various ML classifiers such as Deep Neural Networks (DNN), Support Vector Machines (SVM), Decision Trees (DT) and Random Forests (RF), different flavors of Naïve Bayes (NB), Logistic Regression (LR) and K Nearest Neighbors (KNN). Moreover, our secure classification protocols also deal with malicious users that arbitrarily deviate from the protocol and they show no loss of accuracy due to secure classifications. In the process, our participants interact with each other in order to fulfill strict security. privacy and efficiency requirements. To these ends, we provide confidentiality, integrity and authenticity to each interaction by signing their hashed contents with the corresponding participants' private key. We assure the consistency among interactions by introducing timestamps and linking them with the hashed content(s) of the preceding interaction(s). This makes our protocols a natural fit for blockchain technology. Moreover, the proposed cryptographic tools are proven to be resistant to quantum computer attacks, making our protocols applicable to the post quantum world. We did our theoretical analysis and extensive experimental evaluations over benchmark datasets related to cyber-security and health. They show that our protocols have an advantage ranging from several times to orders of magnitudes with respect to the state-of-the-art in terms of computation and communication costs. This makes our protocols among the most efficient ones in literature. Also, they are among the best in terms of security and privacy properties and allow high rate of fault tolerance and collusion attacks of dataset owners with respect to the state-of-the-art.

ÖZET

KUANTUM SONRASI 5G/6G İLE ETKİNLEŞTİRİLMİŞ ENDÜSTRİYEL IoT VE İLGİLİ SİBER GÜVENLİK VE SAĞLIK UYGULAMALARI İÇİN BLOK ZİNCİR GÜDÜMLÜ GÜVENLİ VE MAHREMİYET KORUYUCU MAKİNE ÖĞRENİMİ ALGORİTMALARI

ARTRIM KJAMILJI
Doktora Tezi, Temmuz 2021
Danışman: Prof. Dr. Albert Levi
Eş-Danışman: Prof. Dr. Erkay Savaş

Anahtar sözcükler: blok zincir; çok etiketli çok çıktılı makine öğrenimi algoritmaları; güvenli nesnelerin internet; mahremiyet koruma; öznitelik seçimi; eğitim; sınıflandırma; homomorfik şifreleme; gizli anlaşma saldırıları: dağıtık ortamlar; siber güvenlik; Tıbbi Nesnelerin İnterneti

Bu tezde, 5G/6G ağları ile etkinleştirilmiş dağıtık uç IoT (Nesnelerin İnterneti) ortamlarında yarı dürüst modele sahip güvenli ve mahremiyeti koruyucu, çok etiketli ve çok çıkışlı makine öğrenimi (ML – Machine Learning) algoritmaları için genel bir çerçeve önerilmiştir. Önerilen çerçeve, ikili, çok sınıflı ve çok etiketli ML algoritmalarının özel durumlarını içerir. Hem yatay hem de dikey olarak bölümlenmiş veri kümeleriyle çalışılmıştır. İlk olarak, (i) özniteliklerin dağıtık ortamlardaki bilgi kazanımlarını homomorfik olarak değerlendiren yeni güvenli öznitelik seçim protokolleri önerilmiştir, (ii) seçilen öznitelikler kümesi üzerinde yeni güvenli eğitim protokolleri ile ilerlenmiştir, daha sonra (iii) ML algoritmalarında yaygın olarak kullanılan yeni güvenli yapı taşları (örn. güvenli toplam, karşılaştırma, argmax, top-

K, sıralama, permütasyon, vb.) ile lineer cebir işlemlerini (örn. güvenli iç çarpım, sıralı matris-vektör ve matris-matris çarpımları, matris transpozu, vb.) güvenli hale getirecek yöntemler önerilmiştir ve son olarak (iv) önerilen güvenli yapı taşlarının üzerine, Derin Sinir Ağları (DNN - Deep Neural Networks), Destek Vektör Makineleri (SVM - Support Vector Machines), Karar Ağaçları (DT - Decision Trees), Rastgele Ormanlar (RF - Random Forests), Naïve Bayes (NB)'in değişik varyasyonları, Lojistik Regresyon (LR) ve K En Yakın Komşular (KNN - K Nearest Neighbors) gibi çeşitli ML sınıflandırıcıları için yeni güvenli ML sınıflandırma protokolleri oluşturulmuştur. Ayrıca, önerilen güvenli sınıflandırma protokolleri, keyfi olarak protokolden sapan kötü niyetli kullanıcılarla da baş eder ve güvenli sınıflandırma kaynaklı doğruluk kaybı göstermezler. İşlemler sırasında protokol katılımcıları sıkı güvenlik, mahremiyet ve verimlilik gereksinimlerini karşılamak üzere birbirleriyle etkileşime girerler. Bu amaçla, kriptografik özet fonksiyonundan geçirilen içerikler ilgili katılımcıların özel anahtarıyla imzalanarak her mesajlaşmanın gizliliği, bütünlüğü ve özgünlüğü sağlanmış olur. Mesajlaşmalar arasındaki tutarlılığı, zaman damgaları ekleyerek ve bunları önceki mesajların içerik özetlerine bağlayarak sağlamaktayız. Bu, protokollerimizi blok zincir teknolojisine doğal bir şekilde uyumlu hale getirir. Ayrıca, önerilen kriptografik araçların kuantum bilgisayar saldırılarına karşı dirençli olduğu da kanıtlanmıştır, bu da protokollerimizi kuantum sonrası dünya için kullanışlı kılmaktadır. Teorik analizler ile siber güvenlik ve sağlıkla ilgili karşılaştırmalı veri kümeleri üzerinde kapsamlı deneysel değerlendirmeler yapılmıştır. Bu analiz ve değerlendirmeler, önerilen protokollerin, hesaplama ve iletişim maliyetleri açısından bilinen en iyi duruma göre birkaç kez ila büyüklük kertesine kadar değişen oranlarda avantaj sağladığını göstermiştir. Bu durum da önerilen protokolleri literatürdeki en verimliler arasına sokmaktadır. Ayrıca, önerilen protokoller güvenlik ve gizlilik özellikleri açısından da en iyiler arasındadır ve en son teknolojiye göre yüksek hata toleransı oranı ve veri seti sahiplerinin gizli anlaşma saldırılarına karşı yüksek direnç gösterirler.

to my beloved family

ACKNOLEDGMENTS

I got to know my PhD thesis advisor, Prof. Albert Levi, during my PhD application interview. It was an instant respect and admiration for a researcher of his caliber, which would further grow when I took his "Security in wireless networks" course and while working as his TA. Similarly, I got acquainted with my co-advisor, Prof. Erkay Savaş, on that fateful day when out of curiosity I decided to only attend the first lecture of his "Cryptography" course without the intention of taking it. At the end of that day, not only that I learned from him that "there are no un-doable or hard things, rather there are challenging things", I was also so impressed by him that I decided to take the course. At the end of the semester I was so taken from the way he explained applied cryptography that, eventually, I decided that it will be part of my PhD research. Both of my advisors showed high level of professionalism, support, care and patience while they were guiding me through my PhD dissertation. It was a privilege to be supervised by them. I will always feel indebted to them for the rest of my life and I am thankful from the bottom of my heart for everything they did for me.

Special thanks go to my dissertation committee members, Prof. Cem Güneri and Prof. Yücel Saygın for their invaluable comments, feedback and help during my PhD dissertation progresses. I am also very grateful to Assoc. Prof. Alptekin Küpçü from Koç University and Asst. Prof. Ahmet Onur Durahim from Boğaziçi University for being kind enough to review my dissertation, provide useful comments and be jury members of my dissertation.

I want to acknowledge the contribution of TÜBİTAK (The Scientific and Technological Research Council of Turkey) which was generous enough to select me for their quota from North Macedonia for their International BIDEB 2215 Scholarship Grant. I express my most sincere appreciations for Sabancı University for providing a warm environment which made me feel as if I am at my home, for their continuous support throughout my tough times, for the excellent research environment and for the conference grants.

Worm thanks go to my father Irfan, mother Nebika and sister Ardiana for their support and special treatment they had towards me by sparing me from the daily family duties, especially in the final phases of my thesis. I owe them a public apology for misusing their kindness towards me by pretending to still be working on my thesis when actually I was already done with the bulk of it, so as to just to enjoy the special treatment. Mom, dad, sis, I love you!

At last, but not least, or maybe most, I would like to express my heartfelt thanks to my wife Ferihane and my 9 months old son Yahya, without whom this dissertation would have been finished at least a couple of years earlier. Though, I have to give credits to them for making it up. My wife through her unconditional support, patience, love, care and motivation throughout all of our marriage, including the stressful situations during my dissertation. My son by crying early in the mornings and in the process waking me up so I could work on my dissertation. I love you both the most!

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
CNN	Convolutional Neural Networks
CRT	Chinese Remainder Theorem
DT	Decision Trees
DM	Data Mining
DNN	Deep Neural Networks
DS	Dataset
E2DS	The Edge Encryption/Decryption Server
EC	Edge Client
EDO	Edge Dataset Owner
FHE	Fully Homomorphic Encryption
H	Hash function
HE	Homomorphic Encryption
IoT	Internet Of Things
IIoT	Industrial Internet Of Things
kNN	K-Nearest Neighbor
LR	Logistic Regression
LWE	Learning With Errors
ML	Machine Learning
MNB	Multinomial Naïve Bayes
MLMO	Multi-label multi-output
NB	Naïve Bayes
PP	Privacy Preserving
RF	Random Forests

RLWE	Ring Learning With Errors
SF	Selected Features
SIMD	Single Instruction Multiple Data
SMC	Secure Multi Party Computation
STC	Substitution-Then-Comparison Attack
SVM	Support Vector Machines
SWHE	Somewhat Homomorphic Encryption
TEAS	The Edge Aggregation Server
2PC	Two Party Computation
3PC	Three Party Computation

LIST OF SYMBOLS

SECURE AND PRIVATE FEATURE SELECTION (Chapter 4)

Cryptographic notations:

A_v	An integer vector denoted by "$\_v$" at the end, $A\_v = \{a_1, \dots, a_N\} = \{(a_i)_{i=1}^{N}\}$,
A_p	A CRT encoded plaintext denoted by "$\_p$" at the end, $A\_p = Encode(A\_v)$
A_c	A ciphertext denoted by "$\_p$" at the end, $A\_p = Encode(A\_v)$ and $A\_c = Encrypt(A\_v)$, or $A\_c = EncEncr(A\_v)$

General and common notations for all the (binary, multiclass, MLMO) datasets:

n	Number of Edge Dataset Owners (EDO)
EDO_k	The *k*-th EDO, $1 \leq k \leq n$
$NT(k)$	the number of transactions at EDO_k, for both textual and non-textual datasets
NT	The number of global transactions (records) in *n* EDOs, $NT = \sum_{k=1}^{n} NT(k)$ for both textual and non-textual datasets
W	words' set represented by permuted hashes $W = \{\pi H(w_1), \pi H(w_2), \dots, \pi H(w_{\|W\|})\}$ in textual (binary and multi-label multi-output - MLMO) datasets (DS).

Binary textual datasets:

SF	The set of *m* selected features (words), $SF = \{w_1, \dots, w_m\}$, a subset of W
C'	The set of classes $C' = \{c_h, c_s\}$ which consist of two classes, ham and spam, for the binary textual DS
$N^{(k)}(c_j)$	Number of local records belonging to class c_j at EDO_k
$N(c_j)$	Number of global records belonging to class c_j, $N(c_j) = \sum_{k=1}^{n} N^{(k)}(c_j)$
$f^{(k)}(w_i, c_j)$	Local frequency of word w_i in documents classified as belonging to c_j at at EDO_k
$f(w_i, c_j)$	Global frequency of word w_i in documents classified (labeled) as belonging to class c_j in binary textual DS, where $c_j = c_h$ or $c_j = c_s$

$N^{(k)}(w_i, c_j)$	Local count of word w_i in documents classified as belonging to c_j at EDO$_k$
$N(w_i, c_j)$	Count (number) of documents where w_i appears at least once in documents classified as c_j in binary textual DS, where $1 \leq i \leq m$ and $c_j = c_h$ or $c_j = c_s$
$N(w_i)$	Global counts (appearances) of word $w_i \in W$ for binary textual datasets, $N(w_i) = N(w_i, c_h) + N(w_i, c_s)$
$N(\overline{w_i}, c_j)$	The count (number) of documents in the binary textual dataset labeled as belonging to class c_j where word $w_i \in W$ does not appear, $N(\overline{w_i}, c_j) = N(c_j) - N(w_i, c_j)$.
$N(\overline{w})$	The count (number) of documents in the binary textual dataset were $w_i \in W$ does not appears at all, $N(\overline{w}) = NT - N(w_i)$
$P(w_i)$	The probability of w_i to appear in a document, $P(w_i) = \frac{N(w_i)}{NT}$
$P(\overline{w_i})$	The probability of w_i not appearing in a document, $P(\overline{w_i}) = \frac{N(\overline{w_i})}{NT}$
$P(w_i, c_j)$	The probability of word w_i to appear in a binary textual document classified as c_j, $P(w_i, c_j) = \frac{N(w_i, c_j)}{NT}$, where $c_j = c_h$ or $c_j = c_s$.
$P(\overline{w_i}, c_j)$	The probability of word w_i not appearing in a binary textual document classified as c_j, $P(\overline{w_i}, c_j) = \frac{N(\overline{w_i}, c_j)}{NT}$, where $c_j = c_h$ or $c_j = c_s$.
$IG(w_i)$	The information gain of word w_i in a binary textual dataset.
$P(c_j)$	The class probabilities $P(c_j) = \frac{N(c_j)}{NT}$ for binary textual DS, $c_j = c_h$ or $c_j = c_s$

Multi-label multi-output datasets:

L	The set of labels for a certain multi-label multi-output dataset, $L = \{L_1, \dots, L_{\lvert L \rvert}\}$
C^l	The set of corresponding classes for each label, $C^l = \left\{C_1^l, \dots, C_{\lvert C^l \rvert}{}^l\right\}$, $1 \leq l \leq \lvert L \rvert$ and $\lvert C^l \rvert$ is the cardinality of (number of classes belonging to) set C^l
SF^l	The set of m^l selected features for label l, $SF^l = \{H(w_1^l), \dots, H(w_{m^l}^l)\}$,

SF^{MLMO}	The set of $\lvert L\rvert$ sets of m^l selected features for the multi-label multi-output (MLMO) scenario, $SF^{MLMO} = \left\{\{SF^l\}_{l=1}^{\lvert L\rvert}\right\} = \left\{\left\{\{H(w_i^l)\}_{i=1}^{m^l}\right\}_{l=1}^{\lvert L\rvert}\right\}$, where $1 \le l \le \lvert L\rvert$.
$N({C_c}^l)$	The counts (number) of documents belonging to class ${C_c}^l$
$N(w_i^l, {C_c}^l)$	The counts of documents belonging to class ${C_c}^l$ having at least one appearance of the word (feature) $w_i^l \in SF^l$ in a multi-label multi-output dataset
$f(w_i^l, {C_c}^l)$	The frequency of word $w_i^l \in SF^l$ appearing in documents belonging to class ${C_c}^l$ in a multi-label multi-output dataset, for $1 \le l \le \lvert L\rvert$, $1 \le i \le m^l$ and $1 \le c \le \lvert C^l\rvert$
$P({C_c}^l)$	The ${C_c}^l$ classes probability, $P({C_c}^l) = \frac{N({C_c}^l)}{NT}$,$1 \le l \le \lvert L\rvert$ and $1 \le c \le \lvert C^l\rvert$
$N(w_i^l)$	The count of documents where $w_i^l \in SF^l$ appears at least once in a MLMO scenario, $N(w_i^l) = \sum_{c=1}^{\lvert C^l\rvert} N(w_i^l, {C_c}^l)$, for $1 \le l \le \lvert L\rvert$, $1 \le i \le m^l$ and $1 \le c \le \lvert C^l\rvert$
$N(\overline{w_i^l})$	The count of documents where $w_i^l \in SF^l$ does not appears, $N(\overline{w_i^l}) = NT - N(w_i^l)$
$N(\overline{w_i^l}, {C_c}^l)$	Counts of documents of class ${C_c}^l$ where w_i^l doesn't appear, $N(\overline{w_i^l}, {C_c}^l) = N({C_c}^l) - N(w_i^l, {C_c}^l)$
$P(w_i^l)$	The probability of w_i^l to appear in a document, $P(w_i^l) = \frac{N(w_i^l)}{NT}$
$P(\overline{w_i^l})$	The probability of w_i^l not to appear in a document, $P(\overline{w_i^l}) = \frac{N(\overline{w_i^l})}{NT}$
$P(w_i^l, {C_c}^l)$	The probability of word w_i^l to appear in a document belonging to ${C_c}^l$, $P(w_i^l, {C_c}^l) = \frac{N({w_i^l}_i, {C_c}^l)}{NT}$, where $1 \le l \le \lvert L\rvert, 1 \le i \le m^l$ and $1 \le c \le \lvert C^l\rvert$
$P(\overline{w_i^l}, {C_c}^l)$	The probability of word w_i^l not appearing in a document belonging to class ${C_c}^l$, $P(\overline{w_i^l}, {C_c}^l) = \frac{N(\overline{w_i^l}, {C_c}^l)}{NT}$, where $1 \le l \le \lvert L\rvert, 1 \le i \le m^l$ and $1 \le c \le \lvert C^l\rvert$
$IG^l(w_i^l)$	Information gain of $w_i^l \in SF^l$ for label l, where $1 \le l \le \lvert L\rvert, 1 \le i \le m^l$

SECURE AND PRIVATE MACHINE LEARNING TRAINING (Chapter 5)

Multiclass non-textual datasets (including the corresponding symbols of the previous chapter(s)):

f	Number of features in a non-textual dataset
c	Number of classes (labels) in a non-textual dataset
F_i	The values feature i $F_i = \{V_{1,Fi}, V_{2,Fi}, \dots, V_{\lvert Fi \rvert,Fi}\}$ can take for non-textual datasets, $V_{m,Fi}$ is the m-th element of the feature set F_i, and $1 \leq i \leq f$, $1 \leq m \leq \lvert \mathrm{F}_i \rvert$.
C	The set of classes for non-textual (multiclass) datasets, $C = \{C_1, C_2, \dots C_c\}$
$N^{(k)}(C_j)$	The local counts of records belonging to class C_j at EDO$_k$
$N(C_j)$	The global (overall) counts of records belonging to class C_j in *n* EDOs.
$N^{(k)}(V_{m,Fi}; C_j)$	The count of the m-th value of feature F_i having class C_j at EDO$_k$
$N(V_{m,Fi}; C_j)$	The global count of the m-th value of the feature F_i having class C_j where $1 \leq i \leq f,\ 1 \leq j \leq c$ and $1 \leq m \leq \lvert F_i \rvert$ for non-textual datasets
$P(C_j)$	The class probability, i.e. the probability for a certain instance to belong to class C_j $P(C_j) = \frac{N(C_j)}{NT}$,
$P(V_{m,Fi} \lvert C_j)$	The conditional feature value-class probabilities, $P(V_{m,Fi} \lvert C_j) = \frac{N(V_{m,Fi}\,;\,C_j)}{N(C_j)}$ where $1 \leq i \leq f,\ 1 \leq j \leq c$ and $1 \leq m \leq \lvert F_i \rvert$.
$C(\cdot)$	The trained MNB/NB model for non-textual DS consisted of $P(C_j)$, $P(V_{m,Fi} \lvert C_j)$, where $1 \leq i \leq f,\ 1 \leq j \leq c$ and $1 \leq m \leq \lvert F_i \rvert$.

Binary textual datasets (including the corresponding symbols of the previous chapter(s)):

$P(w_i \lvert c_j)$	$P(w_i \lvert c_j) = \frac{f(w_i,c_j)}{N(c_j)}$ and $P(w_i \lvert c_j) = \frac{N(w_i,c_j)}{N(c_j)}$ denote the conditional word-class probabilities for MNB and NB cases in binary textual DS, respectively.
$C_{TM}(\cdot)$	The trained MNB/NB model for binary textual DS consisted of $P(c_j)$, $P(w_i \lvert c_j)$,where $c_j = c_h$ or $c_j = c_s$ and $w_i \in SF$.

Multi-label multi-output datasets (including the corresponding symbols of the previous chapter(s)):

$P\left(w_i^l \mid {C_c}^l\right)$	The conditional word-class probability. $P\left(w_i^l \mid {C_c}^l\right) = \frac{N\left(w_i^l, {C_c}^l\right)}{N\left({C_c}^l\right)}$ for the NB classifier, $P\left(w_i^l \mid {C_c}^l\right) = \frac{f\left(w_i^l, {C_c}^l\right)}{N\left({C_c}^l\right)}$ for the MNB classifier.
TM^{MLMO}_c	The trained model for the multi-label multi-output datasets, which is consisted of $P\left({C_c}^l\right)$ and $P\left(w_i^l \mid {C_c}^l\right)$ for $1 \leq l \leq \lvert L \rvert$, $1 \leq i \leq m^l$ and $1 \leq c \leq \lvert C^l \rvert$.

SECURE AND PRIVATE MACHINE LEARNING CLASSIFICATIONS (Chapter 6)

Multiclass non-textual datasets (including the corresponding symbols of the previous chapter(s)):

X	Unclassified feature vector, $X = \{X_1, X_2 \ldots, X_f\}$, where $X_i \in F_i$

Binary textual datasets (including the corresponding symbols of the previous chapter(s)):

q_v	The query vector for textual datasets, $q\_v = \{1, f_q(w_1), \ldots, f_q(w_m)\}$, $f_q(w_i)$ is the frequency of appearance of word w_i in the query q_v and $w_i \in SF$.

Multi-label multi-output datasets:

q_v^{MLMO}	The multi-label multi-output unclassified query vector, $q\_v^{MLMO} = \left\{\left\{1, \left\{f_q^l\left(w_i^l\right)\right\}_{i=1}^{m^l}\right\}_{c=1}^{\lvert C^l \rvert}\right\}_{l=1}^{\lvert L \rvert}$

Linear algebra based ML classification (including the corresponding symbols of the previous chapter(s)):

M	The trained model matrix for classification based on linear algebra. For the NB case $M = \left[\left\{C^{(j)}\right\}_{j=1}^{c}\right]_{c \times (f+1)}$, where for $1 \leq i \leq f$, $1 \leq j \leq c$ and $1 \leq m \leq \lvert F_i \rvert$. $C^{(j)} = \left\{\lfloor K log P(C_j) \rfloor, {}_{i=1}^{f}\left({}_{m=1}^{\lvert F_i \rvert}\lfloor K log(V_{m,F_i} \mid C_j)\rfloor\right)\right\}$.

	For SVM and LR $\mathrm{M} = \left[\left\{\mathrm{W}^{(\mathrm{j})}\right\}_{\mathrm{j}=1}^{\mathrm{c}}\right]_{\mathrm{c}\times(\mathrm{f}+1)}$, where $\mathrm{W}^{(\mathrm{j})} = \left\{\mathrm{b}^{(\mathrm{j})}, \mathrm{w}_1^{(\mathrm{j})}, \dots, \mathrm{w}_\mathrm{n}^{(\mathrm{j})}\right\}$ is the j-th hyperplane. For deep neural networks (DNN) M is consisted of all the layer matrices M^i and activation functions $f^i(\cdot)$, where $0 \le i \le l$, and l is the number of layers in DNN
X'	The query vector $X = \left\{1, \mathop{}\limits_{i=1}^{f}\left(\mathop{}\limits_{m=1}^{\lvert F_i\rvert} V_{m,F_i}\right)\right\} \cong \{1, X_1, \dots, X_f\}$, where $X_i \in F_i$, $1 \le i \le f$ and $1 \le m \le \lvert F_i\rvert$. If V_{m,F_i} appears in the original query (i.e. $V_{m,F_i} = X_i$), its value is 1, otherwise it is 0.
S	A matrix whose columns are q query vectors denoted as $\mathrm{X}^{(\mathrm{k})}$, for the multi-query linear algebra based classification, thus $\mathrm{S} = \left[\left\{\mathrm{X}^{(\mathrm{k})}\right\}_{\mathrm{k}=1}^{\mathrm{q}}\right]_{(\mathrm{f}+1)\times\mathrm{q}}$
kNN, decision tree and random forests classifiers:	
Y^{r_i}	r_i-th dataset record, $Y^{r_i} = \left\{Y_1^{r_i}, Y_2^{r_i}, \dots, Y_f^{r_i}\right\}$ for kNN. When the records are randomly permuted by a permutation π, they are denoted as $Y^{\pi(r_i)}$
$d(X, Y^{r_i})$	the distance between query X and the r_i-th dataset record for kNN
val_{F_i}	threshold value of features F_i's corresponding node in the decision tree, $1 \le i \le f$

Chapter 1

INTRODUCTION

IPv6 increased the bit size of the IP addresses from 32 to 128 bits. This contributes to the estimated increase of IP connected devices in the Internet of Things (IoT) from 8.4 billion in 2017 to the predicted 30+ billion in 2020 [1]. Together with the emerging trends of other information technologies such as ubiquitous (makes computing omnipresent, anytime and everywhere), wearable (computing devices worn under, with, or on top of clothing) and cloud computing combined, those devices contribute to the rise of the global data volume from 4.4 zettabytes in 2013 to the predicted 44 zettabytes in 2020 [2] in what is known as Big Data. Often those IoT devices collect data to form private datasets, such as different hospitals collecting data about their patients' disease predictions together with the corresponding patient symptoms, or different cyber-security companies collecting log files of computer systems together with the corresponding host/network attack(s) or normal behavior, etc. If those data sets of the same nature are collected in different environments (e.g. different hospitals, different IT systems, etc.) and have different statistical properties, it has been shown that when they are merged into a single data set to train a machine learning (ML) model, the model often ends up being more accurate in its' classifications (predictions) than the human expert of the same field or than the trained models obtained from each of the datasets separately [3].

Fortunately, in the last couple of decades, many techniques have been proposed that enable us to either partially or in total overcome the above mentioned problems. Those techniques enable the ML experts to successfully train and get the final trained ML model from multiple data set owners with little or no exposure of their data or information related to the data. This process is known as **privacy preserving (PP) training**. On the other hand, those techniques can also be utilized for the case when a user wants to classify his query without letting the trained model owner learn anything neither about his query data nor the final prediction (classification), while the user also should not learn anything about the trained model. This process is known as **privacy preserving classification.**

1.1. Motivation and problem statement

In 2000, [1] and [2] almost simultaneously came up with research under similar titles where they addressed the privacy of the datasets over which ML models were trained. Several others followed, some of which are presented in the surveys at [3] and [4]. Those are known as privacy preserving training schemes. Common for those early schemes was that they solely concentrated on the privacy of the datasets, with little or no concern for the privacy of the classification stage. They also payed no attention to the privacy of the trained model as well [3-4].

Realizing these shortcomings, dozens of schemes followed afterwards, which exclusively deal with the classification (inference) stage for different ML classifiers such as Naïve Bayes (NB), Deep Neural Networks (DNN), Support Vector Machines (SVM), Logistic Regression (LR), Decision Trees (DT), Random Forests (RF), etc. These schemes fall into the privacy preserving classification category, such as those in [5-11], to name a few. There are also a few schemes that for consistency and continuity reasons addresses both privacy preserving training and classification problem under the same system architecture and environment settings, such as those proposed in [12-15] and [30-32].

However, before training an ML model, it is common practice to do some pre-processing and feature selection on top of the dataset(s) over which the training is being done. Especially this is common for datasets which are highly dimensional (have a huge number of features), a typical occurrence for text classification datasets such as SMS spam, e-mail spam, document classification, etc., which are known to harbor tens, hundreds of thousands or even millions of unique features (words, tokens). By applying some feature selection over such datasets, not only we save valuable amount of computation and communication cost during the training and classification stages later on, rather trained models obtained over datasets on which a certain feature selection has been applied, in general, are shown to be more accurate during the classification stage [23-29].

Despite this, surprisingly, there is little work done on the area of secure feature selection, especially on edge IoT environments. The few existing schemes have one or several setbacks and disadvantages, which are elaborated in the next section. Furthermore, they are not designed to have in mind the computation and communication environments of edge computing in IoT, nor do the existing schemes provide a mechanism for ensuring the end-to-end integrity, authenticity and consistency (continuity, order of sequence) by which the interactions are done among different participants in the system while executing the protocols.

1.2. Contributions

Definition 1.1: A **blockchain** is a list of blocks (records), where each block has a timestamp and its transaction data (usually in form of a Merkle tree [16]) and is linked with the previous block(s) by including its (theirs) hash(es) in itself, thus forming a chain of blocks (blockchain) [17].

Definition 1.2: Internet of Things (IoT) is a set of cooperating devices (sensors, mobiles, etc.) that can collect and transfer data over a wireless network without explicit human intervention with the aim of reaching a certain functionality [18]

Definition 1.3: Edge computing refers to the technologies that enable computations to be performed at the edge (end) of the network. It's usually done so to improve response times and use less network resources as well as resources of other devices around by doing local processing at the edge node (device). For example, a smart phone is an edge device [19].

Our contributions: In order to overcome the afore mentioned shortcomings (elaborated in more details in Chapter II), we propose a novel **secure feature selectio**n (filtering) protocol based on information theoretic metrics such as entropy. Concretely, we homomorphically evaluate features' information gains **on distributed (horizontally and vertically**

partitioned) datasets over edge IoT devices and select the top m ones with the highest information gain. We choose the information gain since it is shown to be among the most effective feature selecting metrics for text classification [20, 34], and due to the lack of secure and private schemes that use it in literature [23-29]. Since in practice we deal with datasets with sensitive content and knowing that our protocols use several interactions between the participant, for each interaction of each participant we introduce a block (record) of its interaction data and provide integrity by hashing the block's content, confidentiality by encrypting the sensitive data and authenticity by encrypting the overall block content with participant's private key. Furthermore, in order to assure the consistency (order of execution) by which the interactions among participants are done, in our protocols we introduce timestamps to the blocks and link them with the hashed content of the preceding block(s). This makes our schemes a natural fit for **blockchain technology**. In the process each participant keeps only the blocks which are generated by him, forming what we call vertically partitioned distributed public ledger (Chapter 4).

To show the effectiveness of our scheme, on top of the selected features, we provide a **secure and private training and classification protocols** over the same context (system architecture, environment and security settings). In this sense, we continue our blockchain, enabling in the process end-to-end (from raw datasets till the final trained model, i.e. from secure feature selection to secure training) security characteristics inherited by blockchain, (Chapter 4). During the secure classification stage each client, before classifying his query, can verify (check) the correctness, flow and the consistency by which the final trained model was obtained using the blockchain. We formally **prove the security of all of them under the semi-honest model**.

For efficiency purposes, in order not to overload the network and other participants, we tend to do local **processing at the edge** on clear (plain, un-encrypted) data as much as possible, rather than homomorphically evaluate them later on. While doing so we take into consideration the **heterogeneity** (in terms of hardware and software platforms) and the **restricted resources** that are characteristic for edge IoT devices [21, 22]. Our schemes have **high rate of fault tolerance and resistance to collusion attacks** so that, out of n dataset owners, they allow up to $n-3$ failures or up to $n-2$ collusions. Moreover, the underlying cryptographic tools that we use while executing our protocols are proven to be secure under quantum computer attacks, making our protocols **suitable for the post quantum world**. This makes our schemes one of the rare ones (if not the only one) that utilizes blockchain technology to provide an and end-to-end (from raw un-preprocessed datasets to final trained model) secure and private framework for ML algorithms in edge IoT environments [21, 22]. We should note that **our protocols** about secure feature selection, training and classification, **are independent from each other**, in terms that, according to the scenario and needs, each of them can be used solely or in combination with secure and private protocols from other research schemes.

1. Secure feature selection (*secFS*) protocol - requirements:

- Privacy of the input features. We achieve this by randomly permuting the hashes of the input values (words, tokens)
- Privacy of the input features' values
- Security and privacy of intermediate results. We keep the intermediate results of all of our protocols secure and private since they might be used as a trapdoor for total or partial leakage of the input or the output of the corresponding protocol.
- Partial privacy for the output (of the top m selected features). The output cannot be totally private since it is needed as an entry point (input) for the secure classification stage when clients prepare their queries in accordance to the selected features. However, the selected features can be kept private for the secure training protocol.

2. Secure training (*secT*) protocol - requirements:

- Privacy of the input features. The inputs here are the selected features, i.e. the output) of the secFS protocol.
- Privacy of the input features' values
- Security and privacy of intermediate results
- Privacy of the output, i.e. the trained model. We provide the option for the trained model not to be revealed to any of the participants, even during the PP classification stage. This is one of the rare protocols to keep private the final trained model at any stage.

3. Secure classification (*secC*) protocol- requirements:

- Privacy of the trained model
- Privacy of the user query for both query features and their corresponding values (frequencies)
- Security and privacy of intermediate results
- Privacy of the output, i.e. the final classification result
- No loss of accuracy with respect to the plain classifier

We should note the for the purpose of the secure classification protocols we propose the secure comparison protocol based on arithmetic circuits, which securely compares two integers. On top of it we also propose the secure comparison of all data slots – SCADS protocol, which in turn is needed for the secure argmax and secure top-K protocols. While our experimental evaluations show that the proposed secure comparison protocol when used isolated (not in combination with our secure classification protocols) doesn't perfectly hide the difference of the two integers that it compares (Section 6.9), yet we theoretically proof (in Section 6.10) that when our secure comparison, hence SCADS as well, are used in combination with our secure classification protocols, they help us hide the trained model and the user query. We base our proof on top of well-established cryptographic primitives (assumptions in this case), such as ones based on The Learning With Errors – LWE schemes.

4. Have the remaining characteristics given in the Tables of Chapter 3.

The published/accepted/submitted papers derived from this dissertation are:
1. Kjamilji, Artrim, Erkay Savaş, and Albert Levi. "Efficient Secure Building Blocks With Application to Privacy Preserving Machine Learning Algorithms." *IEEE Access* 9 (2021): 8324-8353." (published)
2. Kjamilji, Artrim. Albert Levi, Erkay Savaş and Osman Berke Güney "Secure Matrix Operations for Machine Learning Classifications Over Encrypted Data in Post Quantum Industrial IoT" The International Symposium on Networks, Computers and Communications (ISNCC'21). IEEE, 2021 (accepted, Dubai, 31.10-02.11.2021).

3. Kjamilji, Artrim. Albert Levi, Erkay Savaş "Blockchain driven secure feature selection, training and classifications in distributed edge IoT environments" (submitted to IEEE Journal on Selected Areas in Communications).

Besides the above papers, there a few of other future papers for which in this dissertation we provide their theoretical background, security analysis and proofs, pseudocodes and we already have their implementations (codes) which can be found in [98], as well as their experimental evaluations. After writing their corresponding manuscripts, they will be submitted to related journals/conferences.

1.3. Dissertation organization

In Chapter 2 we give preliminaries related to the research. Namely, we introduce the concepts of NB, multinomial NB (MNB) for multi-label multi-output datasets as well as ML classifiers based on linear algebra operations such as DNN, SVM, LR, NB and MNB (again, this time represented through linear algebra operations). We proceed with the information gain, as well as cryptographic primitives, concepts, definitions and theorems which will be used in later chapters. In Chapter 3 we elaborate the related research on secure feature selection, secure training and secure classification. In Chapter 4 we give our algorithms on secure and private feature selection for multi-label multi-output datasets, while in Chapter 5 we present the corresponding secure and private training algorithms. In Chapter 6 we present novel secure building blocks for general purpose, such as secure sum, secure permutation and its inverse, secure comparison and secure sorting (whose derivatives are secure top-K and secure argmax). We proceed with novel secure building blocks on linear algebra such as secure dot (inner) product, secure compression of sparsely encoded matrices, a couple of versions of secure matrix-vector product, a couple of versions of secure matrix-matrix product, secure matrix transpose and secure Frobenius product. On top of those building blocks we built our secure and private ML classifiers for KNN, DT, RF, NB, MNB, DNN, SVM and LR. At the end of each of the Chapters 3-6 we give theoretical experimental evaluations and comparisons of our protocols with the state-of-the-art as well as we provide security proofs for all of our protocols under the semi-honest (honest but curious) model. In Chapter 6 we deal with malicious users as well during the secure and private classifications. Finally, in

Chapter 7 we conclude our research with discussions as well as proposals and ideas for future research and directions.

Chapter 2

BACKGROUND INFORMATION AND PRELIMINARIES

In this chapter, we provide the preliminaries related to all of our schemes throughput the dissertation. Namely, we derive NB classifier from Bayes' Theorem and use it for both textual and non-textual multi-label multi-output dataset (which in itself includes the special cases of binary classes, multi-class, multinomial and multi-label scenarios). Then we introduce the most commonly used classifiers such as KNN, DF, RF and the others that can be expressed through linear algebra operations such as SVM, LR, MNB and DNN. We proceed with introducing the information gain and cryptographic primitives related to security definitions, concepts and theorems which will be our used for proving other theorems throughout the dissertation. We conclude with the general system architecture of our schemes, participants and their adversary models and briefly introduce our protocol flows, which are elaborated in details in their corresponding chapters.

The notation given here will be valid throughout the paper. We denote a vector of integers by adding the term "$\_v$" at the end of the vector's name, i.e. $int\_v = \{a_1, a_2, \dots a_N\} = \{(a_i)_{i=1}^{N}\}$. By $int1\_v + int2\_v$; $int1\_v - int2\_v$; $int1\_v \times int2\_v$; $int1\_v/int2\_v$, we denote the component (index) wise addition, subtraction, multiplication and division of two integer vectors, respectively. $\lceil \cdot \rceil$ denotes the ceiling function (rounding to the closest greater integer), while $\lfloor \cdot \rfloor$ the floor function (rounding to the closest smaller integer). With $H(\cdot)$ we denote a cryptographic hashing function and by $|S|$ we denote the cardinality (number of elements) of a certain set *S*.

2.1. Bayes' Theorem and Naïve Bayes

Let $A_1, \dots, A_N, B$ be random variables. Let $P(B)$, $P(A_1, \dots, A_N)$, $P(B|A_1, \dots, A_N)$ and $P(A_1, \dots, A_N|B)$ be the probability of observing B, the joint probability of observing the random variables $A_1, \dots, A_N$, the conditional probability of observing B after $A_1, \dots, A_N$ are observed and the conditional probability of observing $A_1, \dots, A_N$ after B is observed, respectively. Then the Bayes' Theorem can be written as:

$$P(B|A_1, \dots, A_N) = \frac{P(A_1,\dots,A_N|B)P(B)}{P(A_1,\dots,A_N)} \tag{2.1}$$

If we are interested on which value of B has the highest probability of appearance after $A_1, \dots, A_N$ have been observed, but not on their actual values, then if we Naively assume that $A_1, \dots, A_N$ are independent to each other (thus $P(A_1, \dots, A_N|B) = \prod_{i=1}^{N} P(A_i|B)$) and knowing that logarithm is a monotonically increasing function, then (1) can be written as:

$$B_{v_{max}} = \left(\begin{matrix} argmax \\ B_v \in B \end{matrix} P(B|A_1, \dots, A_N)\right) = \begin{matrix} argmax \\ B_v \in B \end{matrix} \sum_{i=1}^{N} logP(A_i|B)\, P(B_v) \tag{2.2}$$

where $B_{v_{max}}$ is the value of B which has the highest posterior probability (probability of appearance) after $A_1, \dots, A_N$ have been observed. Equation (2.2) is known as the Naive Bayes formula.

2.1.1. Naïve Bayes for multi-class non-textual datasets

Naïve Bayes is a supervised machine learning (ML) technique used for classification, hence its model is trained from a dataset(s) assumed to be correctly labeled. Such datasets have $f + 1$ columns, where f is the number of features (attributes) and one column is for the class (label). We assume that each feature is categorical (if a certain feature is continuous, it can be discretized). Each record (transaction, instance, row) in the data set for each of the f

features can take one of the values from the feature set $F_i = \{V_{1,Fi}, V_{2,Fi}, \dots, V_{|Fi|,Fi}\}$, where $|F_i|$ is the cardinality (number of elements) of the set F_i and $V_{m,Fi}$ is the m-th element of the feature set F_i, and $1 \leq i \leq f$, $1 \leq m \leq |F_i|$. Let $F \subseteq R^f$ be the set of the Cartesian products of all the elements of all features' set, namely $F = F_1 \times F_2 \times \dots \times F_f$. All of the instances belong to one class from set of classes $C = \{C_1, C_2, \dots C_c\}$, so we have c classes in total. The total number of instances (transactions) in the dataset is *NT*. In this sense, a dataset is comprised of records which are tuples $\{Y_r, Z_r\}$, where $Y_r \in F$ and $Z_r \in C$. Let $N(C_j)$ denote the global (overall) frequency (counts) of class C_j and let $N(V_{m,Fi}; C_j)$ denote the frequency of the m-th value of the feature F_i with class C_j where $1 \leq i \leq f$, $1 \leq j \leq c$ and $1 \leq m \leq |F_i|$. Obviously, $NT = \sum_{j=1}^{c} N(C_j)$. For the class and conditional feature value-class probabilities, we have $P(C_j) = \frac{N(C_j)}{NT}$ and $P(V_{m,Fi}|C_j) = \frac{N(V_{m,Fi}\,;\,C_j)}{N(C_j)}$, respectively, and they actually represent the trained model.

To classify an unclassified feature vector $X = \{X_1, X_2 \dots, X_f\}$, where $X_i \in F_i$ (therefore $X \in F$), means to label it with a class from the set C which has the highest posterior probability. Assuming that the features are independent from each other, thus $P(X_1, X_2, \dots, X_f|C_j) = \prod_{i=1}^{f} P(X_i|C_j)$, then using (2.2) for the NB classifier we have:

$$C(X) = argmax_{1 \leq j \leq c}\left[logP(C_j) + \sum_{i=1}^{f} logP(X_i|C_j)\right] \tag{2.3}$$

If the main (global) dataset was obtained by merging n other data sets, then for the frequencies of the dataset we have $N(C_j) = \sum_{k=1}^{n} N^{(k)}(C_j)$, $N(V_{m,Fi}; C_j) = \sum_{k=1}^{n} N^{(k)}(V_{m,Fi}; C_j)$ and $NT = \sum_{k=1}^{n} NT(k)$, where $N^{(k)}(C_j)$ is the local frequency (counts) of class C_j at the data set k, while $N^{(k)}(V_{m,Fi}; C_j)$ is the frequency of the m-th value of feature F_i having class C_j at the dataset k, and $NT(k)$ is the number of transactions at the dataset k, where $1 \leq i \leq f$, $1 \leq j \leq c$, $1 \leq k \leq n$ and $1 \leq m \leq |F_i|$. Having this in mind, then letting each $X_i = V_{m,Fi}$ for $1 \leq i \leq f$ and $1 \leq m \leq |F_i|$, from (2.3) we have:

$$C(X) = argmax_{1\le j\le c} \left[log \frac{\sum_{k=1}^{n} N^{(k)}(C_j)}{\sum_{k=1}^{n} NT(k)} + \sum_{i=1}^{f} log \frac{\sum_{i=1}^{n} N^{(k)}(V_{l,Fi};C_j)}{\sum_{k=1}^{n} N^{(k)}(C_j)}\right] \tag{2.4}$$

In order to integerize for encryption purposes, we multiply the probabilities with a constant K, so for (2.4) we have:

$$C(X) = argmax_{1\le j\le c} \left|K\left[log \frac{N(C_j)}{NT} + \sum_{i=1}^{f} log \frac{N(V_{m,Fi}; C_j)}{N(C_j)}\right]\right| \approx$$
$$logP(C_j) + \sum_{i=1}^{f} logP(V_{m,Fi}|C_j) \tag{2.5}$$

2.1.2. Multinomial Naïve Bayes for multi-label multi-output textual datasets

For simplicity and better understanding, we firstly present the scenario with binary textual datasets and then generalize it to deal with multi-label multi-output textual datasets.

Let the transactions (documents, records, instances) of a certain (pre-processed) dataset DS have a selected feature set SF consisting of m words (features), thus $SF = \{w_1, \dots, w_m\}$. For simplicity and without loss of generality, let the set of classes C of DS consist of two classes, ham and spam, thus $C = \{c_h, c_s\}$. Let $f(w_i, c_j)$ denote the frequency of word w_i in documents classified (labeled) as belonging to class c_j, while $N(w_i, c_j)$ denote the count (number) of documents where w_i appears at least once in documents classified as belonging to c_j, where $1 \le i \le m$ and $c_j = c_h$ or $c_j = c_s$. In this sense, if w_i appears several times in a document belonging to c_j then its frequency $f(w_i, c_j)$ will be incremented as many times as it appears in that particular document, while its document count $N(w_i, c_j)$ will be incremented by one. Let NT and $N(c_j)$ denote the total number of transaction and the number of records belonging to class c_j, respectively. Apparently $NT = \sum_{c_j=c_h,c_j=c_s} N(c_j)$. Let $P(c_j) = \frac{N(c_j)}{NT}$ denote the class probabilities. Let $P(w_i|c_j) = \frac{f(w_i,c_j)}{N(c_j)}$ and $P(w_i|c_j) = \frac{N(w_i,c_j)}{N(c_j)}$ denote the conditional word-class probabilities for the Multinomial Naïve Bayes (MNB) and Naïve Bayes (NB) cases, respectively. Actually $P(c_j)$ and $P(w_i|c_j)$, where $1 \le i \le m$ and $c_j = c_h$ or $c_j = c_s$, represent the trained model $C_{TM}(\cdot)$ for the MNB and NB classifier, respectively. In order to give a chance to words whose conditional probability is zero, due to their counts or frequencies being zero, we add one to each count (frequency) accompanied by adding the vocabulary size (number of unique words) to the denominator of the corresponding conditional probabilities. This process is called Laplace Smoothing. For the MNB case un-classified queries have the form of $q\_v = \{1, f_q(w_1), \dots, f_q(w_m)\}$, where $f_q(w_i)$ is the frequency of appearance of word w_i in the query q_v, while for the NB case it has the form of the binary vector $q\_v = \{1, N_q(w_1), \dots, N_q(w_m)\}$ where $N_q(w_i) = 1$ if w_i appears at least once in the query and $N_q(w_i) = 0$ if w_i doesn't appear in the query, for $1 \leq i \leq m$ and $w_i \in SF$. Applying Naïve Bayes' Theorem for multinomial datasets [18], [33], [35], [39], we classify q_v as:

$$C_{TM}(q\_v) = P(c_j|q\_v) = \underset{c_j=c_h \text{ or } c_j=c_s}{argmax} \frac{P(c_j)\left(\prod_{i=1}^{m} P(w_i|c_j)\right)}{P(q\_v)} \tag{2.6}$$

Similar to the reasoning in Chapter 2.1 and (2.2), having in mind that $P(q\_v)$ is always the same, that logarithm is a monotonically increasing function and naively assuming that appearances of words in a query are independent of each other, i.e. $P(q\_v|c_j) = \prod_{i=1}^{m} P(w_i|c_j)^{f_q(w_i|c_j)}$, then for the MNB case (2.2) can be written as:

$$C_{TM}(q\_v) = sign\left[K\left(logP(c_h) - logP(c_s)\right) + \sum_{i=1}^{m} Kf_q(w_i)\left(logP(w_i|c_h) - logP(w_i|c_s)\right)\right] \tag{2.7}$$

if $C_{TM}(q\_v) > 0$, i.e. $sign = +$, then q_v is classified as c_h (ham), otherwise as c_s (spam). K is a constant used to integerize terms of $C_{TM}(q\_v)$ for encryption purposes (Chapter 2.4.1). If in (2.6) instead of frequencies $f_q(w_i)$ we use the counts $N_q(w_i)$, then (2.6) is valid for the NB case.

If the dataset DS is obtained by merging n other datasets denoted as DS$_k$ for $1 \leq k \leq n$, then $NT = \sum_{k=1}^{n} NT(k)$, $N(c_j) = \sum_{k=1}^{n} N^{(k)}(c_j)$, $f(w_i, c_j) = \sum_{k=1}^{n} f^{(k)}(w_i, c_j)$ and $N(w_i, c_j) = \sum_{k=1}^{n} N^{(k)}(w_i, c_j)$, where $NT(k)$, $N^{(k)}(c_j)$, $f^{(k)}(w_i, c_j)$ and $N^{(k)}(w_i, c_j)$ denote the number of transactions (records), number of records labeled as c_j, frequency of word w_i in documents labeled as c_j and the count (number) of at least one appearance of w_i in documents labeled as c_j at dataset k, respectively, for $c_j = c_h$ or $c_j = c_s$.

For the multi-label multi-output case, where each label is a multiple-classes, let the set of labels for a certain dataset be $L = \{L_1, \dots, L_{|L|}\}$ and let the set of corresponding classes for each label be $C^l = \left\{C_1^l, \dots, C_{|C^l|}{}^l\right\}$, where $1 \le l \le |L|$ and $|C^l|$ is the cardinality of (number of classes belonging to) set C^l. Let $SF^l = \{w_1^l, \dots, w_{m^l}^l\}$ be the set of m^l selected features for label l, where $1 \le l \le |L|$. Let $N(C_c{}^l)$, $N(w_i^l, C_c{}^l)$ and $f(w_i^l, C_c{}^l)$, be the counts (number) of documents belonging to class $C_c{}^l$, counts of documents belonging to class $C_c{}^l$ having at least one appearance of the word (feature) $w_i^l \in SF^l$ and the frequency of word w_i^l appearing in documents belonging to class $C_c{}^l$, respectively, for $1 \le l \le |L|$, $1 \le i \le m^l$ and $1 \le c \le |C^l|$. Let $P(C_c{}^l) = \frac{N(C_c{}^l)}{NT}$ denote the $C_c{}^l$ classes probability and $P(w_i^l | C_c{}^l) = \frac{N(w_i^l, C_c{}^l)}{N(C_c{}^l)}$ or $P(w_i^l | C_c{}^l) = \frac{f(w_i^l, C_c{}^l)}{N(C_c{}^l)}$ denote the conditional word-class probability for NB, respectively for MNB case. Let TM^{MLMO} denote the trained model for the multi-label multi-output datasets, which is consisted of $P(C_c{}^l)$ and $P(w_i^l | C_c{}^l)$ for $1 \le l \le |L|$, $1 \le i \le m^l$ and $1 \le c \le |C^l|$. Then for the query $q\_v^{MLMO} = \left\{\left\{1, \{f_q^l(w_i^l)\}_{i=1}^{m^l}\right\}_{c=1}^{|C^l|}\right\}_{l=1}^{|L|}$, which for label l has m^l words (features), the corresponding classification for each label assigns a class to q_v according to MNB, thus:

$$TM^{MLMO}(q\_v^{MLMO}) = \underset{1 \le l \le |L|}{P\left(\{C_c{}^l\}_{l=1}^{|L|} \middle| q\_v\right)} = \underset{\substack{1 \le l \le |L| \\ 1 \le c \le |C^l|}}{argmax} \left[K log P(C_c{}^l) + \sum_{i=1}^{m^l} K f_q^l(w_i^l) log P(w_i^l | C_c{}^l)\right] \tag{2.8}$$

If $f_q^l(w_i^l)$s are substituted by $N_q^l(w_i^l)$ in (2.8) and probabilities belong to the NB case, then the classification of q_v is done according to NB.

We should emphasize that if each value of each feature is considered as a binary feature on its own (that can take two values, 0 or 1, depending on whether its present or not in the record or the query), then the scenario for textual multi-label multi-output datasets is also valid for the non-textual ones.

2.2. Information gain

For the binary classification case, let $N(\overline{w_i}, c_j) = N(c_j) - N(w_i, c_j)$ denote the count (number) of documents in the dataset labeled as belonging to class c_j where word w_i does not appear, and $c_j = c_h$ or $c_j = c_s$. Let $N(w_i) = N(w_i, c_h) + N(w_i, c_s)$ be the number of documents were word w_i appears at least once and $N(\overline{w}) = NT - N(w_i)$ be the count (number) of documents were w_i does not appears at all. Let $P(w_i) = \frac{N(w_i)}{NT}$ denote the probability of w_i to appear in a document, $P(\overline{w_i}) = \frac{N(\overline{w_i})}{NT}$ denote the probability of w_i not appearing in a document, $P(w_i, c_j) = \frac{N(w_i,c_j)}{NT}$ denote the probability of word w_i to appear in a document classified as c_j and $P(\overline{w_i}, c_j) = \frac{N(\overline{w_i},c_j)}{NT}$ denote the probability of word w_i not appearing in a document classified as c_j, where $c_j = c_h$ or $c_j = c_s$. Then, the information gain (IG) of a word w_i is defined as [34-35]:

$$IG(w_i) = \sum_{c_j=c_h,c_s} \left(P(w_i, c_j) log \left(\frac{P(w_i,c_j)}{P(w_i)P(c_j)} \right) + P(\overline{w_i}, c_j) log \left(\frac{P(\overline{w_i},c_j)}{P(\overline{w_i})P(c_j)} \right) \right) \tag{2.8}$$

The information gain is a quantitative metric which measures the reduction of entropy (uncertainty) of a query q_v to belong to a class c_j after word w_i has been observed in the query. The higher the entropy reduction, the more information gain the word w_i offers. This is the reason that makes information gain one of the most effective tools in dimension reduction (feature selection), especially when choosing the top m words with the highest information gain in what is known as "bag-of-words" [33],[34],[37]. Substituting the probabilities with their counts and having in mind that we want to find the top m words with the highest IG, but not their exact IG, then (2.8) can be rewritten as:

$$IG(w_i) \sim \sum_{c_j=c_h,c_s} \left(N(w_i, c_j) log \left(\frac{N(w_i,c_j)NT}{N(w_i)N(c_j)} \right) + N(\overline{w_i}, c_j) log \left(\frac{N(\overline{w_i},c_j)NT}{N(\overline{w_i})N(c_j)} \right) \right) \tag{2.9}$$

which makes the homomorphic evaluations and selection of the top m words with the highest IG easier (Chapter 2.4).

For the multi-label multi-output datasets let $N(w_i^l) = \sum_{c=1}^{|C^l|} N(w_i^l, {C_c}^l)$, $N(\overline{w_i^l}) = NT - N(w_i^l)$, and $N(\overline{w_i^l}, {C_c}^l) = N({C_c}^l) - N(w_i^l, {C_c}^l)$ denote the count (number) of documents where $w_i^l \in SF^l$ appears at least once, count of documents where w_i^l doesn't appear at all and counts of documents of class ${C_c}^l$ where w_i^l doesn't appear , respectively, for $1 \leq l \leq |L|, 1 \leq i \leq m^l$ and $1 \leq c \leq |C^l|$. Let $P(w_i^l) = \frac{N(w_i^l)}{NT}$ denote the probability of w_i^l to appear in a document, $P(\overline{w_i^l}) = \frac{N(\overline{w_i^l})}{NT}$ denote the probability of w_i^l not appearing in a document, $P(w_i^l, {C_c}^l) = \frac{N(w_i^l, {C_c}^l)}{NT}$ denote the probability of word w_i^l to appear in a document belonging to ${C_c}^l$ and $P(\overline{w_i^l}, {C_c}^l) = \frac{N(\overline{w_i^l}, {C_c}^l)}{NT}$ denote the probability of word w_i^l not appearing in a document belonging to class ${C_c}^l$, where $1 \leq l \leq |L|, 1 \leq i \leq m^l$ and $1 \leq c \leq |C^l|$. Then, for label l, where $1 \leq l \leq |L|$, the information gain of word w_i is

$$IG^l(w_i^l) = \sum_{c=1}^{|C^l|} \left(P(w_i^l, {C_c}^l) log\left(\frac{P(w_i^l, {C_c}^l)}{P(w_i^l)P({C_c}^l)}\right) + P(\overline{w_i^l}, {C_c}^l) log\left(\frac{P(\overline{w_i^l}, {C_c}^l)}{P(\overline{w_i^l})P({C_c}^l)}\right)\right) \quad (2.10)$$

Substituting the probabilities with their counts and having in mind that for each label label l, where $1 \leq l \leq |L|$, we want to find the top m^l words with the highest IG, but not their exact IG, then (6) can be rewritten as

$$IG^l(w_i^l) \sim \sum_{c=1}^{|C^l|} \left(N(w_i^l, {C_c}^l) log\left(\frac{N(w_i^l, {C_c}^l)NT}{N(w_i^l)N({C_c}^l)}\right) + N(\overline{w_i^l}, c_j) log\left(\frac{N(\overline{w_i^l}, {C_c}^l)NT}{N(\overline{w_i^l})N({C_c}^l)}\right)\right) \quad (2.11)$$

2.3. Machine Learning classifications

According to Dua [38]: "Machine learning (ML) is the computational process of automatically inferring and generalizing a learning model from sample data. In supervised machine learning, an algorithm is fed sample data that are labeled in meaningful ways. The algorithm uses the labeled samples for training and obtains a model. Then, the trained machine-learning model can label the data points that have never been used by the algorithm".

Below we give some of the widely used ML techniques with application to cyber security and corresponding use-cases. Other case-studies on all of the mentioned classifiers on this section are reported in [38] and [39].

2.3.1. K-Nearest Neighbor (KNN)

Is a widely used classification technique which is simple to implement, although it requires a lot of computation power and storage resources. It is simple since it doesn't require prior training, rather, given an observed instance $X = \{X_1, X_2, \dots X_f\}$ which we want to classify (label) and a dataset DS with fairly enough of already correctly classified instances, we want to find the instance(s) of DS that are closer to X and correspondingly label X as belonging to its closest instance(s) class in terms of a distance. However, in order to avoid biases, distances that do not properly find neighbors, and/or mistakes that might have happened while capturing (measuring) the instances X's features, when classifying X, we might consider to take into consideration several (say k) closest neighbors of X in DS, hence the name *kNN*. Afterwards the majority voting is applied to determine X's final vote. In order to avoid equal number of votes for different classes from the neighbors of X, it is desires k to be odd.

Let $Y^{r_i} = \left\{Y_1^{r_i}, Y_2^{r_i}, \dots, Y_f^{r_i}\right\}$ be a record of DS, where $1 \leq i \leq NT$ and NT is the number of transactions (records, instances) in DS. As a distance metric between X and Y^{r_i} can be considered several ones. The most used distance metric is the Euclidian distance, which is calculated as:

$$d(X, Y^{r_i}) = \sqrt{\sum_{y=1}^{f} (X_i - Y_y^{r_i})^2} \quad (2.12)$$

The squared Euclidean distance between X and Y^{r_i} is defined as:

$$d(X, Y^{r_i}) = \sum_{y=1}^{f} (X_i - Y_y^{r_i})^2 \quad (2.13)$$

The Manhattan distance between X and Y^{r_i} is defined as:

$$d(X, Y^{r_i}) = \sum_{y=1}^{f} |X_i - Y_y^{r_i}| \tag{2.14}$$

The cosine similarity between X and Y^{r_i} is defined as:

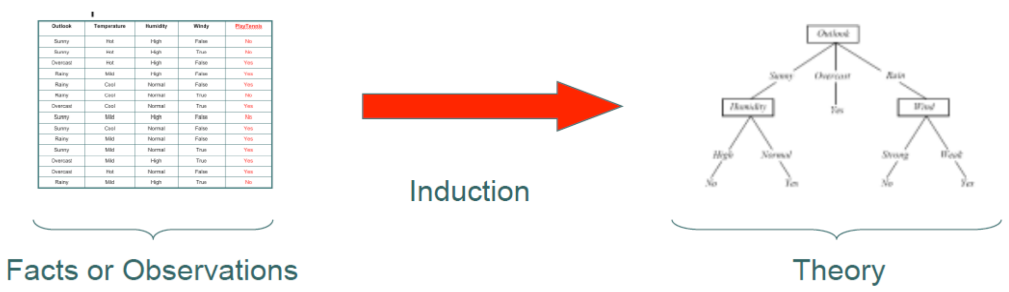
$$d(X, Y^{r_i}) = \cos(\alpha) = \frac{X \times Y^{r_i}}{|X||Y^{r_i}|} \tag{2.15}$$

where $X \times Y^{r_i}$ is the inner product of X and Y^{r_i}, and $|X|$ and $|Y^{r_i}|$ are their canonical (L2) forms, respectively. Besides those, there are others less used distance metrics such as Jaccard's, Minkowski's, Chebyshev's distance, etc.

Since X checks for all of the instances in DS, kNN it has a linear complexity with respect to the number of records in DS.

2.3.2. Decision Trees and Random Forests

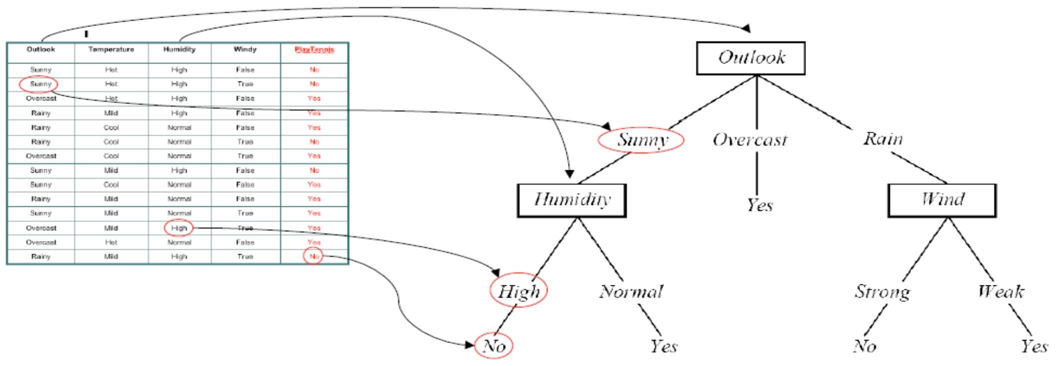
Decision trees are also one of the most frequently used machine Learning (ML) techniques, known also as decision trees. Decision trees (DT) are build based on an observation database (dataset) (Fig. 2.1.). After building the tree, those techniques are used to give prediction (classification) for a record (instance, raw) that has only its feature values, but not the class to which that particular record (instance) belongs to. Thus we want to classify (label) a newly observed instance which might not have been seen before as part of the dataset.



Outlook	Temperature	Humidity	Windy	PlayTennis
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

Fig.2.1. Building a decision tree out of observations (dataset) [27]

Starting from the root of the tree and guided by the value(s) of certain attribute(s), we follow a link that sends to another node and so on till we reach a leaf node which eventually contains the final classification for that record as it is shown in Fig.2.2. A tree is expressed as conjunction of disjunctions in terms of if-and-…-and-then rules.



Outlook	Temperature	Humidity	Windy	PlayTennis
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

Fig.2.2. Finding a binary tree classification guided by attributes and their values [42]

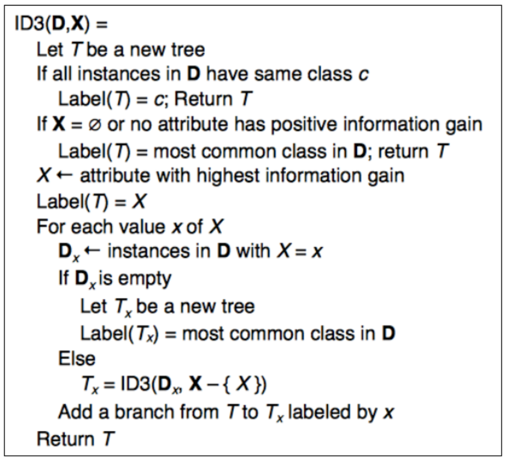
There are several ML binary tree techniques, which are similar and variations of each other. The most famous ones are J48, ID3 (proposed by Quinlan [40]), C4.5 (Quinlan [41]) and CART (Classification and Regression Tree). Here we will briefly give only ID3, whose algorithm (pseudocode) is given in Fig.2 5.

In order to properly build a DT, we introduce two notions (metrics) that will help us build the tree. One of them is the notion of entropy (the degree of uncertainty a system has) which is given with the formula (3.22)

$$H(S) = \sum_{i=0}^{N} -p_i log(p_i) \tag{2.16}$$

where p_i is the probability of appearance of a certain value for a certain feature (attribute) .

Another notion is that of the Information Gain that we introduced in Chapter 2.2. In this sense, according to the pseudocode in Fig.2.3, until there are no features (attributes) left without being selected, starting from the root of the DT, in each iteration we select the feature whose feature values introduce the highest IG are selected to be the root of the DT.



```
ID3(D,X) =
  Let T be a new tree
  If all instances in D have same class c
    Label(T) = c; Return T
  If X = ∅ or no attribute has positive information gain
    Label(T) = most common class in D; return T
  X ← attribute with highest information gain
  Label(T) = X
  For each value x of X
    $D_x$ ← instances in D with X = x
    If $D_x$ is empty
      Let $T_x$ be a new tree
      Label($T_x$) = most common class in D
    Else
      $T_x$ = ID3($D_x$, X – { X })
    Add a branch from T to $T_x$ labeled by x
  Return T
```

Fig.2.3. Pseudocode for the ID3 algorithm [40]

Random forest (RF) on the other hand is an algorithm that is made of multiple DT. Those DT are selected in such a way that they complement each-other to give better results. Since all of the DT algorithms have strong and week sides, the RF is designed to combine the strong sides of several decision tree algorithms. Since different trees might give different classification for an un-classified instance, RF uses the majority voting to give the final thought of its instance classification (labeling). In order to avoid imbalances that might appear due to different natures of individual trees in the RF, techniques such as boosting, bagging and others are used [42-43].

2.3.3. Machine Learning classifications based on linear algebra operations

In this chapter we introduce several classifiers which are based or can be expressed through linear algebra operations such as dot (inner) product of two vectors, matrix-vector product, single matrix-matrix product and cascading (sequential, one after another) matrix-matrix products. Those are SVM, LR and DNN. Also we will re-introduce NB and MNB classifiers, but this time expressed through linear algebra operations. In the following sub-chapters, we assume that the trained model of the corresponding ML algorithm already exists and we deal only with the classification (prediction) stage for unclassified queries. For some of the ML algorithms we give a brief overview on how the trained model is obtained

2.3.3.1 Naïve Bayes and Multinomial Naïve Bayes (revisited). Having in mind the notations for NB in Chapter 2.1.1, for the set of classes $C = \{C_1, C_2, \dots C_c\}$ let us define the training integer row-vector $C^{(j)}$ corresponding to class C_j as $C^{(j)} = \left\{ \lfloor KlogP(C_j) \rfloor, \mathop{}_{i=1}^{f}\left(\mathop{}_{m=1}^{|F_i|} \lfloor Klog(V_{m,F_i}|C_j) \rfloor, \right) \right\} =$ $\{\lfloor KlogP(C_j) \rfloor, \lfloor Klog(V_{1,F_1}|C_j) \rfloor, \lfloor Klog(V_{2,F_1}|C_j) \rfloor, \dots , \lfloor Klog(V_{|F_n|,F_n}|C_j) \rfloor\}$, which makes $C^{(j)}$ an $f + 1$ dimensional vector, where $1 \le j \le c$,. At its first index it has the *log* of the class probability - $\lfloor KlogP(C_j) \rfloor$, followed by all of the f conditional feature value probabilities - $\lfloor Klog(V_{m,F_i}|C_j) \rfloor, \mathrm{s.t.}\, 1 \le i \le f, 1 \le m \le |F_i|$- of the remaining f indexes in sequential order. The multiplication of the logs of the probabilities by a constant K and rounding them to the closest smaller integer ($\lfloor \cdot \rfloor$) is done for encryption purposes. Let we have an unclassified query $X' = \{X'_1, X'_2, \dots, X'_n\}$, where $X'_i \in F_i$. In similar way, we redefine the query vector X' as a binary row-vector $X = \left\{ 1, \mathop{}_{i=1}^{f}\left(\mathop{}_{m=1}^{|F_i|} V_{m,F_i} \right) \right\} =$ $\{1, V_{1,F_1}, V_{2,F_1}, \dots, V_{|F_n|,F_n}\} \cong \{1, X_1, \dots, X_f\}$, i.e. it has 1 at the first index followed by a sequential order of all of the *f* feature values of all feature sets. For all $X'_i \in F_i$ of the original query vector *X*' we put 1 (one) at the corresponding index of the redefined query and all other values are set to be 0 (zeros). If we define the trained model as an $c \times (f + 1)$ dimensional matrix, whose rows are all of the c training row-vectors - $C^{(j)}$s in sequential order, thus $M = \left[\{C^{(j)}\}_{j=1}^{c} \right]_{c \times (f+1)}$, then the classification of $C_M(X) = C_M(X')$ can be expressed as:

$$C_M(X) = \underset{1 \le j \le c}{argmax} [M \times X] \tag{2.17}$$

where the matrix column-vector multiplication returns a c dimensional column-vector which in its indexes contains the posterior probabilities of X to belong to the corresponding class C_j. The term $\underset{1 \le j \le c}{argmax}$ returns the maximum element of the resulting vector, which is the class with the highest posterior probability, hence the class label for X. A similar reasoning can be done for the MNB and NB case for textual datasets and queries, where the trained model matrix M contains the corresponding row-vector class probabilities -$C^{(j)}$s -in sequential order, obtained having in mind MNB logic (Chapter 2.1.2), whereas the query vector q_v remains the same.

2.3.3.2. Support Vector Machines (SVM). Can directly be used in systems with two classes (binary case) that are linearly separable and the separation is done with planes. Since there are many planes that separate the instances of the two classes, the separation is done (chosen) in such a way that there is a maximum gap between instances of the different classes on each side of the plane (Fig. 2.4). This is obviously the optimal separation of the two classes. The instance that are closer to the separation plane on both sides of it (i.e. of the different classes) make the so called support vectors, hence the name support vector machines (SVM). Let those support vectors be the instances (records) $\{Y^{r_+}, c_+\}$ belonging to one of the binary classes (denoted as c_+) and the other be $\{Y^{r_-}, c_-\}$ belonging to the other class (denoted as c_-), where $Y^{r_+} = \{{Y_1}^{r_+}, \dots, {Y_f}^{r_+}\}$ and $Y^{r_-} = \{{Y_1}^{r_-}, \dots, {Y_f}^{r_-}\}$ are the values of the support vectors they have for each of the f features (dimensions) of the dataset. Let the record $\{Y^{r_+}, c_+\}$ belong to a plane π_+ s.t. $W^T Y^{r_+} + b = 1$, where b is constant and W is the normal vector to the hyperplane π_+. This means that all the f dimensional records $X = \{X_1, \dots, X_f\}$ for which $W^T X + b \geq 1$ holds, i.e. they are on or above π_+, are labeled as belonging to class c_+. Similarly, let the records $\{Y^{r_-}, c_-\}$ belong to a plane π_- s.t. $W^T Y^{r_-} + b = -1$ where b is the same constant and W is also the normal vector to the hyperplane π_-. All the records $X = \{X_1, \dots, X_f\}$ for which $W^T X + b \leq -1$ holds, i.e. they are on or below π_-, are labeled as belonging to class c_-. We can re-write this as $y(W^T Y^{r_y} + b) \geq 1$, where $y = 1$ and $r_y = r_+$ if the class is c_+,while $y = -1$ and $r_y = r_-$ if the class is c_-. This also means that the distance between π_+ and π_- is $\frac{2}{|W|}$, where $|W|$ is the Normal Hesse form of W. Since we want to maximize the distance of the support vectors with the separating plane, that means we should minimize $|W|$. Thus, the plane that in the most optimal ways separates the two classes is derived by optimizing those formulas:

$$argmin(\frac{1}{2}|w|^2) \quad (2.18)$$

$$y(W^T Y^{r_y} + b) \geq 1 \quad (2.19)$$

The splitting plane, which is given by its plane equation $W^T X + b = 0$, afterwards is used to classify new instances $X = \{X_1, \dots, X_f\}$ by evaluating X into the plane formula. The result will be either a positive or a negative number, one for both of the classes.

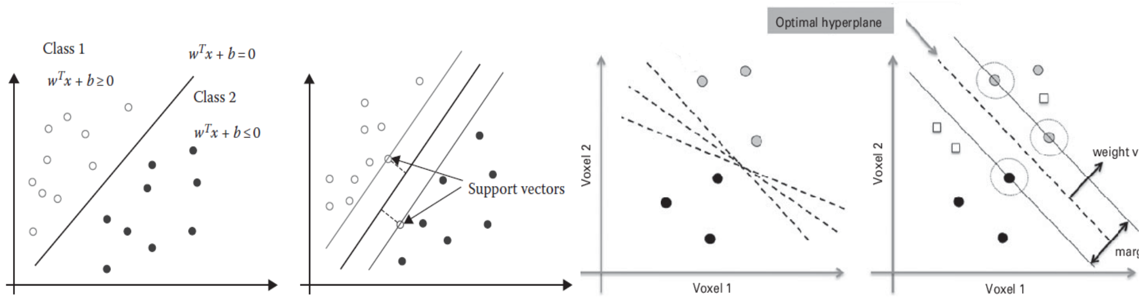


Fig.2.4. Choosing the right class separator plane using SVM [39, 45].

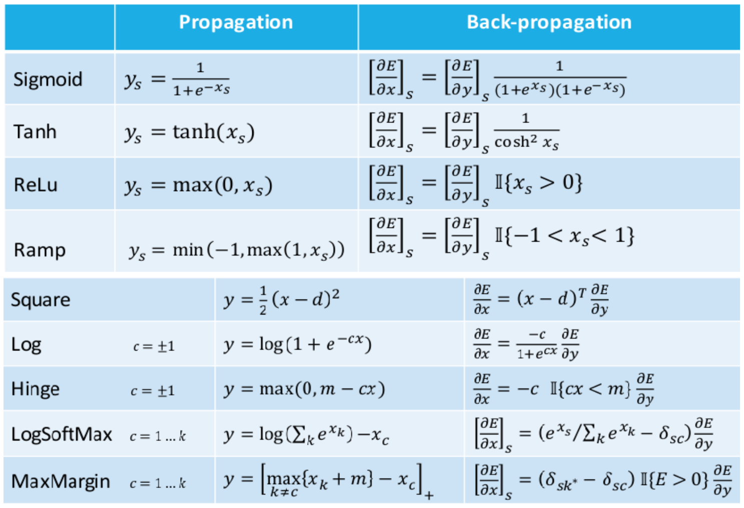
For the case when binary classes are not linearly separable, kernel tricks are used. Usually those kernels add an extra dimension which makes the classes again linearly separable and applies the same logic as it is shown above.

For the cases when there are several classes (more than two, say c classes), SVM can be used in two modes. In one mode each class is separated from every other class using the above logic. This means that for each pair of classes we have a classifier, which in total make for $c(c-1)/2$ plane classifiers. Since in some cases this is a lot, another approach is to have "one versus other classes" classifier. This means that in total we have c-1 planes [30].

In order to represent SVM multiclass classifiction in term os linear algebra (concretly matrix-vector product) operations, let us have f features, denoted as $F_1, \dots, F_f$ and c classes, $C = \{C_1, C_2, \dots, C_c\}$. Each of the c classes has its own trined $f + 1$ dimensional hyperplane $W^{(j)} = \left\{b^{(j)}, w_1^{(j)}, \dots, w_n^{(j)}\right\}$, for $1 \leq j \leq c$, that tends to maximize the gap between the closest instances (support vector machines) of the that class with the rest. In this sense the trained model can be expressed as rows of $W^{(j)}$s, thus $M = \left[\left\{W^{(j)}\right\}_{j=1}^{c}\right]_{c \times (f+1)}$. If the query vector X is expressed as a column vector $X = \left\{1, X_1, \dots, X_f\right\}$, where $X_i \in F_i$ for $1 \leq i \leq f$, then the classification of X - $C_M(X)$ is done using (2.17) [9-10], [43-44]

2.3.3.3. Logistic regression (LR). The trained model M and the user query X have the same construction as in 3.1.2, hence its classification is done using (2) again. LR differes from SVM only by the algorithm by which the trained model M is obtained [16], [21].

Table 2.1. Common perceptron activation functions [45]



		Propagation	Back-propagation
Sigmoid		$y_s = \frac{1}{1+e^{-x_s}}$	$\left[\frac{\partial E}{\partial x}\right]_s = \left[\frac{\partial E}{\partial y}\right]_s \frac{1}{(1+e^{x_s})(1+e^{-x_s})}$
Tanh		$y_s = \tanh(x_s)$	$\left[\frac{\partial E}{\partial x}\right]_s = \left[\frac{\partial E}{\partial y}\right]_s \frac{1}{\cosh^2 x_s}$
ReLu		$y_s = \max(0, x_s)$	$\left[\frac{\partial E}{\partial x}\right]_s = \left[\frac{\partial E}{\partial y}\right]_s \mathbb{I}\{x_s > 0\}$
Ramp		$y_s = \min(-1, \max(1, x_s))$	$\left[\frac{\partial E}{\partial x}\right]_s = \left[\frac{\partial E}{\partial y}\right]_s \mathbb{I}\{-1 < x_s < 1\}$
Square		$y = \frac{1}{2}(x-d)^2$	$\frac{\partial E}{\partial x} = (x-d)^T \frac{\partial E}{\partial y}$
Log	$c = \pm 1$	$y = \log(1 + e^{-cx})$	$\frac{\partial E}{\partial x} = \frac{-c}{1+e^{cx}} \frac{\partial E}{\partial y}$
Hinge	$c = \pm 1$	$y = \max(0, m - cx)$	$\frac{\partial E}{\partial x} = -c \ \mathbb{I}\{cx < m\} \frac{\partial E}{\partial y}$
LogSoftMax	$c = 1 \dots k$	$y = \log(\sum_k e^{x_k}) - x_c$	$\left[\frac{\partial E}{\partial x}\right]_s = (e^{x_s}/\sum_k e^{x_k} - \delta_{sc}) \frac{\partial E}{\partial y}$
MaxMargin	$c = 1 \dots k$	$y = \left[\max_{k \neq c}\{x_k + m\} - x_c\right]_+$	$\left[\frac{\partial E}{\partial x}\right]_s = (\delta_{sk^*} - \delta_{sc}) \mathbb{I}\{E > 0\} \frac{\partial E}{\partial y}$

2.3.3.4. Deep neural networks (DNN). One of the hot research areas recently have been DNN. They were designed to imitate (simulate) the way the human brain works. Namely, the human brain is consisted from small processing (neurons) that take input signals from other processing units, process them and send them to another processing unit. The same is with the elementary constituents of DNN that take some weighted input(s) from other processing units, do some linear or nonlinear transformation to it and then pass the output to another processing unit. These elementary processing units are called perceptrons (shown as circular shaped nodes in Fig. 2.5). The function that perceptrons use to transform the (weighted) input to an output is called the activation function. Table 2.1 gives the most common activation functions. In some cases, if the output is positive it is said that the perceptron fires. Due to the limitations of single perceptrons, layers of and later on multiple layer of perceptrons were introduced (Fig.2.5). Sometimes DNN can have hundreds if not thousands of layers, each with also hundreds or thousands of perceptrons.

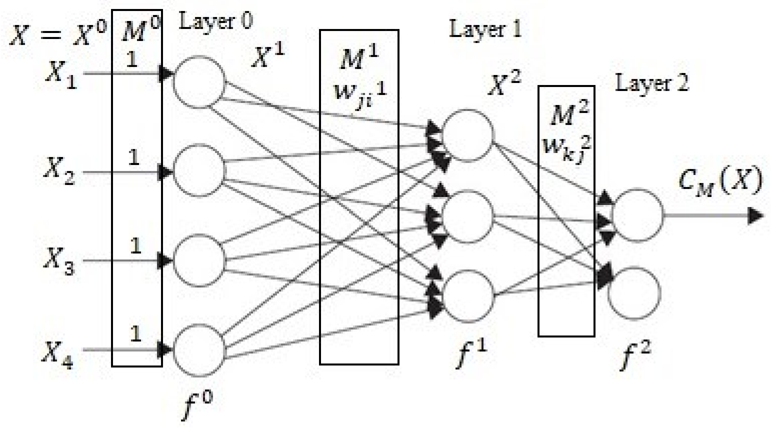


Fig.2.5. A multilayer deep neural network [39]

The DNN in Fig.6 has 3 layers. The input (first from the left) has 4 perceptrons, the middle layer has 3 and the output layer 2 perceptrons. Although there is a huge research going on with ANN and deep learning, still there is no clear policy on how to determine the adequate number of layers, the number of perceptrons in a layer, the activation function for perceptrons or even the number of inputs in the network.

As it can be seen from the DNN in fig.6, the input $X = \{X_1, X_2, X_3, X_4\}$ is fed to the four perceptrons of the input layer. Inside the perceptrons the activation functions (say one from the table 4) transforms the input into an output. For each input layer node (perceptron) there is a link to the next (hidden) layer node which has a weight (usually in decimal numbers) assigned to it. Initially those weights are randomly assigned and will get fixed during the training process. Those weights can be represented as a matrix, in which the rows are the inputs and columns the output (target) perceptron nodes, which in Fig.2.6 are denoted as M^1 matrix, with the corresponding weights (elements in the matrix) denoted as w_{ij}. For instance, the weight of the link that comes out from the second input node and goes to the 3rd node of the hidden layer is w_{23}. In the second (hidden) layer, all of the nodes some up the products of the output that the input layers generate with the weights. Then this sum of products is fed to the activation function of the hidden layer perceptron, which in turn comes up with an output that will be send to the proceeding layer, which in our case is the output layer. Of course, the links between the hidden and the output layer have also their weights, represented by the M^2 matrix. The M^2 matrix has j rows and k columns in Fig.2.5, which are the number of perceptrons (nodes) in the hidden and the output layer, correspondingly.

Let us define M^0 as a column vector which has f ones, thus its weights are all 1. Let the input layer be denoted as layer 0, let we have l layers and let the activation functions for each layer i be denoted as $f^i(\cdot)$, where $0 \le i \le l$. Apparently, the trained model M of our DNN is consisted of all the matrixes M^i and $f^i(\cdot)$, for $0 \le i \le l$. Let the output of layer i be a column vector denoted as X^{i+1} and let the input X also be denoted as X^0. Then, the final output of the DNN can be denoted as $C_M(X)$, which is actually the classification of our input query $X = X^0$ according to M, thus

$$C_M(X) = \prod_{i=0}^{l}\left(f^i(M^i \times X^i)\right) \tag{2.20}$$

Equation (2.20) is equivalent to (2.17) when the DNN has only layers 0 and 1 and their correpodning activation functions are linear, thus $f^0(x) = f^1(x) = x$.

$$\underset{\substack{1\le j\le c\\1\le k\le q}}{\boldsymbol{argmax}}\left(\overset{\overset{\boldsymbol{M}}{\longleftarrow f+1 \longrightarrow}}{\begin{bmatrix}\leftarrow \boldsymbol{C}^{(1)}\rightarrow\\ \leftarrow \boldsymbol{C}^{(2)}\rightarrow\\ \cdots\\ \leftarrow \boldsymbol{C}^{(c)}\rightarrow\end{bmatrix}} \times \overset{\overset{\boldsymbol{S}}{\longleftarrow q \longrightarrow}}{\begin{bmatrix}\uparrow & \uparrow & & \uparrow\\ \boldsymbol{X}^{(1)} & \boldsymbol{X}^{(2)} & \dots & \boldsymbol{X}^{(q)}\\ \downarrow & \downarrow & & \downarrow\end{bmatrix}}\right)$$
$$= \boldsymbol{C_M}(\boldsymbol{S}) = \left[\boldsymbol{C_M}(\boldsymbol{X}^{(1)})\ \boldsymbol{C_M}(\boldsymbol{X}^{(2)}) \dots \boldsymbol{C_M}(\boldsymbol{X}^{(q)})\right]$$

Fig.2.6. Illustration of our proposed multi-query classifications

2.3.4.5. Multi-query classifications. Since all of the above ML classification schemes use the same logic during the ML classification stage, which can be expressed in terms of a multiplication of a matrix with a column-vector, intuitively, a matrix-matrix multiplication can classify multiple queries at once. *Let* $S = \left[\left\{X^{(k)}\right\}_{k=1}^{q}\right]_{(f+1)\times q}$ be the $(f+1) \times q$ dimensional query matrix obtained by appending q column vector queries of dimension $n+1$, s.t. $X^{(k)} = \left\{1, X_1^{(k)}, \dots, X_f^{(k)}\right\}$, for $1 \le k \le q$. In that case, for SVM, LR, NB and MNB, as shown in Fig.2.6, classifying q queries at once can be expressed as

$$C_M(S) = \underset{1\le j\le c, 1\le k\le q}{argmax}[M \times S] \tag{2.21}$$

Similarly, for the DNN (2.20) can be written as:

$$C_M(S) = \prod_{i=0}^{l} \left(f^i(M^i \times S^i)\right) \tag{2.22}$$

2.4. Cryptographic primitives

In this section we give introduce some cryptographic primitives, definitions concepts and theorems that will be used in our schemes.

In order to achieve data privacy, in our research we are interested and deal only with cryptographic techniques, mainly due to their accuracy and efficiency. Those techniques are based on secure Two-Party-Computation (2PC), somewhat homomorphic encryption (allows limited number of mathematical and/or Boolean operations on ciphertexts), oblivious transfer (OT), private information retrieval (PIR), etc. The pioneering works of these techniques are Yao's circuits [47], ElGamal's [48] public crypto-system that supports additive homomorphic properties, Pailler's [49] crypto-system that besides the additive homomorphic property also allows a multiplication with a constant, the Goldwasser-Micali scheme [50] that enables secure XOR operations between two encrypted bits, etc. The secure multi-party computation (MPC) given in [51-52] is a generalization of 2PC to more than two parties. MPC however suffers from computation and communication cost, making it impractical for many real-case scenarios [53].

Somewhat homomorphic encryption (SWHE) schemes allow only a limited number of homomorphic additions and multiplications on the ciphertexts. Gentry's seminal work of [54], paved the way for the Fully Homomorphic Encryption schemes (FHE), which allows arbitrary number of additions and multiplications on the ciphertexts. This is done by introducing the bootstrapping (homomorphic re-encryption) technique applied on SWHE schemes. Further improvements of [54] were seen in [55-57], which made FHE suitable for practical applications, hence resulted in the development of actual libraries such as IBM's HElib [31] based on the BGV scheme from [55] and Microsoft's SEAL [59] based on both the SWHE FV presented scheme in [56] and some characteristics of the BGV scheme.

2.4.1. Public Somewhat Homomorphic Encryption schemes

Homomorphic encryption (HE) schemes allow for certain arithmetic or Boolean operations to be evaluated (done) over the ciphertexts without decrypting them (while the ciphertexts are still encrypted) [49], [54], [56], [57], [59]. So far, the strongest form of HE are Fully HE (FHE) schemes which allow unlimited numbers of homomorphic additions and multiplications over the ciphertexts. This is due to the computationally expensive technique known as bootstrapping (homomorphic evaluation of the decryption circuit), which as an output gives a re-encrypted ciphertext over which new homomorphic operations can be done. The first FHE scheme was proposed in 2009 [54], and over the years several others would follow [59]. All of the recent FHE scheme are based on the assumption of the hardness of Decision-RLWE (Ring Learning With Errors, Section IX), known to be resistant to quantum computer attacks [57], [59]. While the security of Somewhat HE (SWHE) schemes is also based on Decision-RLWE, they are a weaker variant of FHE in terms that they allow only a limited number of homomorphic multiplications (known as the circuit depth), but in the process they avoid the costly bootstrapping operation [56], [59]. However, in most of the real case scenarios, the circuit depth is known in advance, which allows for the encryption parameters to be set in a way that no bootstrapping will be needed. This makes SWHE an ideal choice over FHE. The plaintexts and the ciphertexts in FHE and SWHE schemes are polynomial rings with modulus $X^N + 1$ and their coefficients are integers modulo t and q, respectively, s.t. q $\gg t$. Thus, plaintexts belong to the ring $R_t = Z_t[X]/(X^N + 1)$ and ciphertexts to the ring $R_q = Z_q[X]/(X^N + 1)$. In [57] it has been shown that if the polynomial modulus of degree N can be expressed as a multiplication of N irreducible polynomials of degree one, which in turn are automorphic to each other, then, according to the Chinese Remainder Theorem (CRT), we can encode N integers in a single plaintext or ciphertext, one integer for each polynomial coefficient. A single homomorphic operation (addition or multiplication) over two ciphertexts encoded in such a way would result in simultaneous (parallel) component (index, slot) wise execution of the same operation over the encoded integers (Fig.2.7.a)-b)). This allows for a SIMD (Sıngle Instruction Multiple Data) fashion of homomorphically evaluating the ciphertexts, enabling massive efficiency improvements without extra cost. Furthermore, the automorphism of the irreducible polynomials of degree one allows for the encoded integers to change their places, mainly through rotating (shifting) them to the right or left (Fig.2.7c)). SWHE schemes allow SIMD operations between a ciphertext and a plaintext as well, where the result is always a ciphertext. Throughout the paper we assume that all SWHE encodings-encryptions of the plaintexts are done to support SIMD operations. For encoding/encryption purposes plaintexts and ciphertexts are denoted by having "*_p*" and "*_c*" at the end of their names, respectively. Briefly, common functions of a typical public SWHE scheme are [56, 59]:

- $(\boldsymbol{pk}, \boldsymbol{sk}) = \boldsymbol{KeyGen}(\boldsymbol{\lambda}, \boldsymbol{N}, \boldsymbol{t}, \boldsymbol{q})$. Generates a pair of public key cryptosystem (i.e. a public and corresponding secret key) according to the security parameter λ, polynomial modulus N and coefficient moduluses t and q for the plaintext and ciphertext, respectively.
- $\boldsymbol{m\_p} = \boldsymbol{Encode}(\boldsymbol{m\_v})$. SIMD encoding of an integer vector into a plaintext.
- $\boldsymbol{m\_c} = \boldsymbol{EncEncr}(\boldsymbol{m\_v})$. SIMD Encoding and encryption of an integer vector into a ciphertext
- $\boldsymbol{C\_c} = \boldsymbol{A\_c} + \boldsymbol{B\_c}$; $\boldsymbol{C\_c} = \boldsymbol{A\_c} + \boldsymbol{B\_p}$. SIMD addition of a ciphertext with another ciphertext or plaintext (Fig. 4a). The result is always a ciphertext.
- $\boldsymbol{C\_c} = \boldsymbol{A\_c} \times \boldsymbol{B\_c}$; $\boldsymbol{C\_c} = \boldsymbol{A\_c} \times \boldsymbol{B\_p}$. SIMD multiplication of a ciphertext with another ciphertext or plaintext (Fig.2.7b). The result is always a ciphertext.
- $\boldsymbol{B\_c} = \boldsymbol{Rotate}(\boldsymbol{A\_c}, \boldsymbol{R})$**.** Rotating a ciphertext for R slots (indexes). If $R > 0$ rotations are done to the right, otherwise to the left (Fig.2.7c)).
- $\boldsymbol{m\_v}$=$\boldsymbol{Decode}(\boldsymbol{m\_p})$; Decoding a plaintext into an integer vector
- $\boldsymbol{m\_v}$=$\boldsymbol{DecrDec}(\boldsymbol{m\_c})$: Decrypting then decoding a ciphertext into an integer vector

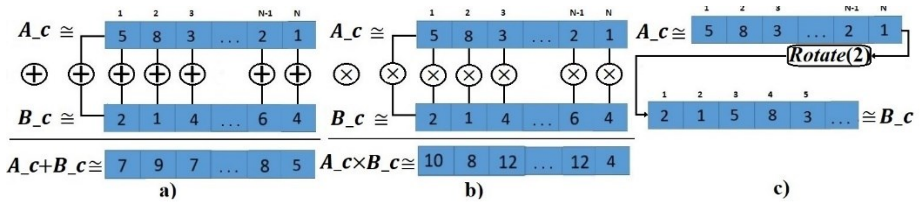


Fig.2.7.Illustration of SWHE SIMD a) addition, b) multiplication and c) Rotation for 2 slots

Let $A\_v = \{a_1, \dots, a_N\} = \{(a_i)_{i=1}^{N}\}$, $B\_v = \{b_1, \dots, b_N\} = \{(b_i)_{i=1}^{N}\}$ be integer vectors and their corresponding SIMD encoded&encrypted ciphertexts $B\_c = EncEncr(A\_v)$ and

$B\_c = EncEncr(B\_v)$, respectively. Let $R\_v = \{R_1, \dots, R_N\} = \{(R_i)_{i=1}^{N}\}$ and $h\_v = \{h_1, \dots, h_N\} = \{\{(h_i)_{i=1}^{N}\}\}$ be random integer vectors s.t. $R_i > 0, |h_i| < R_i$ for $1 \le i \le N$, and let $R\_p = Encode(R\_v), h\_p = Encode(h\_v)$ be their respective SIMD encoding into plaintexts. Then $C\_c = \big((A\_c - B\_c) \times R\_p\big) + h\_p$ is the SIMD secure comparison of the index-wise elements of A_v and B_v, firstly proposed in [18] and also elaborated in Chapter 6.3.4. Namely, let $C\_v = DecrDec(C_c) = \{c_1, \dots, c_N\} = \{(c_i)_{i=1}^{N}\} = \left\{\left(c_i = (a_i - b_i) \times R_i + h_{i_i}\right)_{i=1}^{N}\right\}$, if $c_i = (a_i - b_i) \times R_i + h_i > 0$ then $a_i > b_i$, otherwise $a_i < b_i$ for $1 \le i \le N$.

2.4.2. Security definitions, concepts and theorems

Definition 1: Decision-LWE: for a security parameter λ, let we sample $s \leftarrow U_q^{m\times 1}$, $a \leftarrow U_q^{n\times m}$, $e \leftarrow \chi_q^{n\times 1}$, $c \leftarrow U_q^{n\times 1}$, where U is the uniform distribution and χ is the discrete Gaussian distribution. Decision-LWE is the problem to distinguish between $(a, a \cdot s + e)$ and (*a, c*). [56, 57]**.**

Definition 2: Decision-RLWE: Generalizing LWE for rings [56, 57]**.**

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 [56, 57]**.**

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, respectively, an adversary cannot distinguish which ciphertext belongs to which plaintext [60].

Definition 2.2: Secure Multi-Party Computation (SMC) under the semi-honest model for deterministic functions: Let we have p parties, $P_1, \dots, P_p$, with the corresponding private inputs $x_1, \dots, x_p$ and let $\bar{x} = (x_1, .., x_p)$. With a certain security parameter λ let them execute protocol Π at the end of which each P_i gets the corresponding output $O_{P_i}^{\Pi}(\lambda, \bar{x})$ for $1 \leq i \leq p$, thus the global output is $O^{\Pi} = \left\{ \left(O_{P_i}^{\Pi}(\lambda, \bar{x}) \right)_{i=1}^{p} \right\}$. Let the view of P_i be $V_{P_i}^{\Pi}(\lambda, \bar{x}) = \left\{ \left(m_j^{P_i} \right)_{j=1}^{t} \right\}$, where $m_j^{P_i}$ are the messages that P_i receives while executing Π. We say that Π is a secure MPC protocol under the semi-honest model if there exists a simulator (function) s.t. $S_{P_i}^{\Pi}\left(\lambda, x_i, O_{P_i}^{\Pi}(\lambda, \bar{x})\right) \cong_c V_{P_i}^{\Pi}(\lambda, \bar{x})$, where $\cong_c$ stands for computational indistinguishability against a probabilistic polynomial time adversary [60].

Theorem 2.1: Modular Sequential Composition Theorem: Let Π be a protocol that sequentially calls $\Pi_1, \dots, \Pi_{\Pi}$. If $\Pi_1, \dots, \Pi_{\Pi}$ are SMPC protocols under the semi-honest model, then Π is also.

Proof: Given in [60] ■

Chapter 3

RELATED WORK AND THE STATE-OF-THE-ART

In this Chapter we provide an overview of the related research related to secure feature selection, secure ML training and secure ML classifications

3.1. Secure feature selection

Early secure feature selection schemes rely mostly on the *secure sum* protocol [22-26] as their building block. In *secure sum* n participants, denoted as $P_1, \dots P_n$, securely compute the sum of their corresponding private integer inputs, $i_1, \dots i_n$, assuming an existence of a ring network topology between them, Without loss of generality, the first participants adds a random number R to his input and passes it to the second one, which in turn adds his private input to the received sum and passes the result to the third participants, and so on, until the first participant receives the randomized sum $(R + \sum_{i=1}^{n} i_i)$, subtracts the random R from it and broadcasts the result to all the other participants. In this sense, while executing the protocol, P_t receives $(R + \sum_{i=1}^{t-1} i_i)$ from P_{t-1} and after adding its private input transmits $(R + \sum_{i=1}^{t} i_i)$ to P_{t+1}, where $1 \leq t \leq n$. This makes the communication cost of *secure sum* to be $n \cdot \sum f$ transmissions and $\sum f$ broadcasts, where $\sum f$ is the total number of feature values for which we have to find the sum over n participants. If P_{t-1} and P_{t+1} collude, they can retrieve P_t's private input i_t by subtractingP_t's output and input, thus

$(R + \sum_{i=1}^{t} i_i) - (R + \sum_{i=1}^{t-1} i_i) = i_t$. Also, an eavesdropper that can listen to all communicating channels can retrieve the private input of all the participants. Another drawback of *secure sum* is that it suffers from high communication overhead when f is large, which is the case with text classification datasets that are known to have hundreds of thousands of features (words, tokens), thus making *secure sum* highly impractical. Furthermore, secure sum doesn't work in the secure two party computation (2PC) scenarios.

Among the first schemes to address the problem of secure feature selection is [22]. It uses the *secure sum* protocol to calculate features' misclassification gain [22]. To avoid the collision, based on an assumption of the number of colluding participants, they came up with a metric that assigns a certain degree of collusion probability to participants and propose for each of them to operate in a safe (non-colluding) neighborhood according to a certain threshold. However, such a solution does not guarantee that a collusion will indeed be avoided. Also, it increases the already high communication overhead. In [23] they use the Harsanyi-Farrand-Chang [23] metric for feature selection which, for each feature value, needs a few invocations of *secure sum* to find intermediate metrics such as the correlation or the covariance, without addressing any of the drawbacks of *secure sum*. In [24] they test several metrics for feature selection and in the process provide a trade-off between the privacy and the accuracy of the trained model. In [25] each participant splits his private input into n shares such that their sum is equal to the participant's private input and sends one share to each of the other participants. Afterwards all of the participants locally sum up the received shares from others and invoke secure sum to find the final result. While this solves the collusion attack, it introduces an overhead of $n \cdot (n-1)$ transmissions to the high communication cost of a single *secure sum* invocation. After giving a brief literature review on the topic, [26] proposes ides for a few secure feature selection schemes without engaging into implementation details. Among others, [26] inherits all of the drawbacks of *secure sum* since it is supposed to use it as its main building block. None of [22-26] solved the *secure sum*'s shortcomings of an eavesdropper that can intercept all of the communications, of the inability to deal with 2PC, and of the high communication overhead, especially knowing that the metrics that they propose need several invocation of *secure sum* for a single feature values. In this context, [22-26] use a total of several hundreds of thousands of rounds (interactions) compared to only few that our protocol uses to reach the same goal.

Table 3.1. Properties among different schemes dealing with secure feature selection

Properties \ Schemes	[22]	[23]	[24]	[25]	[26]	[27]	[28]	Our
Privacy of the input features	✕	✕	✕	✕	✕	✕	✕	✓
Privacy of the input features' values	✓	✓	✓	✓	✓	✓	✓	✓
Security and privacy of intermediate results	✓✕	✓✕	✓✕	✓✕	✓✕	✓	✓✕	✓
Privacy for the output (selec. feat.)	✕	✕	✕	✕	✕	✕	✓	✓✕
Data Confidentiality, Integrity and Authenticity. Protocol consistency (blockchain) for interactions	✕	✕	✕	✕	✕	✕	✕	✓
Computing on plain in edge to avoid costly homomorphic operat.	✓	✓	✓	✓	✓	✓	✕	✓
Deals with more than two Edge Dataset Owners (EDOs)	✓	✓	✓	✓	✓	✕	✕	✓
Fault tolerance (byzantine failure) of up to $n-3$ out of n EDO	✓✕	✕	✕	✓	✕	NA	✓✕	✓
Allows up to $n-2$ out of n EDO collisions	✓✕	✕	✕	✓	✕	NA	✓✕	✓
Uses a centralized server(s) to avoid communication overhead	✕	✕	✕	✕	✕	NA	✓	✓
Avoids Using multiple (more than two) semi-honest non colluding servers	✓	✓	✓	✓	✓	✓	✕	✓
Resistant to eavesdropping	✕	✕	✕	✕	✕	✓	✓	✓
Enables 2PC (i.e. 2 DOs)	✕	✕	✕	✕	✕	✓	✓✕	✕
Applicable to the post quantum world (resistant to quantum computers)	✕	✕	✕	✕	✕	✕	✕	✓
cost does not dependent on the total number of records among n EDOs	✕	✕	✕	✕	✕	✕	✕	✓
Multi-label multi-output EDOs	✕	✕	✕	✕	✕	✕	✕	✓
Deals with both horizontally and vertically partitioned datasets	✕	✕	✕	✕	✕	✕	✕	✓

Avoiding *secure sum*, [27] uses Paillier's additive homomorphic encryption scheme [49] to introduce a 4 round 2PC (relatively high for a 2PC in secure ML classification scenarios) and doesn't address the scenario with multiple datasets. The secure feature selection scheme in [28] uses secret sharing for multiple dataset owners to share their data among three non-colluding servers (3PC), a scenario which is not easily feasible in reality, especial knowing that even if two of them collude can reveal the private dataset input values. Furthermore, it has a high interaction rate and knowing that each feature value is processed independently, makes it have a high communication overhead. Depending on to whom the trained model is shown, [28] can also deal with 2 dataset owners when the owner of the trained model is not one of the dataset owners, which rarely happens in practice.

None of the above schemes addresses the issues of authenticity, integrity or protocol flow consistency, i.e. blockchain technology. Table 3.1 summarizes security, privacy and efficiency of properties among several schemes.

3.2. Secure machine learning training

The first scheme to address the issue of privacy preserving (PP) NB training is due to Kantarcıoğlu et.al. in [61]. As it is the case with almost all of the earlier schemes, it exclusively deals with training and doesn't address the privacy preserving classification problem. In order to find the class and the joint class-value frequencies (counts) among all dataset owners, it uses the secure integer sum protocol explained in Chapter 3.1, thus inheriting all of the disadvantages of it. In the same paper, those attacks were avoided by splitting each private integer into integer shares (such that when summed up they give the private value) and each share then follows a different route while executing the secure sum protocol. However, both of them didn't address the privacy of the trained model and work only with three and more dataset owners. In [62] the same group uses the secure lnx algorithm proposed in [63] for training purposes, but it has a high communication cost. Those drawbacks were partially removed in [64] by utilizing a version of the additive homomorphic ElGamal scheme, where owners encrypt and send their data to be aggregated by a central server, removing in the process the communication overhead of the decentralized environments of the previous ones, but the final trained model is leaked again. Yi et.al [65] use the Paillier scheme, the secure lnx algorithm of [63] and two non-colliding servers to hide the final trained model, however, it re-trains the model for every query, which makes it rather inefficient.

Table 3.2. Properties among different schemes dealing with secure training of NB models

Properties \ Schemes	[24] (sec sum)	[24] (sec share)	[25] (sec log)	[32]	[21]	[33]	[23]	[36]	Our
Multiple EDOs	✓	✓	✓	✓	✓	✓	✓	✓	✓
Flexible (less than three EDO) during training	✗	✗	✓	✓	✗	✓	✓	✓	✓
Training comput. cost not dependent on dataset size	✓	✓	✓	✓	✗	✗	✗	✗	✓
Privacy for all Dataset(s) paramet. during training	✓	✓	✓	✓	✓	✓	✓	✓	✓
Total trained model security during training	✗	✗	✗	✗	✗	✗	✗	✗	✓
Resistant to Collusion attack between two or more EDOs	✗	✓	✓	✓	✓	✓	✓	✓	✓
Resistant to eavesdropping	✗	✓	✓	✓	✓	✓	✓	✓	✓
Central server(s) while training (efficient comm.)	✗	✗	✗	✓	✓	✓	✓	✓	✓
Avoiding Unnecessary retraining for each query	✓	✓	✓	✓	✗	✓	✓	✓	✓
Multiclass datasets	✓	✓	✓	✓	✓	✓	✓	✗	✓
Avoids multiple public key pairs or proxy re-encryptions	✓	✓	✓	✓	✓	✓	✗	✗	✗
The scheme is resistant to quantum computers	✗	✗	✗	✗	✗	✓	✓	✓	✓
Multi-label multi-output EDOs	✗	✗	✗	✗	✗	✗	✗	✗	✓
Deals with both horizontally and vertically partitioned datasets	✗	✗	✗	✗	✗	✗	✗	✗	✓

Those by Liu et al. in [66] and [67] are among some of the rare papers to address the issues of both training and classification in privacy preserving fashion of Naïve Bayes models. While [66] suffers from being interactive during the classification, [67] has only one round and hence a better communication efficiency. However, they both suffer from the costly process of proxy re-encryptions, bootstrapping and the lack of local pre-processing in the plain at the owners' location during the training phase, which hurts the overall performances. Also, [67] suffers from doing unnecessary costly homomorphic multiplications due to not using logs of probabilities (which convert products into sums as shown in 3.1). Li et al. [68] has similar properties to [67], however it lacks the details and experimentation results.

Table 3.2 summarizes security, privacy and efficiency of properties among several schemes which deal with secure training of NB or MNB models.

3.3. Secure machine learning classifications

In the abundance of privacy preserving training, the lack of having a privacy preserving classification protocols was realized by Bost et. al. in [53]. They use the additive homomorphic properties of Paillier and a secure argmax protocol (which in turn uses a multi-round secure comparison protocol with two public encryption schemes) to perform secure classification. Without addressing the training part, Park et. al [69] also dealt with the classification problem, by proposing a one-round scheme to overcome the multi-round communication overhead of [53]; but in the process it uses heavy inefficient FHE computations. Li et.al. [70] uses Goldwasser-Micali XOR homomorphic encryption scheme [50] and 3 Paillier public keys as well as several PIR invocations to get the needed class and value-class probabilities from the server and finally uses the secure argmax protocol of [53] to find the final classification. Gao et al. [71] uses parallel OT invocations and the Paillier scheme's additive homomorphic properties to find the class and value-class probabilities and then uses the secure comparison protocol of [15] to get the final result; however, it works with only two classes, unlike the other protocols which enable multi-class classification. While [66] suffers from being interactive during the classification, [67] and [72] suffer from doing unnecessary costly homomorphic multiplications due to not using logs of probabilities (which convert products into sums as shown in 3.1). Li et al. [68] has similar properties to [67], however it lacks the implementation details and experimentation results.

Table 3.3 summarizes security, privacy and efficiency properties among several schemes which deal with secure classifications for different ML classifiers.

Table 3.3. Comparisons of properties among different secure and private classification schemes

SCHEMES / **PROPERTIES**	[53]	[66]	[67]	[68]	[69]	[73]	[74]	[75]	[71]	[72]	[76]	[77]	[50]	[51]	**Our**
Trained model security (privacy)	✓	✓	✓✗	✓✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Query security (privacy	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓
Classification result privacy	✓	✓	✗	✓	✓	✓✗	✗	✓	✓	✓	✓✗	✓	✓	✓	✓
No loss of accuracy	✓	**NR**	✗	**NR**	✓	**NR**	✓	✓	✓	✓	**NR**	**NR**	✗	✓	✓
Flexibility (e.g. server vs user centric)	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓
Avoiding Unnecessary retraining for each query	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Non-interactive classification (exactly one round)	✗	✗	✓	✓	✓	✗	✓	✗	✗	✗	✓	✗	✗	✗	✓
Resistant to STC attack [34]	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✓
Simultaneous classification (packing) of multiple queries for higher throughput	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✓	✗	✗	✓
Multiclass algorithms	✓	✓	✗	✓	✓	✓	✗	✓	✗	✓	✓	✓	✓	✓	✓
Avoids multiple public key pairs or proxy re-encryptions	✗	✗	✗	✗	✓	✓	✓	✗	✗	✓	✓	✓	✓	✓	✓
Uses logs of probabilities instead of plain probab. avoids costlier multiplicat. in favor of additions during NB)	✓	✗	✗	✗	✓	✗	✗	✓	✓	✗	✓	✓	✗	✗	✓
Deals with malicious users during classification	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓
The scheme is resistant to quantum computers	✓✗	✓	✓	✓	✓	✓	✓	✗	✗	✓	✓	✓	✓	✓	✓
Avoids multiple non-colluding servers (i.e. has exactly only one such server) during classification	✓	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓✗

✓=presence; ✓✗=partial presence; ✗ = absence; NR = Not reported

Chapter 4

SECURE AND PRIVATE FEATURE SELECTION

Definition 4.1: Feature selection is the process of reducing the dimensions of a feature set F of a labeled dataset DS into SF according to an algorithm $\mathcal{A}$, thus $SF = \mathcal{A}(\text{DS}, F)$.

In this chapter, we introduce the security, privacy and efficiency goals (requirements) for our secure feature selection protocols. We proceed with the constituents (participants) of our secure feature selection schemes, their adversary models. Also, we provide a brief flow of the protocols, which will be elaborated in more details in the later subchapters of this chapter. We conclude the chapter by experimentally evaluating and comparing our schemes with the related research, which show that our schemes outperform the few state of the art ones for several times in terms of computation and communication costs. We conclude the chapter by proving the security of our protocols under the semi-honest model. All the necessary background information and notations for this chapter was given in Chapter 2.

4.1. Introduction

The security and privacy requirements for our secure feature protocols are:

- **Privacy of the input features**. We achieve this by randomly permuting the hashes of the input values (words, tokens)
- **Privacy of the input features' values** (counts, frequencies, etc.)

- **Security and privacy of intermediate results**. We keep the intermediate results of all of our protocols secure and private since they might be used as a trapdoor for total or partial leakage of the input or the output of the corresponding protocol.
- **Partial privacy for the output of the top *m* selected features**. The output cannot be totally private since it is needed as an entry point (input) for the secure classification stage when clients prepare their queries in accordance to the selected features. However, the selected features can be kept private for the secure training protocol.
- Have other properties related to secure feature selection mentioned in Table 3.1

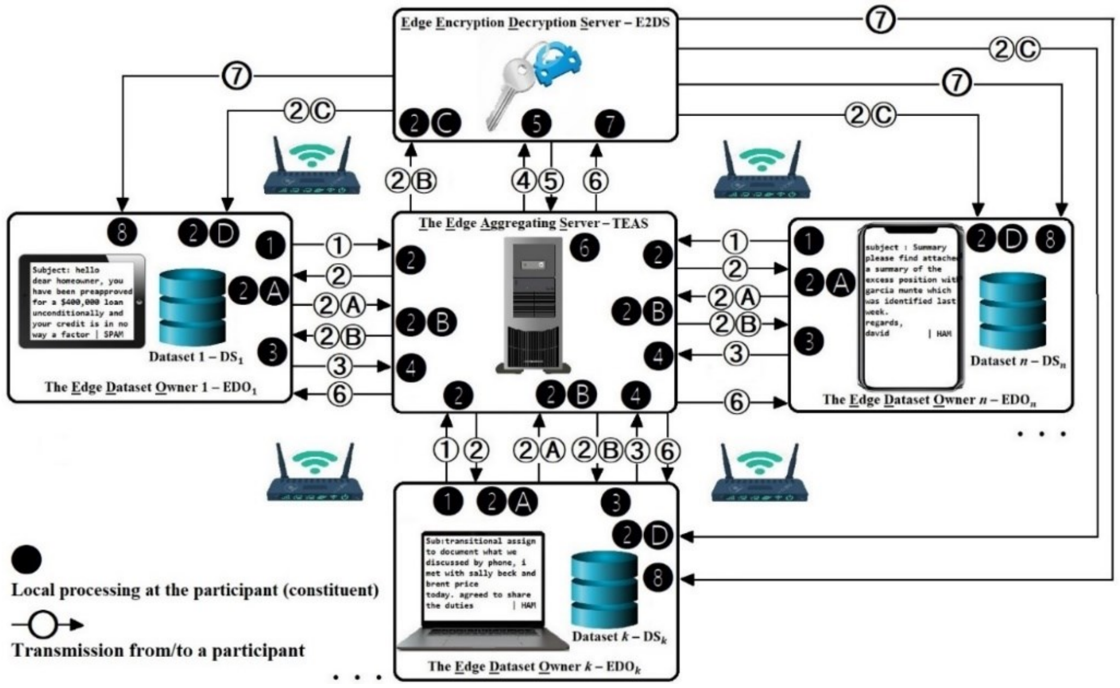
4.2. System architecture, adversary models and protocol flows

Our participants for the secure feature selection protocols are: **1) EDO** (The Edge Dataset Owner) - We have n such EDOs in our system, denoted as EDO_k, each owns a dataset DS_k, where $1 \leq k \leq n$, that they are willing to use for training ML models in a secure and private fashion. **2) TEAS** (The Edge Aggregating Server): a server used to do the bulk of the proposed protocols' homomorphic computation. **3) E2DS** (The Edge Encryption Decryption Server): It's the only participant in the system that has a pair of public keys with SWHE properties (Chapter 2.4). All the data that are homomorphically evaluated in our protocols are encrypted using E2DS' public key, thus it's the only one that can decrypt them. All of them are illustrated in Fig.4.1.

Adversary models: All the participants are assumed to be in the passive **semi-honest** (honest but curious) **model**, which means that they follow the protocol but on the background they try to infer some private data which they are not supposed to. A formal definition of the semi-honest model is given in Chapter 2.4.1. We assume that TEAS and E2DS don't collude. Out of n EDOs, our environment setting allows for up to $n-3$ EDO failures and up to $n-2$ collusions without jeopardizing the privacy of the remaining and non-colluding EDOs . The motivations for participant to behave in the described manners are given in [8-18].

Protocol flows at-a-glance: Each interaction in each protocol is marked by the interacting participant adding its own block with the corresponding data into the blockchain (Fig.4.2). All the participants have a pair of public/secret keys used for signing their corresponding blocks of the blockchain and for secure communication. Additionally, E2DS has a pair of public/secret key with SWHE properties (Chapter 2.4.1). All of these key pairs form the *KeySet* set. While we designed our protocols having in mind primarily binary textual datasets, they are also applicable to non-textual ones and can be easily generalized to multi-class scenarios. This choice was done for simplicity and benchmark (comparison) purposes with the related research.

***GRPV* (Generate Random Permutation Vector):** it is a secure multi-party computation (SMC) protocol through which EDOs agree on a random permutation of their hash bits of their private words, needed for the secure feature selection protocol.

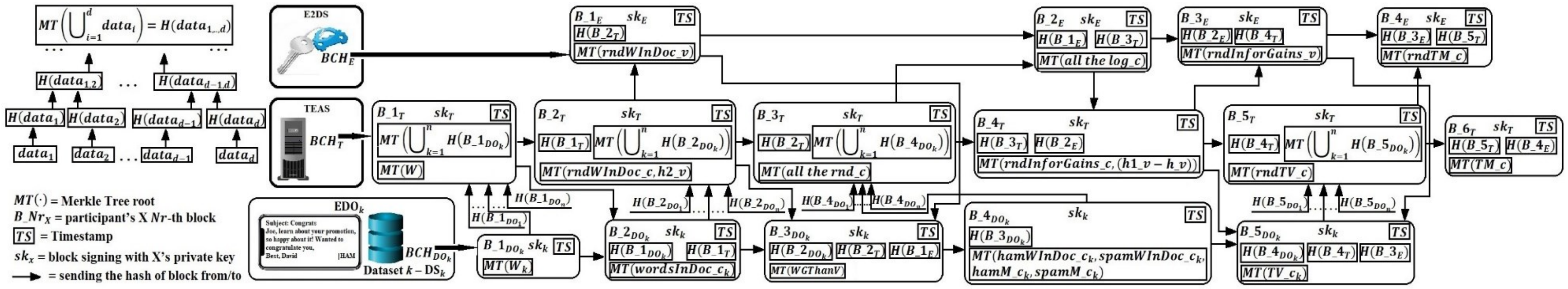

Edge Encryption Decryption Server – E2DS
The Edge Aggregating Server – TEAS
Dataset 1 – DS1
The Edge Dataset Owner 1 – EDO1
Dataset n – DSn
The Edge Dataset Owner n – EDOn
Dataset k – DSk
The Edge Dataset Owner k – EDOk
Local processing at the participant (constituent)
Transmission from/to a participant

Subject: hello
dear homeowner, you
have been preapproved
for a $400,000 loan
unconditionally and
your credit is in no
way a factor | SPAM

subject : Summary
please find attached
a summary of the
excess position with
garcia munte which
was identified last
week.
regards,
david | HAM

Sub:transitional assign
to document what we
discussed by phone, i
met with sally beck and
brent price
today. agreed to share
the duties | HAM

Fig. 4.1. Protocol flows for secure feature selection (*secFS-S1* and *secF-S2*)



$MT\left(\bigcup_{i=1}^{d} data_i\right) = H(data_{1..d})$
$H(data_{1,2})$
$H(data_{d-1,d})$
$H(data_1)$
$H(data_2)$
$H(data_{d-1})$
$H(data_d)$
$data_1$
$data_2$
$data_{d-1}$
$data_d$
$MT(\cdot)$ = Merkle Tree root
B_Nr_X = participant's X Nr-th block
$\boxed{TS}$ = Timestamp
sk_x = block signing with X's private key
= sending the hash of block from/to
E2DS
BCH_E
TEAS
BCH_T
EDO$_k$
Subject: Congrats
Joe, learn about your promotion, so happy about it! Wanted to congratulate you,
Best, David
HAM
BCH_{DO_k}
Dataset k – DS$_k$
B_1_E sk_E TS $H(B\_2_T)$ $MT(rndWInDoc\_v)$
B_2_E sk_E TS $H(B\_1_E)$ $H(B\_3_T)$ $MT(all\ the\ log\_c)$
B_3_E sk_E TS $H(B\_2_E)$ $H(B\_4_T)$ $MT(rndInforGains\_v)$
B_4_E sk_E TS $H(B\_3_E)$ $H(B\_5_T)$ $MT(rndTM\_c)$
B_1_T sk_T TS $MT\left(\bigcup_{k=1}^{n} H(B\_1_{DO_k})\right)$ $MT(W)$
B_2_T sk_T TS $H(B\_1_T)$ $MT\left(\bigcup_{k=1}^{n} H(B\_2_{DO_k})\right)$ $MT(rndWInDoc\_c, h2\_v)$
B_3_T sk_T TS $H(B\_2_T)$ $MT\left(\bigcup_{k=1}^{n} H(B\_4_{DO_k})\right)$ $MT(all\ the\ rnd\_c)$
B_4_T sk_T TS $H(B\_3_T)$ $H(B\_2_E)$ $MT(rndInforGains\_c, (h1\_v - h\_v))$
B_5_T sk_T TS $H(B\_4_T)$ $MT\left(\bigcup_{k=1}^{n} H(B\_5_{DO_k})\right)$ $MT(rndTV\_c)$
B_6_T sk_T TS $H(B\_5_T)$ $H(B\_4_E)$ $MT(TM\_c)$
$H(B\_1_{DO_1})$ $H(B\_1_{DO_n})$
$H(B\_2_{DO_1})$ $H(B\_2_{DO_n})$
$H(B\_4_{DO_1})$ $H(B\_4_{DO_n})$
$H(B\_5_{DO_1})$ $H(B\_5_{DO_n})$
$B\_1_{DO_k}$ sk_k TS $MT(W_k)$
$B\_2_{DO_k}$ sk_k TS $H(B\_1_{DO_k})$ $H(B\_1_T)$ $MT(wordsInDoc\_c_k)$
$B\_3_{DO_k}$ sk_k TS $H(B\_2_{DO_k})$ $H(B\_2_T)$ $H(B\_1_E)$ $MT(WGThanV)$
$B\_4_{DO_k}$ sk_k TS $H(B\_3_{DO_k})$ $MT(hamWInDoc\_c_k, spamWInDoc\_c_k, hamM\_c_k, spamM\_c_k)$
$B\_5_{DO_k}$ sk_k TS $H(B\_4_{DO_k})$ $H(B\_4_T)$ $H(B\_3_E)$ $MT(TV\_c_k)$

Fig.4.2. Illustration of generating the blocks of the vertically partitioned distributed public ledger (blockchain) while executing protocols *secFS-S1*, *secFS-S2* and *secT*.

***secFS*: (secure feature selection – stage 1 and 2):** Illustrated in Fig.4.1. Simultaneously, all of the n EDOs find W_k, the set of hashes of the local unique words (features) of their local textual datasets, where $1 \leq k \leq n$, ❶ and send them to TEAS ①. TEAS finds their union (the global set of hashes of unique words) $W = \cup_{k=1}^{n}\left[Dec_{sk_T}(W_k)\right]$❷ and broadcasts it to all EDOs ②. Using each word of W as an index entry, EDOs simultaneously construct the corresponding ciphertexts of the word counts (appearances) in ham and spam documents in the local dataset, i.e. $hamWInDoc\_c_k, spamWInDoc\_c_k$, then they construct the replicated ciphertexts of the number of ham and spam documents in the local datasets, $hamM\_c_k, spamM\_c_k$, ❸ and send them to TEAS ③. TEAS finds the global number of ham and spam documents, as well as the global ham and spam counts for each word of W by homomorphically summing up the locally constructed ciphertexts by EDOs, i.e. $hamWInDoc_c = \sum_{k=1}^{n}\left(hamWInDoc_{c_k}\right), hamM_c = \sum_{k=1}^{n}\left(hamM_{c_k}\right), spamWInDoc\_c = \sum_{k=1}^{n}(spamWInDoc\_c_k)$ and $spamM\_c = \sum_{k=1}^{n}(spamM\_c_k)$, then proceeds to find the terms needed to calculate the information gains of each word, randomizes those terms ❹ and sends them to E2DS ④. E2DS decrypts those randomized terms, uses them to compute the vector of the randomized logarithmic terms of the information gains for each word, encrypts the randomized logarithmic terms ❺ and sends them back to TEAS ⑤. TEAS homomorphically removes the randomizations of the logarithmic terms and finds the information gain data for each word in the corresponding index ($inforGains\_c$), adds some randomizations to get $rndInforGains\_c$ and sends it E2DS ❻ while broadcasts the randomizing numbers to each EDO ⑥. E2DS decrypts $rndInforGains\_c$ to get the randomized integer vector of information gains $rndInforGains\_v$ ❼ and broadcasts it to the EDOs ⑦. Each of the EDOs removes the randomization from $rndInforGains\_v$ to get $inforGains\_v$ vector which at the corresponding indexes contains the information gain data for each of the words and chooses the top m words with the highest information gain as the selected features' set SF, thus $SF = \{w_1, \dots w_m\}$, where $w_1, \dots w_m$ for performance reasons during the classification stage are sorted according to their hash values ❽.

4.3. Secure feature selection for binary datasets

***GRPV* (Generate Random Permutation Vector)**: Due to our strict security and privacy requirements (elaborated in Section I) of keeping private both the words and their counts (frequencies), simple hashing of the words will not help. Since the EDO records (emails, documents) are in a natural spoken language (say English), by hashing all of dictionary words of that language we can relatively easy apply the dictionary attack to find matches with the word hashes of the EDOs' records. Especially this is problematic for EDOs that wish to use a single record (mail) for training purposes (a scenario which we don't exclude), since the dictionary attack would reveal much of the record's (mail's) content. In order to guard against the dictionary attack, EDOs permute the hash bits of each word according to a random permutation upon which all EDOs agree. To obtain this random permutation, all of the EDOs engage in a secure multi-party computation (SMC) protocol given in Algorithm 4.1.

PROTOCOL 4.1: GRPV (Generate Random Permutation Vector)

INPUT: $\boldsymbol{hashBitSize}$
$\boldsymbol{hashBitSize}$**:** the size of the hash digest in bits

OUTPUT: $\boldsymbol{hashBitPermVec\_v}$
$\boldsymbol{hashBitPermVec\_v}$: the vector used to permute bit hashes at EDOs

```
PHASE I - EDOs:
1  for $k = 1$ to $n$ do
2      $rndVec\_v_k = generateRandomVector()$
3      $rndVec\_c_k = EncEncr(rndVec\_v_k)$
4      send $rndVec\_c_k$ to TEAS
Phase II – TEAS:
5  $rndVec\_v_T = generateRandomVector()$
6  $rndVec\_c_T = EncEncr(rndVec\_v_T)$
7  $rndVec\_c = \sum_{i=1}^{k} rndVec\_c_k + rndVec\_c_T$
8  send $rndVec\_c$ to E2DS
9  for $k = 1$ to $n$ do
10     send $\mathrm{Enc}_{pk_k}(rndVec\_v_T)$ to $\mathrm{EDO}_k$
Phase III-E2DS
11 $rndVec\_v = DecrDec(rndVec\_c)$
12 for $k = 1$ to $n$ do
13     send $\mathrm{Enc}_{pk_k}(rndVec\_v)$ to $\mathrm{EDO}_k$
Phase IV-EDOs:
14 for $k = 1$ to $n$ do
15     $rndVec2\_v = Dec_{sk_k}(rndVec\_v) - Dec_{sk_k}(rndVec\_v_T)$
16     $rndVec2\_v = rndVec2\_v \ \% \ hashBitSize$
17     $hashBitPermVec\_v = calcPermVec(rndVec2\_v)$
```

GRPV: each of the EDOs locally constructs a random integer vector of $hashBitSize$ elements, encodes and encrypts it and sends it to TEAS (lines 1-4). TEAS homomorphically adds them up, adds its own random terms, send its random terms to each EDO by encrypting them with EDOs' corresponding public keys and sends the sums of the random terms to be decrypted at E2DS (lines 5-10). E2DS decrypts the randomized sums and sends them to each EDO using their corresponding public keys (lines 11-13). EDOs decrypt and subtract the data obtained by E2DS and TEAS and apply the modulo operator with modulus of $hashBitSize$ in component (index)-wise manner to each of them to get the random $hashBitSize$ numbers placed in the $rndVec2\_v$, which are solely generated by EDOs (lines 15-16). $rndVec2\_v$ is than used by the $calcPermVec(\cdot)$ function to generate the $hashBitPermVec\_v$ by which the EDOs do the permutations of the hash bits of their words in the proceeding protocols. E.g. if in slot (index) 0 of $rndVec2\_v$ we have the value of 179, that means that the 0^{th} bit of the original hash will be the 179^{th} in the permuted hash bits,. If in slot, say 7, of $rndVec2\_v$ we have again the value of 179, which obviously is already used in the permuted hash bits, than we try the next slot to the right up until we find an empty place.

***secFS* - Secure Feature selection (Algorithm 4.2 and 4.3 - stage I and II)** is done in two stages. At the first stage the union of all of the words among n datasets is found and those that globally appears in less than val documents from all the datasets are filtered out, while at the second stage the top m words with the highest IG are selected. All of this is done under strict security and privacy requirements by 1) adding the corresponding block into the blockchain after each interaction (Fig.3) and 2) participants securely communicate with each other either by encrypting their data with E2DS public key with SIMD SWHE properties or by using the recipients public key generated for secure communication purposes.

secFS – S1: In Phase I all of the EDOs do some pre-processing on their local datasets, locally find all of the unique word and permute their binary hashes according $hashBitPermVec\_v$, and send them to TEAS (lines1-4). In Phase II TEAS finds the global union of the permuted hashes of the words, $W = \{\pi H(w_1), \pi(w_2), \dots, \pi(w_{|W|})\}$, and sends them to all EDOs (lines 5-8). In Phase IIA each of the EDOs, using the permuted hashes of the words in W as vector entries (indexes), locally construct $wordsInDoc\_c_k = \{N^{(k)}(w_1), \dots, N^{(k)}(w_{|W|})\} = \left\{\left(N^{(k)}(w_i, c_h)\right)_{i=1}^{|W|}\right\}$, corresponding to EDO$_k$'s counts of words $w_1, \dots, w_{|W|}$ in the documents of the local dataset DS$_k$, where $1 \leq k \leq n$, and send this ciphertext to TEAS (lines 9-12). In Phase IIB TEAS encodes the replicated vector of N integers whose value is val (line 13), then for the needs of secure comparisons generates and encodes random vectors $R\_v = \{R_1, \dots, R_{|W|}\} = \{(R_i)_{i=1}^{N}\}$ and $h1\_v = \{h1_1, \dots h1_{|W|}\} = \{(h1_i)_{i=1}^{|W|}\}$ s.t. $R_i > 0, |h1_i| < R_i$ for $1 \leq i \leq |W|$ (see Chapter 2.4.1) and generates $h2\_v = \{h2_1, \dots h2_{|W|}\} = \{(h2_i)_{i=1}^{N}\}$ for randomizing the secure comparison (line13). Afterwards sums up all of the $wordsInDoc\_c_k$ to get $wordsInDoc\_c$, which is an encryption of the vector of global counts of words appearing in documents, and performs the secure comparison proceeded with randomizations to get $rndWInDoc\_c$ (line 14). TEAS sends $rndWInDoc\_c$ to E2DS for decryption, while sending the randomization vector $h2\_v$ to each EDO (lines 16-18). In Phase IIC E2DS decrypts $rndWInDoc\_c$, which looks like $rndWInDoc\_v = \left\{\left((N(w_i) - val) \times R_i + h_{1,i}) + h_{2,i}\right)_{i=1}^{|W|}\right\}$ (line 19), and sends it to each EDO (lines 19-22).

Finally, in Phase IID each EDO removes the randomization by doing the subtraction $rndWInDoc\_v - h2\_v$ to get the secure comparison results $wordsInDoc\_v = \left\{\left((N(w_i) - val) \times R_i + h_{1,i}\right)_{i=1}^{|W|}\right\}$ (line 24). The words w_i for which the corresponding $\left((N(w_i) - val) \times R_i + h_{1,i}\right)_{i=1}^{|W|}$ term is negative (i.e. they appear in less than val global documents) are filtered out and are not part of the $WGThanV$ set which has $|W'|$ elements (words) (line 25).

secFS-S2: In Phase III, using each of the $|W'|$ words in $WGThanV$ as index entries, each EDO locally constructs vectors $hamWInDoc\_v_k = \left\{\left(N^{(k)}(w_i, c_h)\right)_{i=1}^{|W'|}\right\}$, $spamWInDoc\_v_k = \left\{\left(N^{(k)}(w_i, c_s)\right)_{i=1}^{|W'|}\right\}$ as well as the replicated $hamM\_v_k = \{N^{(k)}(c_h), \dots, N^{(k)}(c_h)\}$ and $spamM\_v_k = \{N^{(k)}(c_s)), \dots, N^{(k)}(c_s)\}$, containing the number of local ham and spam mails (documents), respectively, replicated for $|W'|$ times.

ALGORITHM 4.2: ***secFS-S1*** **(secure Feature Selection – Stage I)**

INPUT: $\boldsymbol{n}, \{\boldsymbol{DS_k}\}_{k=1}^{n}, \boldsymbol{KeySet} = \{(\boldsymbol{pk_k}, \boldsymbol{sk_k})_{k=1}^{n}, (\boldsymbol{pk_T}, \boldsymbol{sk_T}), (\boldsymbol{pk_E}, \boldsymbol{sk_E})\}, (\boldsymbol{pk}, \boldsymbol{sk}), \boldsymbol{val},$
$\boldsymbol{n}$: the number of EDOs
$\{\boldsymbol{DS_k}\}_{k=1}^{n}$**:** the local datasets of EDOs $1 \leq k \leq n$
$\boldsymbol{KeySet}$: set of all of the participants' public key pairs for blockchain and secure. communication
$(\boldsymbol{pk}, \boldsymbol{sk})$: key pairs of E2DS with SWHE properties

OUTPUT: $\boldsymbol{wordsGThanV\_v}$
$\boldsymbol{WGThanV}$: the set of words with at least val global document appearances

```
PHASE I - EDOs:
1 for $k = 1$ to $n$ do
2     $W_k = \pi(H(PreProcess(DS_k)))$ //permut. of hashes of unique words
3     $B\_1_{DO_k} = (MT(W_k), TS)$; $BCH_{DO_k}.AddBlock(Enc_{sk_k}(B\_1_{DO_k}))$
4     send $(Enc_{pk_T}(W_k), H(B\_1_{DO_k}))$ to TEAS
PHASE II – TEAS
5 $W = \bigcup_{k=1}^{n}[Dec_{sk_T}(W_k)]$ //sorted
6 $B\_1_T = (MT(\bigcup_{k=1}^{n} H(B\_1_{DO_k})), MT(W), TS)$; $BCH_T.AddBlock(Enc_{sk_T}(B\_1_T))$
7 for $k = 1$ to $n$ do
8     send $(\mathrm{Enc}_{pk_k}(W), H(B\_1_T))$ to EDO$_k$
PHASE IIA – EDOs
9 for $k = 1$ to $n$ do
10    $wordsInDoc\_c_k = getEncryptedCounts(DS_k, Dec_{sk_k}(W))$
11    $B\_2_{DO_k} = (H(B\_1_{DO_k}), H(B\_1_T), MT(wordsInDoc\_c_k), TS)$; $BCH_{DO_k}.AddBlock(Enc_{sk_k}(B\_2_{DO_k}))$
12    send $(wordsInDoc\_c_k, H(B\_2_{DO_k}))$ to TEAS
PHASE IIB – TEAS
13 $val\_p = Enc(\{val, \ldots, val\})$;$(R\_v, h1\_v, h2\_v) = rndVecsforComp()$;$(R\_p, h\_p) =$
$Encode(R\_v, (h1\_v - h2\_v))$
14 $rndWInDoc\_c = ((\sum_{k=1}^{n} wordsInDoc\_c_k - val\_p) \times R\_p) + h\_p$
15 $B\_2_T = (H(B\_1_T), MT(\bigcup_{k=1}^{n} H(B\_2_{DO_k})), MT(rndWInDoc\_c, h2\_v), TS)$;
$BCH_T.AddBlock(Enc_{sk_T}(B\_2_T))$
16 send $(rndWInDoc\_c, H(B\_2_T))$ to EDS
17 for $k = 1$ to $n$ do
18    send $(Enc_{pk_k}(h2\_v), H(B\_2_T))$ to EDO$_k$
PHASE IIC – E2DS
19 $rndWInDoc\_v = DecrDec(rndWInDoc\_c)$
20 $B\_1_E = (H(B\_2_T), MT(rndWInDoc\_v), TS)$; $BCH_E.AddBlock(Enc_{sk_E}(B\_1_E))$
21 for $k = 1$ to $n$ do
22    $send$ $(Enc_{pk_k}(rndWInDoc\_v), H(B\_1_E))$ to EDO$_k$
PHASE IID – EDOs
23 for $k = 1$ to $n$ do
24    $wordsInDoc\_v = (Dec_{sk_k}(rndWInDoc\_v) - Dec_{sk_k}(h2\_v))$
25    $WGThanV = greaterThanVal(W, wordsInDoc\_v)$
26    $B\_3_{DO_k} =$
      $(H(B\_2_{DO_k}), H(B\_2_T), H(B\_1_E), MT(WGThanV), TS)$; $BCH_{DO_k}.AddBlock(Enc_{sk_k}(B\_3_{DO_k}))$
```

After encoding and encrypting all of them they're send to TEAS (lines 1-4). In Phase IV TEAS homomorphically aggregates (sums up) those vectors to get the global counts for each word to appear in ham and spam documents, as well as the global number of ham and spam mails (lines 7-10). Then it finds $nrMails\_c, wordsInDoc\_c,$ $wordsNOTInDoc\_c, hamWNOTinDoc\_c$ and $spamWNOTinDoc\_c$ (lines 12-15, Fig.4.3), which in turn are used to find all of the randomized nominator and denominator terms inside the logarithms in (4) for each word (lines 16-23), denoted as $allRndNomDenom\_c$, and all of them are send to E2DS (line 25). E.g. in Fig4.4a) we illustrate the SIMD evaluations and randomizations of the nominator and denominator of the term $\frac{N(w_i,c_j)NT}{N(w_i)N(c_j)}$ in (4) done in lines 16 and 20. The nominators and denominators of the other terms in (4) are found in similar way. In Phase V E2DS decrypts all of the $allNomDenomRnd\_c$ sent by TEAS (lines 27) and finds the encryption of all of the logarithmic terms of (4) in randomized form, denoted as $allTheRndLogs\_c$. E.g. $logHamWInDoc\_c = EncEncr(K1 \times log$ $\left(\frac{rndHamWInD_{mul_{nrM_v}}}{rdWInDoc_{mul_{hamM_v}}}\right) = EncEncr\left(K1 \cdot log\left(\frac{N(w_i,c_h)\cdot NT\cdot R_{1,i}}{N(w_i)\cdot N(c_h)\cdot R_{5,1}}\right)_{i=1}^{|W'|}\right)$ (upper vector in Fig4.4b), where $K1$ is a constant used for integerization purposes and $R_{1,i}, R_{5,i}$ are random numbers (lines 28-31). Afterwards E2DS sends the $allTheRndLogs\_c$ to TEAS (line 34). In Phase VI TEAS removes the randomizations and multiplies each of the $allTheRndLogs\_c$ with the corresponding term of (4) (lines 35-42). E.g. $tmp\_c_1 = hamWInDoc\_c \times (logHamWInDoc\_c + invLogHamWInD\_p)$ in line 39 is illustrated in Fig.4.4b, corresponds to term $N(w_i, c_j)log\left(\frac{N(w_i,c_j)NT}{N(w_i)N(c_j)}\right)$ of (4). The other logarithmic terms multiplied with the corresponding counts of (4) are found in similar way. After homomorphically finding and randomizing the information gains to get $rndInforGains\_c$ (lines 43-46), it is send to E2DS for decryption, while a partial portion of the randomization is send to each EDO (lines 48-50). In Phase VII E2DS decrypts $rndInforGains\_c$ and sends it to each EDO. Finally, in Phase VIII each of the EDOs partially removes the randomization to get $inforGains\_v = \left\{(IG(w_i) \times R + h)_{i=1}^{|W'|}\right\}$ (line 58). Since all of the $IG(w_i)$ terms are multiplied and added to the same random R and h, respectively, it is easy for each of the EDOs to find the top m words with the highest IG. Afterwards EDOs apply the inverse permutation to word hashes using $hashBitPermVec\_v$ proceeded with sorting with respect to hash values, thus getting the selected features set, *SF* (line 59). Sorting is done for performance reasons when users prepare their queries according to the selected m words during *secC*.

Algorithm 4.3: ***secFS-S2*** **(<u>sec</u>ure <u>F</u>eature <u>S</u>election-Stage II)**

INPUT: $\boldsymbol{n}, \{\boldsymbol{DS_k}\}_{\boldsymbol{k=1}}^{\boldsymbol{n}}, \boldsymbol{KeySet}, (\boldsymbol{pk}, \boldsymbol{sk}), \boldsymbol{m}, \boldsymbol{WGThanV}$
$\boldsymbol{m}$: the number of features (words) to be selected at the end
$\boldsymbol{WGThanV}$: words with at least val global document appearances

OUTPUT: $\boldsymbol{SF} = \{\boldsymbol{H(w_1)}, \dots, \boldsymbol{H(w_m)}\}$**:**
$\boldsymbol{SF}$**:** m selected words with the highest IG sorted by their hashes

```
PHASE III – EDOs
1 for $k = 1$ to $n$ do
2     $(hamWInDoc\_c_k, spamWInDoc\_c_k, hamM\_c_k, spamM\_c_k) =$
          $= getEncryptedCounts(DS_k, wordsGThanV\_v)$
3     $B\_4_{DO_k} = (H(B\_3_{DO_k}), MT(hamWInDoc\_c_k,$
          $spamWInDoc\_c_k, hamM\_c_k, spamM\_c_k), TS)$
4     send $(hamWInDoc\_c_k, spamWInDoc\_c_k,$
          $hamM\_c_k, spamM\_c_k, H(B\_4_{DO_k}))$ to TEAS
5 $BCH_{DO_k}.AddBlock(Enc_{sk_k}(B\_4_{DO_k}))$
PHASE IV-TEAS
6 $(R1\_p, \dots, R8\_p) = RndEncodedVectors()$
7 $hamWInDoc\_c = \sum_{k=1}^{n}(hamWInDoc\_c_k),$
8 $hamM\_c = \sum_{k=1}^{n}(hamM\_c_k)$
9 $spamWInDoc\_c = \sum_{k=1}^{n}(spamWInDoc\_c_k),$
10 $spamM\_c = \sum_{k=1}^{n}(spamM\_c_k)$
11 $nrMails\_c = hamM\_c + spamM\_c,$
12 $wordsInDoc\_c = hamWInDoc\_c + spamWInDoc\_c$
13 $wordsNOTInDoc\_c = nrMails\_c - wordsInDoc\_c$
14 $hamWNOTinDoc\_c = hamM\_c - hamWInDoc\_c$
15 $spamWNOTinDoc\_c = spamM\_c - spamWInDoc\_c$
16 $rndHamWInD\_mul\_nrM\_c = (hamWInDoc\_c \times nrMails\_c) \times R1\_p$
17 $rndHamWNOTInD\_mul\_nrM\_c = (hamWNOTInDoc\_c \times nrMails\_c) \times R2\_p$
18 $rndSpamWInD\_mul\_nrM\_c = (spamWInDoc\_c \times nrMails\_c) \times R3\_p$
19 $rndSpamWNOTInD\_mul\_nrM\_c = (spamWNOTInDoc\_c \times nrMails\_c) \times R4\_p$
20 $rndWInDoc\_mul\_hamM\_c = (wordsInDoc\_c \times hamM\_c) \times R5\_p$
21 $rndWNOTInDoc\_mul\_hamM\_c = (wordsNOTInDoc\_c \times hamM\_c) \times R6\_p$
22 $rndWInDoc\_mul\_spamM\_c = (wordsInDoc\_c \times spamM\_c) \times R7\_p$
23 $rndWNOTInDoc\_mul\_spamM\_c = (wordsNOTInDoc\_c \times spamM\_c) \times R8\_p$
24 $B\_3_T = (H(B\_2_T), MT(\bigcup_{k=1}^{n} H(B\_4_{DO_k})), MT(allNomDenomRnd\_c), TS)$
25 send $(allRndNomDenom\_c, H(B\_3_T))$ to E2DS
26 $BCH_T.AddBlock(Enc_{sk_T}(B\_3_T))$
PHASE V – E2DS
27 $(rndHamWInD\_mul\_nrM\_v, rndHamWNOTInD\_mul\_nrM\_v,$
   $rndSpamWNOTInD\_mul\_nrM\_v,$
   $rndSpamWNOTInD\_mul\_nrM\_v,$
   $rndWInDoc\_mul\_hamM\_v, rndWNOTInDoc\_mul\_hamM\_v$
   $rndWInDoc\_mul\_spamM\_v, rndWNOTInDoc\_mul\_spamM\_v)$
                                   $= DecrDec(allRndNomDenom\_c\ sent\ by$ TEAS)
28 $logHamWInDoc\_c = EncEncr(K1 \times log(rndHamWInD\_mul\_nrM\_v/rndWInDoc\_mul\_hamM\_v))$
29 $logHamWNOTInDoc\_c = EncEncr(K1 \times log(rndHamWNOTInD\_mul\_nrM\_v/$
$rndWNOTInDoc\_mul\_hamM\_v))$
30 $logSpamWInDoc\_c = EncEncr(K1 \times log(rndSpamWInD\_mul\_nrM\_v/$
$rndWInDoc\_mul\_spamM\_v))$
31 $logSpamWNOTInDoc\_c = EncEncr(K1 \times log(rndSpamWNOTInD\_mul\_nrM\_v/$
$rndWNOTInDoc\_mul\_spamM\_v))$
32 $B\_2_E = (H(B\_1_E), H(B\_3_T), MT(allTheRndLogs\_c), TS)$
33 $BCH_E.AddBlock(Enc_{sk_E}(B\_2_E))$
```

```
34 send $(allTheRndLogs\_c, H(B\_2_E))$ to TEAS
PHASE VI –TEAS
35 $invLogHamWInD\_p = Encode(K1 \times log(R5\_v/R1\_v))$
36 $invLogHamWNOTInD\_p = Encode(K1 \times log(R6\_v/R2\_v))$
37 $invLogSpamWInD\_p = Encode(K1 \times log(R7\_v/R3\_v))$
38 $invLogSpamWNOTInD\_p = Encode(K1 \times log(R8\_v/R4\_v))$
39 $tmp\_c_1 = hamWInDoc\_c \times (logHamWInDoc\_c + invLogHamWInD\_p)$
40 $tmp\_c_2 = hamWNOTInDoc\_c \times (logHamWNOTInDoc\_c + invLogHamWNOTInD\_p)$
41 $tmp\_c_3 = spamWInDoc\_c \times (logSpamWInDoc\_c + invLogSpamWInD\_p)$
42 $tmp\_c_4 = spamWNOTInDoc\_c \times (logSpamWNOTInDoc\_c + invLogSpamWNOTInD\_p)$
43 $inforGains\_c = \sum_{i=1}^{4} tmp\_c_i$
44 $R\_v = \{R, \ldots R\}; h1\_v = \{h_{1,1}, \ldots, h_{N,1}\}; h2\_v = \{h, \ldots, h\}$
45 $(R\_p, h1\_p) = Encode(R\_v, h1\_v)$
46 $rndInforGains\_c = (inforGains\_c \times R\_p) + h1\_p$
47 $B\_4_T = (H(B\_3_T), H(B\_2_E), MT(rndInforGains\_c, (h1\_v - h\_v)), TS)$
48 send $(rndInforGains\_c, H(B\_4_T))$ to E2DS
49 for $k = 1$ to $n$ do
50     send $(Enc_{pk_k}(h1\_v - h\_v), H(B\_4_T))$ to EDO$_k$
51 $BCH_T.AddBlock(Enc_{sk_T}(B\_4_T))$
PHASE VII-E2DS:
52 $rndInforGains\_v = DecrDec(rndInforGains\_c)$
53 $B\_3_E = (H(B\_2_E), H(B\_4_T), MT(rndInforGains\_v), TS)$
54 for $k = 1$ to $n$ do
55     send $(Enc_{pk_k}(rndInforGains\_v), H(B\_3_E))$ to EDO$_k$
56 $BCH_E.AddBlock(Enc_{sk_E}(B\_3_E))$
PHASE VIII-EDOs
57 $rndInforGains\_v = Dec_{sk_k}(rndInforGains\_v)$
58 $inforGains\_v = rndInforGains\_v - Dec_{sk_k}(h1\_v - h\_v)$
59 $SF = sort(\pi^{-1}(topM(inforGains\_v)), hashBitPermVec\_v)$
```

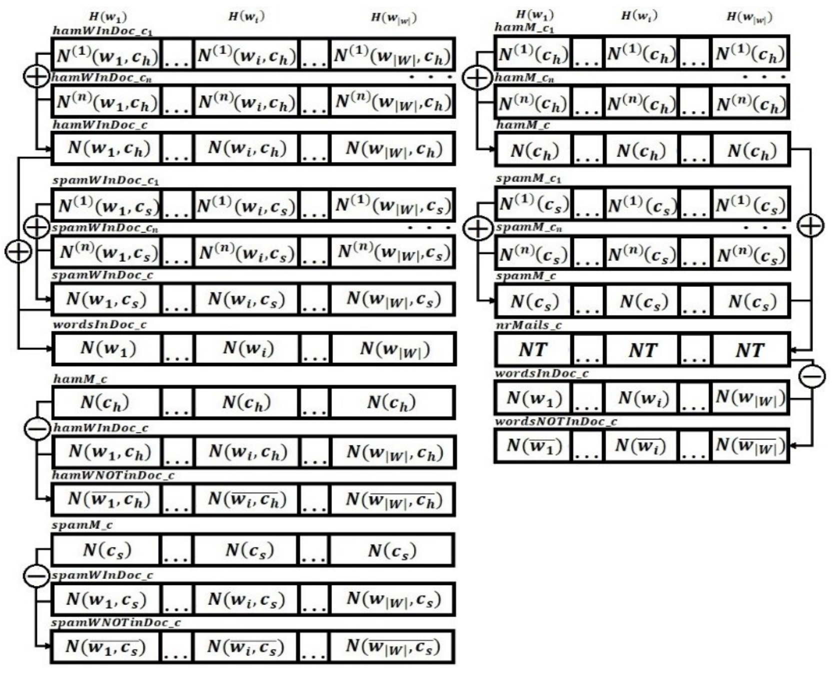


Fig.4.3. Illustration of SIMD evaluation of values in lines 13-16 of *secFS-S2*

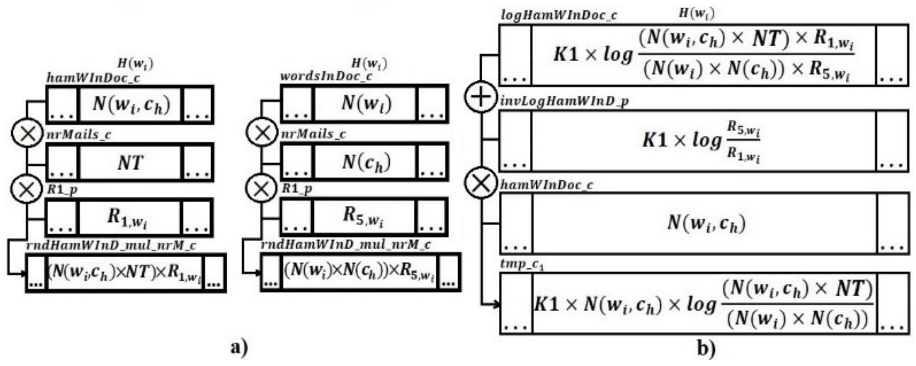


Fig.4.4. Illustration of SWHE SIMD evaluations of terms in (4) in *secFS-S2*

4.4. Secure feature selection for multi-label multi-output datasets

The background information is given in Chapter 2.2.2. The corresponding secure algorithm is given in Algorithm 4.4. Generally, Algorithm 4.4 follows Algorithm 4.3, but adjusted to the multi-label multi-output scenario.

ALGORITHM 4.4: ***secFS-MLMO-S2*** **(secure Feature Selection for Multi-Label Multi-Output datasets – Stage II)**

INPUT: $\boldsymbol{n}, \{\boldsymbol{DS_k}\}_{k=1}^{n}, \boldsymbol{KeySet} = \{(\boldsymbol{pk_k}, \boldsymbol{sk_k})_{k=1}^{n}, (\boldsymbol{pk_T}, \boldsymbol{sk_T}), (\boldsymbol{pk_E}, \boldsymbol{sk_E})\}, (\boldsymbol{pk}, \boldsymbol{sk}), \boldsymbol{m^l}, \boldsymbol{WGThanV}, \boldsymbol{L}, \boldsymbol{C^l}$

$\boldsymbol{L} = \{\boldsymbol{L_1}, \dots, \boldsymbol{L_{|L|}}\}$**:** the labels' set

$\boldsymbol{C^l} = \{\boldsymbol{C_1^l}, \dots, \boldsymbol{C_{|C_l|}}^{\boldsymbol{l}}\}$: The label's l set of classes, $1 \le l \le |L|$

$\boldsymbol{SF^l}$: the number of features (words) to be selected at the end for label l, $1 \le l \le |L|$

$\boldsymbol{WGThanV}$: the set of words with at least val global document appearances

OUTPUT: $\boldsymbol{SF^{MLMO}} = \left\{\{\boldsymbol{SF^l}\}_{\boldsymbol{l=1}}^{|L|}\right\} = \left\{\left\{\{\boldsymbol{H(w_i)}\}_{\boldsymbol{i=1}}^{\boldsymbol{m^l}}\right\}_{\boldsymbol{l=1}}^{|\boldsymbol{L}|}\right\}$

$\boldsymbol{SF^{MLMO}}$**:** the set of $|L|$ set hashes, each of m^l selected features with the highest IG for each label, $1 \le l \le |L|$

```
PHASE III – EDOs
1 for $k = 1$ to $n$ do
2     for $l = 1\ to\ |L|$ and $c = 1\ to\ |C^l|$
3         $classCount\_c_k[l][c], WInClassCount\_c_k[l][c] = getEncyrptedCountVectors(DS_k, WGThanV)$
4         $B\_4_{DO_k} = (H(B\_3_{DO_k}), MT(classCount\_c_k[l][c], WInClassCount\_c_k[l][c], TS)$;
          $BCH_{DO_k}.AddBlock(Enc_{sk_k}(B\_4_{DO_k}))$
5         send $(classCount\_c_k[l][c], WInClassCount\_c_k[l][c], H(B\_4_{DO_k}))$ to TEAS
PHASE IV-TEAS
7 $NT\_c = \sum_{k=1}^{n}\left(\sum_{c=1}^{|C^1|} classCount\_c_k[1][c]\right)$
8 $WInDocCount\_c = \sum_{k=1}^{n}\left(\sum_{c=1}^{|C^1|} WInClassCount\_c[1][c]\right)$
9 $WNOTInDocCount\_c = NT\_c - WInDocCount\_c$
10 for $l = 1\ to\ |L|$
11     for $c = 1\ to\ |C^l|$
12         $classCount_{c[l][c]} = \sum_{k=1}^{n} classCount_{c_k[l][c]}$ ;
           $WInClassCount\_c[l][c] = \sum_{k=1}^{n} WInClassCount\_c_k[l][c]$
13         $WNOTInClassCount\_c[l][c] = classCount\_c[l][c]\ - WInClassCount\_c[l][c]$
14         $rndWInClassCount\_mul\_NT\_c[l][c] = WInClassCount\_c[l][c] \times NT\_c \times R1[l][c]$
15         $rndClassCount\_mul\_WInDoc\_c[l][c] = classCount\_c[l][c] \times WInDocCount\_c \times R2[l][c]$
16         $rndWNOTInClassCount\_mul\_NT\_c[l][c] = WNOTInClassCount\_c[l][c] \times NT\_c \times R3[l][c]$
17         $rndClassCount\_mul\_WNOTInDoc\_c[l][c] = classCount\_c[l][c] \times WNOTInDocCount\_c \times R4[l][c]$
18 $B\_3_T = (H(B\_2_T), MT\big(\bigcup_{k=1}^{n} H\big(B\_4_{DO_k}\big)\big), MT(allNomDenomRnd\_c), TS)$;
$BCH_T.AddBlock\big(Enc_{sk_T}(B\_3_T)\big)$
19 send $(allRndNomDenom\_c, H(B\_3_T))$ to E2DS
PHASE V – E2DS
21 for $l = 1\ to\ |L|$
22     for $c = 1\ to\ |C^l|$
23         $(rndWInClassCount\_mul\_NT\_v[l][c], rndClassCount\_mul\_WInDoc\_v[l][c],$
24         $rndWNOTInClassCount\_mul\_NT\_v[l][c], rndClassCount\_mul\_WNOTInDoc\_v[l][c])$
25         $= DecrDec(allRndNomDenom\_c\ sent\ by$ TEAS)
26         $logWInCl\_c[l][c] = EncEncr(K1 \times log(rndWInClassCount\_mul\_NT\_v[l][c]/$
                                  $rndClassCount\_mul\_WInDoc\_v[l][c]))$
27         $logWNOTInCl\_c[l][c] = EncEncr(K1 \times log(rndWNOTInClassCount\_mul\_NT\_v[l][c]/$
                                  $rndClassCount\_mul\_WNOTInDoc\_v[l][c]))$
28 $B\_2_E = (H(B\_1_E), H(B\_3_T), MT(allTheRndLogs\_c), TS)$; $BCH_E.AddBlock(Enc_{sk_E}(B\_2_E))$
30 send $(allTheRndLogs\_c, H(B\_2_E))$ to TEAS
PHASE VI –TEAS
31 for $l = 1\ to\ |L|$
32     for $c = 1\ to\ |C^l|$
33         $invLogWInC\_p[l][c]\ = Encode(K1 \times log(R2\_v[l][c]/R1\_v[l][c]))$
34         $invLogWNOTInC\_p[l][c]\ = Encode(K1 \times log(R4\_v[l][c]/R3\_v[l][c]))$
35         $tmp\_c_1[l][c] = WInClassCount\_c[l][c] \times (logWInCl\_c[l][c] + invLogWInC\_p[l][c])$
36         $tmp\_c_2[l][c] = WNOTInClassCount\_c[l][c] \times (logWNOTInCl\_c[l][c] +\ \ invLogWNOTInC\_p[l][c])$
37     $inforGains\_c[l]\ = \sum_{c=1}^{|C^l|}\Big((\sum_{i=1}^{2} tmp\_c_i[l][c]))\Big)$
38 $R\_v[l] = \{R^l, \dots R^l\}; h1\_v[l] = \{h_{1,1}{}^l, \dots, h_{N,1}{}^l\}; h2\_v[l] = \{h^l, \dots, h^l\}$;
    $(R\_p[l], h1\_p[l]) =\ Encode(R\_v[l], h1\_v[l])$
39     $rndInforGains\_c[l]\ = (inforGains\_c[l]\ \times R\_p[l]) + h1\_p[l]$
40     send $(rndInforGains\_c[l]\ , H(B\_4_T))$ to E2DS
41     for $k\ = 1$ to $n$ do
42         send $(Enc_{pk_k}(h1\_v[l] - h\_v[l]), H(B\_4_T))$ to EDO$_k$
43 $B\_4_T = (H(B\_3_T), H(B\_2_E), MT(rndInforGains\_c, (h1\_v - h\_v)), TS)$;
$BCH_T.AddBlock\big(Enc_{sk_T}(B\_4_T)\big)$
PHASE VII-E2DS:
45 for $l = 1\ to\ |L|$
46     $rndInforGains\_v\ = DecrDec(rndInforGains\_c)$
47     for $k\ = 1$ to $n$ do
48         send $(Enc_{pk_k}(rndInforGains\_v), H(B\_3_E))$ to EDO$_k$
49 $B\_3_E = (H(B\_2_E), H(B\_4_T), MT(rndInforGains\_v), TS)$; $BCH_E.AddBlock(Enc_{sk_E}(B\_3_E))$
PHASE VIII-EDOs
51 for $l = 1\ to\ |L|$
51     $rndInforGains\_v = Dec_{sk_k}(rndInforGains\_v)$
52     $inforGains\_v = rndInforGains\_v -\ Dec_{sk_k}(h1\_v - h\_v)$
53     $SF^l = \{H(w_1), \dots, H(w_{m^l})\} = sort(\pi^{-1}(topM^l(inforGains\_v[l])))$ //inverse permut of hashes
54 $return\ SF^{MLMO}$ //words are sorted according to their hashes for performance reasons
```

Improvement 4.1. For computational and communicational efficiency purposes, whenever possible we apply polynomial switching technique proposed in [55], use multiple cores and threads (parallelize to the maximum extend) and simultaneously execute parts of the protocol in different participants whenever it's possible (e.g. adding block to blockchain after sending data to other participants). Those techniques alone give an improvement of several folds.

4.5. Experimental evaluations and comparisons

We run our codes in a machine with Intel Core i3-4000M processor with two physical cores, each of 2.4GHz (we utilize one core only in our implementations) with 4GB of DDR3 RAM and 64-bit Wındows 10 Pro as an OS. For SWHE purposes we use C++ based Microsoft's SEAL 3.2 library [40] which implements a version of [59].

For benchmark and comparison purposes with the state-of-the-art schemes, we used the email textual dataset Enron [37]. Its implementations can be found in [81]. for a total of around 9 000 lines of original C++ code. Enron e-mail dataset is a collection of e-mails from 150 user profiles, with a total of 16 555 ham and 17 148 spam e-mails. For the dataset pre-processing in line 2 of the $secFS - S1$ (Algorithm 4.2) we applied stop-words removal, partial punctuation removal, stemming by using the library in [45] and converting all of the letters to lowercase. For our protocols, the input parameter values for the Enron dataset are $val = 5\ and\ K1 = 100\ 000$.

For the SWHE parameters (Chapter 2.4.1) for $N = 8192{,}16384$ we use the corresponding values of $t = 37{,}60$ bits and $q = 218{,}438$ bits, respectively and security of $\lambda = 128$ bits.

Table 4.1. gives the cumulative (total) computation and communication costs for Enron email dataset for each participant while running Algorithm 4.2 and 4.3 (both stage I and II) for different polynomial modulus *N* and number of dataset owners *n.*

In Table 4.2 we compare the best results of our secure feature selection protocol (Algorithms 4.2 and 4.3, both *secFS-S1* and *secFS-S2*) with secure feature protocol in [28], which is the closest to ours from the related literature. In [28] they use a light version of the information gain – the Gini index, for secure feature selection. In [28] they use the non-textual Speed Dating binary dataset of [47] which has $NT = 8378$ records, $|F|$ =122 initial features, from which they select $m = 67$ features and finish the task in 60.57 min. Since [83] is non-textual dataset and the features are known publicly, they don't engage in a protocol similar to ours *GRPV* and *secFS-S1,* which makes their task easier.

Table 4.1. Computation and communication costs among different participants for different polynomial modulus N and number of the dataset owners n

N	n	Computation cost (s)			Communication cost (MB)		
		EDO	**TEAS**	**E2DS**	**EDO**	**TEAS**	**E2DS**
8192	**10**	197.7	394.4	2.0	190.2	45.7	16.5
	20	163.3	442.8	2.1	378.5	59.3	22.5
	30	148.5	484.4	2.1	566.4	72.9	28.6
	40	53.3	176.6	2.5	754.3	86.6	34.7
	50	34.1	173.6	1.9	482.1	100.2	40.7
16384	**10**	77.2	141.6	1.5	379.3	77.5	26.8
	20	69.0	195.5	1.1	756.6	91.1	32.8
	30	54.7	240.7	2.3	1133	104.8	38
	40	39.2	323.0	2.1	1510	118.4	45.0
	50	33.3	386.5	2.1	1887	132.0	51.0

Table 4.2. Comparison of different secure feature selection schemes

Scheme	NT	$\|F\|$	m=$\|SF\|$	**Comp.**	**Comm.**
Speed Dating [47]					
[29]*	8,378	122	67	60.57 min	Not reported
Enron email dataset [39]					
Ours	33,703	157,458	2047	10.15 min	310 MB

*4 co-located F32s V2 Azure, each with 32 cores and 64 GB RAM

4.6. Security analysis and proofs

We proof the security of our protocols under the semi-honest model using the definition 2.2 given in Chapter 2.4.2.

Theorem 4.1 *GRPV* (Algorithm 4.1) is secure under the semi-honest model

Proof: For $1 \leq k \leq n$, $V_{EDO_k}^{GRPV}(\lambda, \bar{x}) = \{Enc_{pk_k}(rndVec\_v), Enc_{pk_k}(rndVec\_v_T)\}$ is the view and $O_{EDO_k}^{GRPV}(\lambda, \bar{x}) = \{hashBitPermVec\_v\}$ is the output of EDO$_k$.

Let $\widetilde{rndVec\_v}$ be a random vector and let $\widetilde{rndVec\_c_T} = hashBitPermVec\_v - \widetilde{rndVec\_v}$. For the simulator of each eDo we have $S_{EDO_k}^{GRPV}\left(\lambda, O_{EDO_k}^{GRPV}(\lambda, \bar{x})\right) = \left\{Enc_{pk_k}\left(\widetilde{rndVec\_v}\right), Enc_{pk_k}\left(\widetilde{rndVec\_v_T}\right)\right\} \cong_c V_{EDO_k}^{GRPV}(\lambda, \bar{x})$.

For the view of TEAS we have $V_{TEAS}^{GRPV}(\lambda, \bar{x}) = \{(rndVec\_c_k)_{k=1}^{n}\}$ and it has no output. For the simulator we construct random SWHE ciphertexts, thus $S_{TEAS}^{GRPV}(\lambda) = \left(\widetilde{rndVec\_c_k}\right)_{k=1}^{n} \cong_c V_{TEAS}^{GRPV}(\lambda, \bar{x})$ due to the semantic security of the RLWE SWHE schemes.

For E2DS the view is $V_{E2DS}^{GRPV}(\lambda, \bar{x}) = \{rndVec\_c\}$ and it has no output. For its simulator we have $S_{E2DS}^{GRPV}(\lambda) = \{\widetilde{rndVec\_c}\} \cong_c V_{E2DS}^{GRPV}(\lambda, \bar{x})$ since RLWE based SWHE ciphertexts $rndVec\_c$ and $\widetilde{rndVec\_c}$ are indistinguishable to each other due to their semantic security.

None of the participants has a private input in *GRPV*. ■

Theorem 4.2: *secFS-S1* (Algorithm 4.2) is secure under the semi-honest model.

Proof: For $1 \leq k \leq n$, $V_{EDO_k}^{secFS-S1}(\lambda, \bar{x}) = \left\{Enc_{pk_k}(W), Enc_{pk_k}(h2\_v), Enc_{pk_k}(rndWInDoc\_v)\right\}$ is the view (where $\bar{x}$ the set of private input of all participants), $x_{EDO_k}^{secFS-S1} = \{PreProcess(DS_k)\}$ is the private input and $O_{EDO_k}^{secFS-S1}(\lambda, \bar{x}) = \{WGThanV\}$ is the output of EDO$_k$. From $x_{EDO_k}^{secFS-S1}$ and $O_{EDO_k}^{secFS-S1}(\lambda, \bar{x})$ we construct the set of the union of words $\widetilde{W} = \bigcup(W_k, WGThanV, rndW)$, where W_k is obtained as in lines 2-4 of *secFS-S1* and $rndW$ are permuted hashes of random words. $\widetilde{h2\_v}$ and $\widetilde{rndWInDoc\_v}$ are randomly chosen such that in the resulting $\widetilde{rndWInDoc\_v} - \widetilde{h2\_v}$ the indexes that correspond to the words of $WGThanV$ in $\widetilde{W}$ are positive, thus greater than val. For the simulator of EDO$_k$ we have $S_{EDO_k}^{secFS-S1}\left(\lambda, x_{EDO_k}^{secFS-S1}, O_{EDO_k}^{secFS-S1}(\lambda, \bar{x})\right) = \left\{Enc_{pk_k}(\widetilde{W}), Enc_{pk_k}\left(\widetilde{h2\_v}\right), Enc_{pk_k}\left(\widetilde{rndWInDoc\_v}\right)\right\}$, thus $S_{EDO_k}^{secFS-S1}\left(\lambda, x_{EDO_k}^{secFS-S1}, O_{EDO_k}^{secFS-S1}(\lambda, \bar{x})\right) \cong_c V_{EDO_k}^{secFS-S1}(\lambda, \bar{x})$. For TEAS $V_{TEAS}^{secFS-S1}(\lambda, \bar{x}) = \left\{\left(Enc_{pk_T}(W_k)\right)_{k=1}^{n}, (wordsInDoc\_c_k)_{k=1}^{n}\right\}$ is the view, $x_{TEAS}^{secFS-S1} = \emptyset$ is the input and $O_{TEAS}^{secFS-S1}(\lambda, \bar{x}) = \emptyset$ is the output it receives at the end of the protocol, where $\emptyset$ is the empty set. Let us construct n sets of permuted word hashes $(\widetilde{W_k})_{k=1}^{n}$ so that TEAS can't tell apart from them and$(W_k)_{k=1}^{n}$, and n random ciphertexts $(\widetilde{wordsInDoc\_c_k})_{k=1}^{n}$ which can't be distinguished from $(wordsInDoc\_c_k)_{k=1}^{n}$ due to the semantic security of the RLWE schemes. For the TEAS' simulator then we have $S_{TEAS}^{secFS-S1}\left(\lambda, x_{TEAS}^{secFS-S1}, O_{TEAS}^{secFS-S1}(\lambda, \bar{x})\right) = \left\{\left(Enc_{pk_T}(\widetilde{W_k})\right)_{k=1}^{n}, \left(\widetilde{wordsInDoc\_c_k}\right)_{k=1}^{n}\right\} \cong_c V_{TEAS}^{secFS-S1}(\lambda, \bar{x})$.

E2DS' view is $V_{E2DS}^{secFS-S1}(\lambda, \bar{x}) = \{rndWInDoc\_c\}$ and its input and output is the empty set. We construct a random ciphertext $\widetilde{rndWInDoc\_c}$ which can't be distinguished from $rndWInDoc\_c$ to the semantic security of the RLWE based schemes. For the simulator then we have $S_{E2DS}^{secFS-S1}\left(\lambda, x_{E2DS}^{secFS-S1}, O_{E2DS}^{secFS-S1}(\lambda, \bar{x})\right) = \left\{\widetilde{rndWInDoc\_c}\right\} \cong_c V_{E2DS}^{secFS-S1}(\lambda, \bar{x})$. ∎

Corollary 4.1: *secFS-S1* is secure under the semi-honest model when out of n, $n-3$ EDOs fail or $n-2$ EDOs collude.

Proof: Without loss of generality, let the 3 left (not failed) EDOs be EDO_k, for $1 \leq k \leq 3$, and let us treat the n-2 colluding EDOs as a single entity EDO_3, thus we again have 3 participating EDOs denoted as EDO_k, for $1 \leq k \leq 3$. We go through the similar lines as Theorem 2 to prove Corollary 1. If there are only 2 non-failed or non-colluding EDOs left, then for the words in W (line 8 and 14) which don't appear in one EDO, not only this EDO will know that they exist in the other EDO, he will also know whether the counts of those words is greater or smaller than val based on the final $WGThanV$ words' set, which constitutes a partial leakage that goes against the strict security and privacy requirements set in Chapter 4.1. ∎

Theorem 4.3: *secFS-S2* (Algorithm 4.3) is secure under the semi-honest model.

Proof: For $1 \leq k \leq n$, $V_{EDO_k}^{secFS-S2}(\lambda, \bar{x}) = \left\{Enc_{pk_k}(rndInforGains\_v), Enc_{pk_k}(h1\_v - h\_v)\right\}$ is the view, $x_{EDO_k}^{secFS-S2} = \{PreProcess(DS_k), WGThanV\}$ is the private input and $O_{EDO_k}^{secFS-S2}(\lambda, \bar{x}) = \{SF\}$ is the output of EDO_k. We construct random $\widetilde{rndInforGains\_v}$ and $h1\_\widetilde{v - h\_v}$ s.t. after $\widetilde{rndInforGains\_v} - h1\_\widetilde{v - h\_v}$, the resulting vector, that in its corresponding indexes is supposed to store the relative IG of the words in $WGThanV$, with have its top m highest values for the indexes that that correspond to the words of the selected features' set SF. For the corresponding simulator of each of the EDOs we have

$S_{EDO_k}^{secFS-S2}\left(\lambda, x_{EDO_k}^{secFS-S2}, O_{EDO_k}^{secFS-S2}(\lambda,\bar{x})\right) =$
$\left\{Enc_{pk_k}\left(rndIn\widetilde{forGains}\_v\right), Enc_{pk_k}(h1\_\widetilde{v-h}\_v)\right\}$, thus $S_{EDO_k}^{secFS-S2}\left(\lambda, x_{EDO_k}^{secFS-S2}, O_{EDO_k}^{secFS-S1}(\lambda,\bar{x})\right) \cong_c V_{EDO_k}^{secFS-S2}(\lambda,\bar{x})$.

For the view of TEAS we have $V_{TEAS}^{secFS-S2}(\lambda,\bar{x}) = \{(hamWInDoc\_c_k)_{k=1}^{n}, (spamWInDoc\_c_k)_{k=1}^{n}, (hamM\_c_k)_{k=1}^{n},$
$(spamM\_c_k)_{k=1}^{n}, allTheRndLogs\_c\}$, $x_{TEAS}^{secFS-S2} = \emptyset$ is the input and $O_{TEAS}^{\mathrm{secFS-S2}}(\lambda,\bar{x}) = \emptyset$ is the output. For the simulator we construct the corresponding ciphertexts randomly, which can't be differentiated from their counterparts due to the semantic security of the RLWE schemes, thus $S_{TEAS}^{secFS-S2}\left(\lambda, x_{TEAS}^{secFS-S2}, O_{TEAS}^{secFS-S1}(\lambda,\bar{x})\right) =$
$\{\left(ham\widetilde{WInDoc}\_c_k\right)_{k=1}^{n}, \left(spam\widetilde{WInDoc}\_c_k\right)_{k=1}^{n}, \left(ha\widetilde{mM}\_c_k\right)_{k=1}^{n},$
$\left(spa\widetilde{mM}\_c_k\right)_{k=1}^{n}, allThe\widetilde{Rnd}Logs\_c\} \cong_c V_{TEAS}^{secFS-S2}(\lambda,\bar{x})$.

E2DS' *view is* $V_{E2DS}^{secFS-S2}(\lambda,\bar{x}) = \{allRndNomDenom\_c, rndInforGains\_c\}$ and its input and output is the empty set. For the simulator we construct random ciphertexts, thus $S_{E2DS}^{secFS-S2}\left(\lambda, x_{E2DS}^{secFS-S2}, O_{E2DS}^{secFS-S2}(\lambda,\bar{x})\right) =$
$\left\{allRnd\widetilde{NomDenom}\_c, rndIn\widetilde{forGains}\_c\right\} \cong_c V_{E2DS}^{secFS-S2}(\lambda,\bar{x})$

Corollary 4.2: *secFS-S2* is secure under the semi-honest model when out of n $n-3$ EDOs fail or $n-2$ EDOs collude. ■

Theorem 4.4: *secFS-MLMO-S2* (Algorithm 4.4) is secure under the semi-honest model
Proof: We follow similar reasoning as with Theorem 4.4 ■

Chapter 5

SECURE AND PRIVATE MACHINE LEARNING TRAINING

Definition 5.1: ML model training is the process of acquiring parameters of the trained model TM from a labeled dataset DS consisted of a selected feature set SF according to an algorithm $\mathcal{T}$, thus $TM = \mathcal{T}(\mathrm{DS}, SF)$.

In this chapter, we introduce the security, privacy and efficiency goals (requirements) for our secure training protocols. We proceed with constituents (participants) of our secure training schemes, their adversary models as well as we provide a brief flow of the protocols, which will be elaborated in more details in the later subchapters of this chapter. We conclude the chapter by experimentally evaluating and comparing our schemes with the related research as well as we proof their security under the semi-honest model. All the necessary background information and notations for this chapter was given in Chapter 2. We should note that our secure training protocol(s) can be seen as a natural follow up of the corresponding secure feature selection protocol(s). Namely, after securely and privately selecting the most suitable features, we proceed to train a ML model based on the selected features.

5.1. Introduction

The security and privacy requirements for our secure training protocol are the following:

- **Privacy of the input features**. The inputs here are the m selected features, i.e. the output of the *secFS* protocol
- **Privacy of the input features' values**
- **Security and privacy of intermediate results**
- **Privacy of the output, i.e. the trained model.** This is one of the rare protocols to keep private the final trained ML model at any stage.
- Have other properties related to secure training mentioned in Table 3.2

5.2. System architecture, adversary models and protocol-flows-at-a-glance

Similarly to the secure feature selection case (Chapter 4.2), our participants for the secure feature selection protocols are: **1) EDO** (The Edge Dataset Owner) - We have n such EDOs in our system, denoted as EDO_k, each owns a dataset DS_k, where $1 \leq k \leq n$, that they are willing to use for training ML models in a secure and private fashion. **2) TEAS** (The Edge Aggregating Server): a server used to do the bulk of the proposed protocols' homomorphic computation. **3) E2DS** (The Edge Encryption Decryption Server): It's the only participant in the system that has a pair of public keys with SWHE properties (Chapter 2.4). All the data that are homomorphically evaluated in our protocols are encrypted using E2DS' public key, thus it's the only one that can decrypt them. All of them are illustrated in Fig.5.1.

Adversary models: All the participants are assumed to be in the passive **semi-honest** (honest but curious) **model**, which means that they follow the protocol but on the background they try to infer some private data which they are not supposed to. A formal definition of the semi-honest model is given in Chapter 2.4.1. We assume that TEAS and E2DS don't collude. Out of n EDOs, our environment setting allows for up to $n - 3$ EDO failures and up to $n - 2$ collusions without jeopardizing the privacy of the remaining and non-colluding EDOs . The motivations for participant to behave in the described manners are given in [8-18].

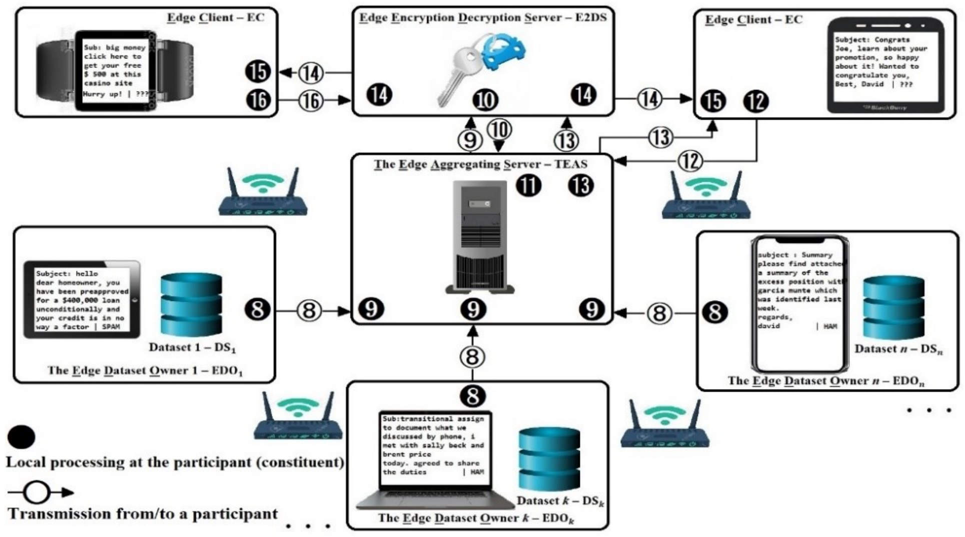

Edge Client – EC
Edge Encryption Decryption Server – E2DS
Edge Client – EC
The Edge Aggregating Server – TEAS
Dataset 1 – DS$_1$
The Edge Dataset Owner 1 – EDO$_1$
Dataset n – DS$_n$
The Edge Dataset Owner n – EDO$_n$
Dataset k – DS$_k$
The Edge Dataset Owner k – EDO$_k$
Local processing at the participant (constituent)
Transmission from/to a participant

Fig. 5.1. Protocol flows for secure training protocol (*secT*)

Protocol flows at-a-glance: In secure training we continue the blockchain (Fig.4.2) started in Chapter 4. All the participants have a pair of public/secret keys used for signing their corresponding blocks of the blockchain and for secure communication. Additionally, E2DS has a pair of public/secret key with SWHE properties (Chapter 2.4.1). All of these key pairs form the *KeySet* set. While we designed our protocols having in mind primarily binary textual datasets, they are also applicable to non-textual ones and can be easily generalized to multi-class scenarios. This choice was done for simplicity and benchmark (comparison) purposes with the related research.

***secT* (secure training):** Illustrated in Fig.5.1. Using the selected words of SF as index entries, n EDOs simultaneously construct the ciphertext of the training vector which containing the local frequencies of each word, TV_c_k, ❽ and send them to TEAS ⑧. TEAS sums them up to get the global frequencies, randomizes this result to get $rndTV\_c = (\sum_{k=1}^{n} TV\_c_k) \times R\_p$ ❾ and sends it to E2DS ⑨. After decrypting it, E2D2 finds the randomized class and conditional word-class logarithms of probabilities, integerizes and encrypts them to get the randomized trained model $rndTM\_c$ ❿ and sends it back to TEAS ⑩, which homomorphically removes the randomization to get the final trained model TM_c which represents the Naïve Bayes (NB) or the multinomial NB (MNB) classifier ⓫.

5.3. Secure training for non-textual datasets

Considering the notations in Chapter 2.1.1 for the non-textual datasets, the architecture, protocol flows and participants in Chapter 5.2, in this chapter we give a detailed pseudocode of the privacy preserving training protocol of Naïve Bayes (NB) models for non-textual datasets (Algorithm 5.1) which was briefly elaborated in Chapter 5.2. It is also accompanied with corresponding illustrations and comments in the pseudocode.

ALGORITHM 5.1: ***PPTMDO*** **(Privacy Preserving Training From Multiple Dataset Owners)**

```
INPUT: $\{\boldsymbol{DS_k}\}_{k=1}^{n}, \boldsymbol{F}, \boldsymbol{C}, \boldsymbol{n}$
$\{\boldsymbol{DS_k}\}_{k=1}^{n}$: The local datasets owned by the $k$-th EDO, for $1 \leq k \leq n$. Each EDO has one private dataset
$\boldsymbol{F}$: $F = \{F_1, F_2, \dots, F_f\}$, where $F_i = \{V_{1,Fi}, V_{2,Fi}, \dots, V_{|Fi|,Fi}\}$. $F_i$, st. $1 \leq i \leq f$  (explained in Chapter 2.1.1)
$\boldsymbol{C}$: The set of classes $C = \{C_1, C_2, \dots, C_c\}$  (explained in Chapter 2.1.1)
$\boldsymbol{n}$: number of dataset owners
OUTPUT: $\boldsymbol{TM\_c}$
$\boldsymbol{TM\_c}$: the encryption of the SIMD encoded final trained model which will be stored at TEAS
PHASE VIII - EDOs:
1  for $k = 1$ to $n$ do
2      $TV\_v_k = getTVReplicated()$//the case when we deal with simultaneous classification of $p$ queries
3      $TV\_c_k = EncEncr(TV\_v_k)$
4      send $TV\_c_k$ to TACS       //all of the $[TV(k)]$ training vectors should look as depicted in Fig.5.2-5.3
PHASE IX - TEAS:
5  $GTV\_c = \sum_{k=1}^{n} TV\_c_k$  //sum them up to get the Global Trained Vector-$GTV\_c$, Fig.5.4
6  $R\_v = \{R_{N(C_1)}, R_{N(V_{1,F1};C_1)}, \dots\}$; $R\_p = Encode(R\_v)$
7  $rndGTV\_c = GTV\_c \times R\_p$       //Fig.5.5
8  send $rndGTV\_c$ to E2DS
9  $randLogsOFInvProbs\_v = calcRndLogsOfProbs(R\_v)$                    //equation (5.1)
10 $randLogsOFInvProbs\_p = Encode(randomLogsOFInvProbs\_v)$//eq. (5.1), upper vector in Fig.5.6
PHASE X – E2DS:
11 $rndGTV\_v = Decrypt\_Decode(rand\_GTV\_c)$
12 $rndTM\_v = calculateKLogOfProbs(rndGTV\_v)$                    //equation (5.2)
13 $rndTM\_c = EncEncr(rndTM\_v)$ //eq. (5.2), middle vector in Fig.5.6
14 send $rndTMC\_c$ to TACS
PHASE XI - TEAS:
15 $TM\_c = rndTM\_c + rndLogsOFInvProbs\_p$ //equation (5.3), all illustrated in Fig.5.6
```

Phase VIII (lines 1-4) is done in parallel at all of the n EDOs. Each of the DOs locally constructs the training vector (TV_v_k) (Fig.5.2-5.3) so that EDO k (s.t. $1 \leq k \leq n$) for a certain class C_j (s.t. $1 \leq j \leq c$), at the beginning puts the local frequency (counts) $N^{(k)}(C_j)$ for that class, proceeded with local joint class-value counts $N^{(k)}(V_{m,Fi}\,;\,C_j)$ for all the f features ($1 \leq i \leq f$) and all feature-values m ($1 \leq m \leq |F_i|$), as it is shown in Fig.5.2. With this approach only ($\sum|F_i| + 1$) slots per class are needed. However, for efficiency purposes when using the *secSum* algorithm (Chapter 6.2), we make sure that each class has a portion (number) of slots which is a power of 2, concretely $n_s = 2^{\lceil log(\sum|F_i|+1)\rceil}$ slots, where the remainder of $2^{\lceil log(\sum|F_i|+1)\rceil} - (\sum|F_i| + 1)$ slots are filled up with dummy values (preferably zeros), as shown at the portion of the dummy value slots at Fig.5.2. The same is repeated for all of the c classes, thus each TV_v_k has $n_L = c \cdot n_s + 1$ slots, where the last slot is reserved for the number of transactions at EDO $k - NT(k)$ (Fig.5.3). If we want a simultaneous classifications of p queries, we replicate the TV_v_k for p times (line 2). Then the EDOs encode and encrypt their final (replicated) training vectors TV_c_k and send them to TEAS (line 4).

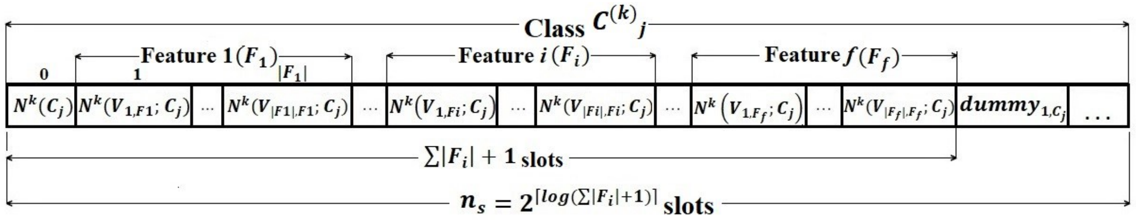


Fig. 5.2. Depiction of the portion of the encrypted counts of the training vector of Edge Dataset Owner k holding counts related to class C_j

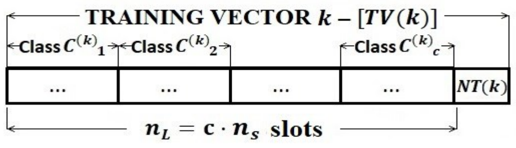


Fig. 5.3. The overall training vector TV_v_k (for all classes) at the Edge Dataset Owner k

In Phase IX TEAS receives all of the trained vectors from EDOs, aggregates (sums them up) to get the global training vector ciphertext GTV_c, which contains the global counts in a single ciphertext (line 5, Fig.5.4). Afterwards TEAS constructs a random looking plaintext (R_p) (line 6), multiplies GTV_c with it to get the randomized $rndGTV\_c$ (line 7, Fig.5.5), then sends this $rndGTV\_c$ to E2DS (lines 8). Meanwhile in lines 9-10 TEAS calculates and construct the plaintext of the inverse logs of probabilities of the random vector R_p ($randLogsOFInvProbs\_p$), shown at the upper vector at Fig.5.6 and (5.1). For efficiency purposes lines 9-10 at TEAS are done in parallel (overlap) with Phase X.

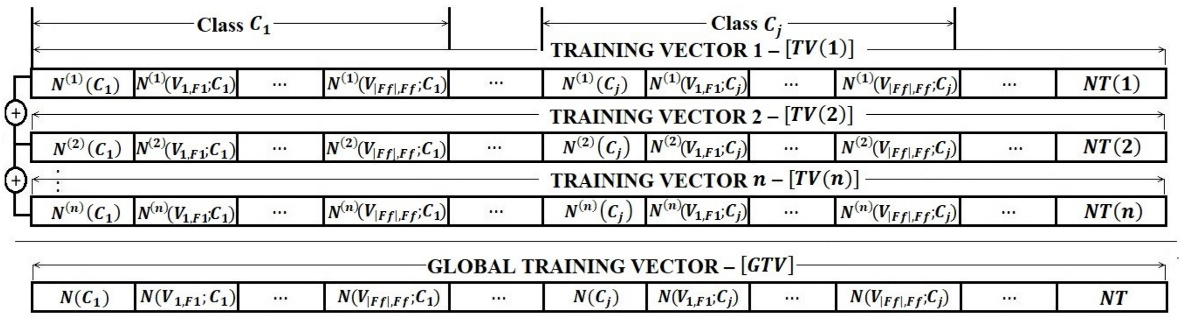

Class C_1
Class C_j
TRAINING VECTOR 1 – $[TV(1)]$
$N^{(1)}(C_1)$ $N^{(1)}(V_{1,F1};C_1)$ … $N^{(1)}(V_{|Ff|,Ff};C_1)$ … $N^{(1)}(C_j)$ $N^{(1)}(V_{1,F1};C_j)$ … $N^{(1)}(V_{|Ff|,Ff};C_j)$ … $NT(1)$
TRAINING VECTOR 2 – $[TV(2)]$
$N^{(2)}(C_1)$ $N^{(2)}(V_{1,F1};C_1)$ … $N^{(2)}(V_{|Ff|,Ff};C_1)$ … $N^{(2)}(C_j)$ $N^{(2)}(V_{1,F1};C_j)$ … $N^{(2)}(V_{|Ff|,Ff};C_j)$ … $NT(2)$
TRAINING VECTOR n – $[TV(n)]$
$N^{(n)}(C_1)$ $N^{(n)}(V_{1,F1};C_1)$ … $N^{(n)}(V_{|Ff|,Ff};C_1)$ … $N^{(n)}(C_j)$ $N^{(n)}(V_{1,F1};C_j)$ … $N^{(n)}(V_{|Ff|,Ff};C_j)$ … $NT(n)$
GLOBAL TRAINING VECTOR – $[GTV]$
$N(C_1)$ $N(V_{1,F1};C_1)$ … $N(V_{|Ff|,Ff};C_1)$ … $N(C_j)$ $N(V_{1,F1};C_j)$ … $N(V_{|Ff|,Ff};C_j)$ … NT

Fig.5.4. Aggregating (homomorphically summing up) the local training vectors – TV_c_k to get the global training vector GTV_c

$$\left[K\,log\frac{R_{NT}}{R_{N(C_j)}}\right]$$
$$\left[Klog\frac{R_{N(C_j)}}{R_{N(V_{m,Fi}\,;\,C_j)}}\right] \tag{5.1}$$

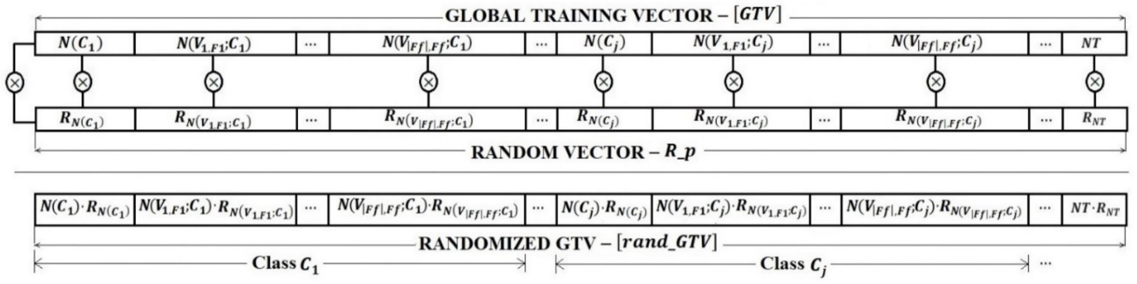

GLOBAL TRAINING VECTOR – $[GTV]$
$N(C_1)$ $N(V_{1,F1};C_1)$ … $N(V_{|Ff|,Ff};C_1)$ … $N(C_j)$ $N(V_{1,F1};C_j)$ … $N(V_{|Ff|,Ff};C_j)$ … NT
$R_{N(C_1)}$ $R_{N(V_{1,F1};C_1)}$ … $R_{N(V_{|Ff|,Ff};C_1)}$ … $R_{N(C_j)}$ $R_{N(V_{1,F1};C_j)}$ … $R_{N(V_{|Ff|,Ff};C_j)}$ … R_{NT}
RANDOM VECTOR – R_p
$N(C_1)\cdot R_{N(C_1)}$ $N(V_{1,F1};C_1)\cdot R_{N(V_{1,F1};C_1)}$ … $N(V_{|Ff|,Ff};C_1)\cdot R_{N(V_{|Ff|,Ff};C_1)}$ … $N(C_j)\cdot R_{N(C_j)}$ $N(V_{1,F1};C_j)\cdot R_{N(V_{1,F1};C_j)}$ … $N(V_{|Ff|,Ff};C_j)\cdot R_{N(V_{|Ff|,Ff};C_j)}$ … $NT\cdot R_{NT}$
RANDOMIZED GTV – $[rand\_GTV]$
Class C_1
Class C_j

Fig.5.5. Randomizing GTV_c to get $rndGTV\_c$

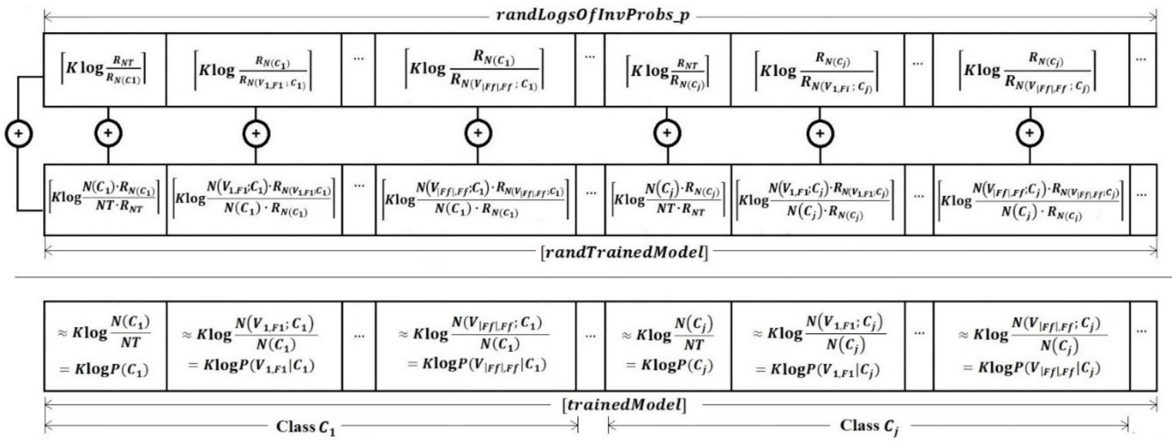

$randLogsOfInvProbs\_p$
$\left[K\log\frac{R_{NT}}{R_{N(C1)}}\right]$ $\left[Klog\frac{R_{N(C_1)}}{R_{N(V_{1,F1}:C_1)}}\right]$ … $\left[Klog\frac{R_{N(C_1)}}{R_{N(V_{|Ff|,Ff}:C_1)}}\right]$ … $\left[K\log\frac{R_{NT}}{R_{N(C_j)}}\right]$ $\left[Klog\frac{R_{N(C_j)}}{R_{N(V_{1,Fi}:C_j)}}\right]$ … $\left[Klog\frac{R_{N(C_j)}}{R_{N(V_{|Ff|,Ff}:C_j)}}\right]$ …
$\left[Klog\frac{N(C_1)\cdot R_{N(C_1)}}{NT\cdot R_{NT}}\right]$ $\left[Klog\frac{N(V_{1,F1};C_1)\cdot R_{N(V_{1,F1}C_1)}}{N(C_1)\cdot R_{N(C_1)}}\right]$ … $\left[Klog\frac{N(V_{|Ff|,Ff};C_1)\cdot R_{N(V_{|Ff|,Ff};C_1)}}{N(C_1)\cdot R_{N(C_1)}}\right]$ … $\left[Klog\frac{N(C_j)\cdot R_{N(C_j)}}{NT\cdot R_{NT}}\right]$ $\left[Klog\frac{N(V_{1,F1};C_j)\cdot R_{N(V_{1,F1};C_j)}}{N(C_j)\cdot R_{N(C_j)}}\right]$ … $\left[Klog\frac{N(V_{|Ff|,Ff};C_j)\cdot R_{N(V_{|Ff|,Ff};C_j)}}{N(C_j)\cdot R_{N(C_j)}}\right]$ …
$[randTrainedModel]$
$\approx Klog\frac{N(C_1)}{NT} = KlogP(C_1)$ $\approx Klog\frac{N(V_{1,F1};C_1)}{N(C_1)} = KlogP(V_{1,F1}|C_1)$ … $\approx Klog\frac{N(V_{|Ff|,Ff};C_1)}{N(C_1)} = KlogP(V_{|Ff|,Ff}|C_1)$ … $\approx Klog\frac{N(C_j)}{NT} = KlogP(C_j)$ $\approx Klog\frac{N(V_{1,F1};C_j)}{N(C_j)} = KlogP(V_{1,F1}|C_j)$ … $\approx Klog\frac{N(V_{|Ff|,Ff};C_j)}{N(C_j)} = KlogP(V_{|Ff|,Ff}|C_j)$ …
$[trainedModel]$
Class C_1
Class C_j

Fig.5.6. Adding $randLogsOfInvProbs\_p$ with $rndTM\_c$ to get the trained model TM_c.

In Phase X E2DS decrypts and decodes the randomized global training vector to calculate the randomized probabilities of the trained model according to (5.2), where $1 \leq i \leq f$; $1 \leq j \leq c$ and $1 \leq m \leq |F_i|$ (lines 11). As it's shown in Fig.5.6, after properly encoding and encrypting those probabilities into their corresponding places (slots) to get the $rndTM\_c$, EDS sends back to TEAS the $rndTM\_c$ (lines 12-14).

$$\begin{gathered} \left\lceil K \log \frac{N(C_j)\cdot R_{N(C_j)}}{NT\cdot R_{NT}} \right\rceil \\ \left\lceil K \log \frac{N(V_{m,Fi}\,;C_j)\cdot R_{N(V_{m,Fi}\,;C_j)}}{N(C_j)\cdot R_{N(C_j)}} \right\rceil \end{gathered} \tag{5.2}$$

Finally, in Phase IV TEAS gets and de-randomizes the $rndTM\_c$ by adding it with $randLogsOFInvProbs\_p$ to get the final trained model denoted as TM_c as its shown in (5.3) and Fig.5.6 (line 15). TEAS always holds the TM_c in encrypted form at his side to be later used for classification purposes.

$$\begin{gathered} \left\lceil K\,log \frac{R_{NT}}{R_{N(C_j)}} \right\rceil + \left\lceil K\,log \frac{N(C_j)\cdot R_{N(C_j)}}{NT\cdot R_{NT}} \right\rceil \\ \approx Klog\frac{N(C_j)}{NT} = KlogP(C_j) \\ \left\lceil Klog\frac{R_{N(C_j)}}{R_{N(V_{m,Fi}\,;C_j)}} \right\rceil + \left\lceil K\,log\frac{N(V_{m,Fi}\,;C_j)\cdot R_{N(V_{m,Fi}\,;C_j)}}{N(Cj)\cdot R_{N(Cj)}} \right\rceil \\ \approx Klog\frac{N(V_{m,Fi}\,;C_j)}{N(Cj)} = KlogP(V_{m,Fi}|C_j) \end{gathered} \tag{5.3}$$

Terms in (5.3) are the same as the terms in (2.5) for $1 \leq i \leq f$; $1 \leq j \leq c$ and $1 \leq m \leq |F_i|$. Actually they represent the Naïve Bayes trained model consisted of the global class probabilities and conditional value-class probabilities shown in Chapter 2.1.1 and at the trained model ciphertext TM_c of Fig.5.6.

5.4. Secure training for binary and multi-label multi-output textual datasets

In Algorithm 5.2. we give the pseudocode for the ML training of the binary textual datasets (*secT*), which is expected to follow up the algorithms for secure feature selection of binary textual datasets (Algorithms 4.2 and 4.3). Thus, we train our textual ML models over the selected m features from Algorithms 4.2 and 4.3. The necessary background related to Algorithm 5.2 is given in Chapter 2.1.2. The blockchain started in Algorithms 4.2 and 4.3 is continued in Algorithm 5.2 to provide end-to-end security (from raw data till the final trained model), for which we proof the security in Chapter 5.6.

ALGORITHM 5.2*: secT* (<u>sec</u>ure <u>T</u>raining)

INPUT: $\boldsymbol{n, \{DS_k\}_{k=1}^{n}, KeySet, SF, m}$

$\boldsymbol{SF = \{H(w_1), \dots, H(w_m)\}}$: the set of hashes of m selected features with the highest IG

OUTPUT: $\boldsymbol{TM\_c = \{K(P(c_h) - P(c_s)), K(P(c_h|w_1) - P(c_s|w_1)), \dots, K(P(c_h|w_m) - P(c_s|w_m))\}}$

$\boldsymbol{TM\_c}$: the binary case trained model ciphertext

```
PHASE VIII - EDOs:
1 for $k = 1$ to $n$ do
2     $TV\_c_k = getEncryptTVAccordingToSF(DS_k, SF)$
3     $B\_5_{DO_k} = (H(B\_4_{DO_k}), H(B\_4_T), H(B\_3_E), MT(TV\_c_k), TS)$; $BCH_{DO_k}.AddBlock(Enc_{sk_k}(B\_5_{DO_k}))$
4     send $(TV\_c_k, H(B\_5_{DO_k})$ to TEAS
PHASE IX - TEAS:
5 $R\_p = Encode\{R_{C_h}, \dots, R_{w_i,C_h}, \dots R_{C_s}, \dots, R_{w_i,C_s}, \dots R_{NT}\}$; $rndTV\_c = (\sum_{k=1}^{n} TV\_c_k) \times R\_p$
6 $invLogP\_v = K \times \left\{\left\lceil\frac{R_{NT}}{R_{C_h}}\right\rceil, \dots, \left\lceil\frac{R_{C_h}}{R_{w_i,C_h}}\right\rceil, \dots, \left\lceil\frac{R_{NT}}{R_{C_s}}\right\rceil, \dots, \left\lceil\frac{R_{C_h}}{R_{w_i,C_s}}\right\rceil\right\}$ $invLogP\_p = Encode(invLogP\_v)$
7 $B\_5_T = (H(B\_4_T), MT(\bigcup_{k=1}^{n} H(B\_5_{DO_k})), MT(rndTV\_c), TS)$; $BCH_T.AddBlock(Enc_{sk_T}(B\_5_T))$
8 send $(rndTV\_c, H(B\_5_T))$ to E2DS
PHASE X-E2DS
9 $rndTV\_v = DecrDec(rndTV\_c)$; $rndTM\_c = EncryptRndLogsOfProbs(rndTV\_v)$
10 $B\_4_E = (H(B\_3_E), H(B\_5_T), MT(rndTM\_c), TS)$; $BCH_E.AddBlock(Enc_{sk_E}(B\_4_E))$
11 send $(rndTM\_c, H(B\_4_E))$ to TEAS
12 PHASE XI - TEAS:
13 $TM\_c = rndTM\_c + invLogP\_p$; $TM\_c = TM\_c - Rot(TM\_c, -(m+1))$
14 $B\_6_T = (H(B\_5_T), H(B\_4_E), MT(TM\_c), TS)$; $BCH_T.AddBlock(Enc_{sk_T}(B\_6_T))$
15 return $TM\_c$
```

***secT* – secure Training**. Given in Algorithm 5.2. In Phase VIII each of the EDOs locally constructs the training vector , TV_v_k, s.t. the first and the $(m+1)$-th index have the local ham $N^{(k)}(c_h)$ and spam $N^{(k)}(c_s)$ counts, respectively, while the indexes from 1 to m contain the local ham frequencies $f^{(k)}(w_i, c_h)$, and indexes from $m+2$ till $2m+2$ have the local spam frequencies $f^{(k)}(w_i, c_s)$ corresponding to the words of the selected features' set SF, respectively, where $1 \leq i \leq m$ and $1 \leq k \leq n$ (the upper vectors of Fig.5.7, lines 1-4). For the needs of Laplace Smoothing (Chapter 2.2.1 and 2.2.2) an arbitrarily chosen EDO adds an extra 1 to the indexes corresponding to word frequencies and adds the dictionary size (in our case m) to the first and $(m+1)$-th index corresponding to ham and spam counts. After encoding and encrypting TV_v_k to get TV_c_k, they are send to TEAS, which in Phase IX homomorphically sums them up and randomizes this sum to get $rndTV\_c$ (line 5, Fig.5.7), and sends it to E2DS (line 8). In Phase X E2DS finds the randomized logarithmic terms of the MNB trained model (Section IV-A), encrypts them to get $rndTM\_c$ (Fig.5.8, line 9) and sends it to TEAS (line 11).

Finally, in Phase XI TEAS removes the randomizations and subtracts the rotated result (Fig.5.8, line 13) to get the final MNB trained model according to (2.7).

Note: if instead of the frequencies $f^{(k)}(w_i, c_h)$ and $f^{(k)}(w_i, c_s)$ we put $N^{(k)}(w_i, c_h)$ and $N^{(k)}(w_i, c_s)$ at TV_v_ks in Phase I, the final trained model will be the one based on NB (Chapter 2.1.1).

Improvement 5.1: after obtaining it, the trained model TM_c is rarely changed in practice. We can utilize this fact to send it only once to the EC and amortize the communication cost among all the secure classification instances which will initiated by the EC.

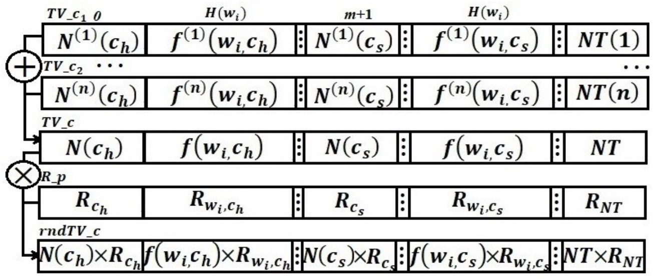
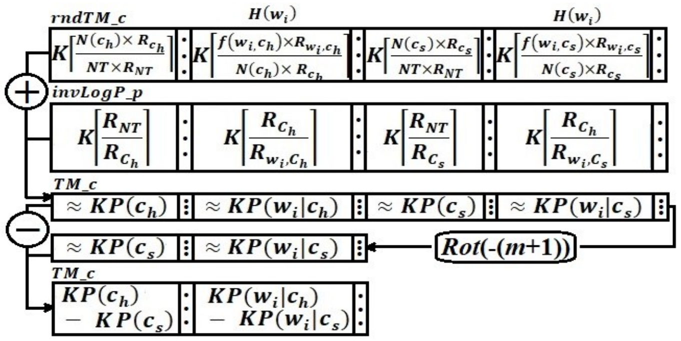


Fig.5.7. Getting and randomizing the global frequencies in SIMD fashion.



rndTM_c
$H(w_i)$
$H(w_i)$
$K\left[\frac{N(c_h)\times R_{c_h}}{NT\times R_{NT}}\right]$
$K\left[\frac{f(w_i,c_h)\times R_{w_i,c_h}}{N(c_h)\times R_{c_h}}\right]$
$K\left[\frac{N(c_s)\times R_{c_s}}{NT\times R_{NT}}\right]$
$K\left[\frac{f(w_i,c_s)\times R_{w_i,c_s}}{N(c_s)\times R_{c_s}}\right]$
invLogP_p
$K\left[\frac{R_{NT}}{R_{C_h}}\right]$
$K\left[\frac{R_{C_h}}{R_{w_i,C_h}}\right]$
$K\left[\frac{R_{NT}}{R_{C_s}}\right]$
$K\left[\frac{R_{C_h}}{R_{w_i,C_s}}\right]$
TM_c
$\approx KP(c_h)$
$\approx KP(w_i|c_h)$
$\approx KP(c_s)$
$\approx KP(w_i|c_s)$
$\approx KP(c_s)$
$\approx KP(w_i|c_s)$
$Rot(-(m+1))$
TM_c
$KP(c_h) - KP(c_s)$
$KP(w_i|c_h) - KP(w_i|c_s)$

Fig.5.8. De-randomizing and rotating $rndTM\_c$ to get the final trained model TM_c

ALGORITHM 5.3*: secT-MLMO* (secure Training for Multi-Label Multi-Output datasets)

INPUT: $\boldsymbol{n, \{DS_k\}_{k=1}^{n}, KeySet, SF^{MLMO}}$

$\boldsymbol{SF^{MLMO}}$: the set of $|L|$ set hashes, each of m^l selected features with the highest IG

OUTPUT: $\boldsymbol{TM^{MLMO}\_c} = \left\{\left\{\left\{KlogP\left(C_c{}^l\right), \left\{KlogP\left(w_i|C_c{}^l\right)\right\}_{i=1}^{m^l}\right\}_{\boldsymbol{c=1}}^{|\boldsymbol{C^l}|}\right\}_{\boldsymbol{l=1}}^{|\boldsymbol{L}|}\right\}$

$\boldsymbol{TM^{MLMO}\_c}$: the MLMO trained model ciphertext

```
PHASE VIII - EDOs:
1 for $k = 1$ to $n$ do
2     $TV^{MLMO}\_c_k = getEncryptTVAccordingToSF - MLMO(DS_k, SF^{MLMO})$
3     $B\_5_{DO_k} = (H(B\_4_{DO_k}), H(B\_4_T), H(B\_3_E), MT(TV^{MLMO}\_c_k), TS)$;
      $BCH_{DO_k}.AddBlock(Enc_{sk_k}(B\_5_{DO_k}))$
4     send $(TV\_c_k, H(B\_5_{DO_k})$ to TEAS
PHASE IX - TEAS:
5 $R\_p = Encode\left\{\left\{\left\{R_{C_c{}^l}, \left\{R_{w_i|C_c{}^l}\right\}_{i=1}^{m^l}\right\}_{\boldsymbol{c=1}}^{|\boldsymbol{C^l}|}\right\}_{\boldsymbol{l=1}}^{|\boldsymbol{L}|}, R_{NT}\right\}$; $rndTV\_c = (\sum_{k=1}^{n} TV^{MLMO}\_c_k) \times R\_p$
6 $invLogP\_v = K \times \left\{\left\{\left\{\left[\frac{R_{NT}}{R_{C_c{}^l}}\right], \left\{\left[\frac{R_{C_c{}^l}}{R_{w_i|C_c{}^l}}\right]\right\}_{i=1}^{m^l}\right\}_{\boldsymbol{c=1}}^{|\boldsymbol{C^l}|}\right\}_{\boldsymbol{l=1}}^{|\boldsymbol{L}|}\right\}$; $invLogP\_p = Encode(invLogP\_v)$
7 $B\_5_T = (H(B\_4_T), MT(\bigcup_{k=1}^{n} H(B\_5_{DO_k})), MT(rndTV\_c), TS)$; $BCH_T.AddBlock(Enc_{sk_T}(B\_5_T))$
8 send $(rndTV\_c, H(B\_5_T))$ to E2DS
PHASE X-E2DS
9 $rndTV\_v = DecrDec(rndTV\_c)$; $rndTM^{\boldsymbol{MLMO}}\_c = EncryptRndLogsOfProbs(rndTV\_v)$
10 $B\_4_E = (H(B\_3_E), H(B\_5_T), MT(rndTM\_c), TS)$; $BCH_E.AddBlock(Enc_{sk_E}(B\_4_E))$
11 send $(rndTM\_c, H(B\_4_E))$ to TEAS
12 PHASE XI - TEAS:
13 $TM^{MLMO}\_c = rndTM^{\boldsymbol{MLMO}}\_c + invLogP\_p$;
14 $B\_6_T = (H(B\_5_T), H(B\_4_E), MT(TM\_c), TS)$; $BCH_T.AddBlock(Enc_{sk_T}(B\_6_T))$
15 return $TM^{MLMO}\_c$
```

Algorithm 5.3, which deals with multi-label multi-output textual datasets goes along similar lines with Algorithm 5. Of course, it is designed for such a scenario and in itself incorporates the multi-class scenario missing from Algorithm 5.2 for textual datasets. Algorithm 5.3 can be seen as a continuation of Algorithm 4.4 which securely selects the best features according to the information gain (Chapter 2.2, eq. (2.8)) for each label.

Improvement 5.2. For computational and communicational efficiency purposes for Algorithms 5.1-5.3, whenever possible, we apply the polynomial switching technique proposed in [55], use multiple cores and threads (thus parallelize to the maximum extend), do simultaneous execution of part of the protocols in different participants whenever it's possible, i.e. adding blocks to blockchain after sending data to other participants or generating the participant's corresponding random data before/after the participants receive/send their data, and for multi-query purposes we replicate the ciphertexts at last phases of the protocols instead of from the very beginnings. Those techniques alone give an improvement of several folds with respect to the original protocols.

Improvement 5.3. If the number of slots needed for the training model is n_L (Fig.5.3), then in a single ciphertext we can pack (replicate) the trained model for $p = \frac{N}{n_L}$ time for increased throughput during the secure classification stage, where N is the polynomial modulus of the ciphertext.

Improvement 5.4. During the secure classification stage (Chapter 6) we need to homomorphically find the sum of n_s slots, where n_s is the number of slots dedicated to a single class (Fig. 5.2). An old version of the secure sum requires n_s to be a power of two, and if it is not the case dummy zero are appended up until that goal is reach, in the processing hurting the throughput of the algorithms due to those dummy zeros. In Chapter 6.2 we propose a novel secure sum algorithms (Algortihm 6.1) for which n_s is not necessarily a power of two.

5.5. Theoretical and experimental evaluations and comparisons

Table 5.1. gives the theoretical comparisons for the computation and communication costs among different schemes during the PP training. In the process we tend to use the described schemes in the most efficient and optimized way they can be utilized. However, we do this without losing the generality by making any assumption on the number of features f, the number of classes c or the cardinalities of F_i for $1 \leq i \leq f$.

For the experimental evaluations, we run our codes in a machine with Intel Core i3-4000M processor with two physical cores, each of 2.4GHz (we utilize one core only in our implementations) with 4GB of DDR3 RAM and 64-bit Wındows 10 Pro as an OS. For SWHE purposes we use C++ based Microsoft's SEAL 3.2 library [40] which implements a version of [59].

For evaluating and comparing our algorithms with the state of the art over the same benchmark datasets, we chose the textual SMS spam dataset [84-85]. After closely examining the dataset we realized that people tend to avoid vowels in their SMSes in order to make them shorter. Also, while writing the words, they make more mistakes when they write the vowels than they do with consonants. In this sense, instead of going with the usual preprocessing procedure (punctuation removal, stop-word removal, stemming, etc.), we came up with the

Table 5.1. Theoretical comparison for the costs of the PP training algorithm for NB models among different schemes

Sch.	Place	Communication	Computation
Kantarc. et.al [61] (sec. sum protocol)	**EDOs and server**	Each dataset owner does $c \cdot (1 + \Sigma\|Fi\|)$ integer transmission. In total $n \cdot c \cdot (1 + \Sigma\|Fi\|)$ transmissions	Each dataset owner and the main server in total do: $n \cdot c \cdot (1 + \Sigma\|Fi\|) + 1$ summations + $n \cdot c \cdot (1 + \Sigma\|Fi\|)$ Divisions + and $n \cdot c \cdot (1 + \Sigma\|Fi\|)$ logs
	Server	$c \cdot (1 + \Sigma\|Fi\|)$ broadcasts	
Kantarc. et. al [61] (sec sum of shares)	**EDOs and server**	Each DO $c \cdot s \cdot (1 + \Sigma\|Fi\|)$ integers. In total $n \cdot c \cdot s \cdot (1 + \Sigma\|Fi\|)$ (s is the number of shares per integer)	Each dataset owner and server do: $n \cdot c \cdot s \cdot (1 + \Sigma\|Fi\|)$ +1 summation + $n \cdot c \cdot (1 + \Sigma\|Fi\|)$ Divisions + $n \cdot c \cdot (1 + \Sigma\|Fi\|)$ logs
	Server	$\Omega(c \cdot (1 + \Sigma\|Fi\|))$ broadcasts	
Vaidya. et.al [62] (sec log)	**All participa nts**	$c \cdot (1 + \Sigma\|Fi\|) \cdot O(log(p))$ oblivious transfers (OT), which is $c \cdot (1 + \Sigma\|Fi\|) O(log(p)(t + n))$ bits (p is the size of the field used for OT, t is the security parameter, n is the number of DOs)	$c \cdot (1 + \Sigma\|Fi\|)(16 \cdot \lceil log(NT) \rceil + 2u$ exponentiations (NT is the total number of records among all dataset owners and u is the degree of the approximated Taylor series (polynomial))
Yang et.al [64]	**EDOs**	**El-Gamal**[10]: Each dataset owner transmits $\Omega(c \cdot (1 + \Sigma\|Fi\|))$ ciphertexts for a total of $\Omega(n \cdot c \cdot (1 + \Sigma\|Fi\|))$ ciphertexts transmiss.	**El-Gamal**[48]: Total of $\Omega(n \cdot c \cdot (1 + \Sigma\|Fi\|))$ encryptions at the dataset owners
	Server	Broadcast of the final trained model, which is $c \cdot (1 + \Sigma\|Fi\|)$ integers	**El-Gamal**[48]: $\Omega(n \cdot c \cdot (1 + \Sigma\|Fi\|))$ multiplication at the server + el-Gamal decryption trials for relatively small integer exponents
Yi et.al [65]	**EDOs**	**Pailler** [49]: A total of $2c \cdot (1 + f)$ ciphertexts, 1024 bit each	**Pailler** [49]: A total of $2c \cdot (1 + f)$ encryptions
	Mixers (Servers)	**Secure log** [5]: A total communication cost of $2 \cdot (c + f)$ engagements in secure log protocol as described in [5].	**Pailler** [49]: A total of $2c \cdot (1 + f)$ multiplications at both mixers (servers) **Secure log** [63]: 2$\cdot$ $(c + f)$ invocations
Liu et.al [66]	**EDOs**	**Bilinear mappings** [33]: $NT \cdot (f + 1)$ ciphertexts transmissions, where NT is the total number of transaction among all n DOs	**Bilinear mappings:** [66]: $NT \cdot (f + c)$ encryptions (original encryption scheme proposed in [33])
	Servers	**Bilinear mappings** [33]: $f + c$ ciphertexts transmission + $\Omega(c \cdot (1 + \Sigma\|Fi\|))$ Integer transmission	**Bilinear mappings** [33]: $NT(f + c)$ proxy-re-encryptions + 2 $\cdot NT \cdot (f + c)$ homomorphic multiplications + $f + c$ decryptions
Liu et.al [67]	**EDOs**	**BGV** [55]: $NT \cdot b \cdot ((f + c)/N)$ ciphertexts	**BGV** [55]: $NT \cdot b \cdot ((f + c)/N)$ SIMD encryptions, N Is the degree of the polynomial, b is the nr. of bits per feature
	Servers	**BGV** [55]: $b \cdot ((f + c)/N)$ ciphertexts + $\Omega(c \cdot (1 + \Sigma\|Fi\|))$ Integer transmissions	**BGV** [55]: $O(ncb2 + max\|Fi\|nc)(t\_add + t\_rt) + O(ncb + max\|Fi\|nc)t\_ml + O(n \cdot max\|Fi\|)\ t\_ml$, where t_add, t_rt, and t_ml are the times for a single BGV addition, rotation and multiplication, respectively
Our scheme (non-textual)	**EDOs**	**FV** [56]: n ciphertext transmission	**FV** [56]: n ciphertexts encryptions (one at each DO)
	Servers	**FV** [56]: 2 ciphertext transmission	**FV** [56]: n homomorphic additions + 2 plain multiplications + 1 encryption + 1 decryption

idea of using bigrams consisted exclusively of consonants. In this manner we extracted 441 features (21x21 consonant pairs for a total of 441 consonant bigrams) instead of the few thousands features (5000 to 20000 thousands of bag of words) that are usually used for SMS spam classification purposes [85]. Depending on whether a bigram is found or not in an SMS, its' value is 0 or 1. E.g., the sms: "hay men, whats ap; ham" has 1 for the columns (features) of those bigrams: "hy", "mn", "wh", "ht" and "ts", while for all the others it has 0s.

For comparison and benchmark purposes with the other non-textual schemes that deal with secure NB training and/or classification, we will also use the Breast Cancer Wisconsin [86] and the Acute Inflammation Disease [87] datasets.

Considering the standardizations of [80], for the polynomial degrees of $N = 4096, 8192, 16384$ we use a plain modulus of $t = 27, 37, 62$ bits, respectively, and a coefficient modulus of $q = 109, 218, 438$ bits, respectively.

For the SMS-spam dataset $C = \{spam, ham\}$, so the number of classes is $c = 2$. For the number of features we have $f = 441$, and for every feature $F_i = \{0,1\}$ so $|F_i| = 2$ for all i, s.t $.1 \leq i \leq f$. We horizontally partition the SMS dataset to simulate for $n = 10, 20, 30, 40, 50$ DOs. In order to construct the training vector which contains the frequencies (Fig.5.2 and Fig.5.3), for the SMS-spam dataset we need $\sum_{i=1}^{441}|F_i| + 1 = 883$ slots per class (Fig.5.2). However, due to the usage of an old version the *secSum* algorithm which works when the number of slots in a ciphertext for which we find the sum is a power of two, and if it's not the case dummy slots with values of zeros are padded to achieve this effect (improved in Chapter 6.2, algorithm 6.1 so the number of slots shouldn't necessarily be a power of two), per class we need $n_s = 2^{\lceil \log(\sum_{i=1}^{441}|F_i|+1) \rceil} = 1024$ slots, and will fill up the remaining slots with dummy values, preferably zeros (Fig.5.2). For both classes in total we need $n_L = \mathrm{c} \cdot n_s = 2048$ slots (Fig.5.3). This means that we need at least a polynomial of degree $N = 2048$ to construct the training vector at DOs. However, due to noise budget being consumed because of homomorphic encryptions and the chosen value t for the plain modulus dictated from the needs of the protocol (we multiply by a constant K and sum up 883 integers whose sum shouldn't surpass t), the lowest degree we can use in our scenario is $N = 4096$ (even when we deal with one query during the classification stage). For the same reasons, when $N = 4096$, the random values of R_p and h_p (needed for the secure comparison algorithm) were small, however, for higher polynomial degrees R_p and h_p can be integers of 64 or more bits, which is more than enough for real case deployments. In order to increase the throughput, for the classification stage we considered packing (batching) of $p = \frac{N}{n_L} = 2, 4, 8$ encrypted queries into one ciphertext using the *CPack* algorithm (Chapter 6.2), and for this purpose for the final ciphertext result we used polynomials of degree $N = 4096, 8192$ and 16384, respectively.

Applying the same logic, for the Breast Cancer dataset we have $f = 9$ since it has 9 features (attributes), and each of the features has 10 values, thus $F_i = \{1, 2 \dots 10\}$ so $|F_i| = 10$ for all i, s.t. $1 \leq i \leq f$. It has two classes, $C = \{malign, benign\}$, so $c = 2$. This means that for the per class portion of the training vector (Fig.5.2), hence per class query vector as well (Fig.5.3), we need $n_s = 2^{\lceil \log(\sum_{i=1}^{9} |F_i| + 1) \rceil} = 128$ slots per class, or in total its $n_L = c \cdot n_s = 256$ slots for the whole training vector $[TV(k)]$ (Fig.5.3). We also split the Breast Cancer dataset to simulate for $n = 10, 20, 30, 40, 50$ DOs. Due to relatively low number of slots used for the trained model or the encrypted query, for batching purposes we can pack $p = \frac{N}{n_L} = 16, 32, 64$ queries into one ciphertext, which for the reasons explained above, will have polynomials of degree $N = 4096$ for $p = 16$, then $N = 8192$ for $p = 32$, and $N = 16384$ for $p = 64$.

On the other hand, for the acute inflammation dataset (AID) we have $f = 6$ attributes where 5 of the 6 attributes are binary (have two values $\{yes, no\}$), while the temperature attribute is an integer varying between 35.5 and 41.5°C. If we assign 53 slots for the temperature by discretizing the integer value it takes, then knowing that we need 10 slots for the other 5 attributes, for the number of slots per class we have $n_s = 2^{\lceil \log(\sum_{i=1}^{6} |F_i| + 1) \rceil} = 64$ slots. Since AID is a multivariate dataset (it has two class labels), one label is for the Inflammation of urinary bladder (IUB), thus $C_1 = \{yes, no\}$ so $c_1 = 2$, and the other one is for the Nephritis of renal pelvis origin (NRPO), thus $C_2 = \{yes, no\}$ so $c_2 = 2$ again. In this manner for both the labels, which have two classes, we have $c_1 \cdot n_c = 128$ slots and $c_2 \cdot n_c = 128$ slots, for a total of $n_L = (c_1 + c_2) \cdot n_s = 256$ slots per query of two labels. However, here with a single query we do two classifications (labeling). For the training stage again we simulate for $n = 10,20,30,40,50$ DOs. For the classification stage we use packing of $p = \frac{N}{n_L} = 16, 32, 64$ queries into one ciphertext for polynomial degrees of $N = 4096, 8192, 16384$, respectively.

For benchmark and comparison purposes with the state-of-the-art schemes dealing with secure training over textual datasets, we used the email textual dataset Enron [37]. Its implementations can be found in [81]. for a total of around 9 000 lines of original C++ code. Enron e-mail dataset is a collection of e-mails from 150 user profiles, with a total of 16 555 ham and 17 148 spam e-mails. For the dataset pre-processing in line 2 of the $secFS - S1$ (Algorithm 4.2) we applied stop-words removal, partial punctuation removal, stemming by using the library in [45] and converting all of the letters to lowercase. For our protocols, the input parameter values for the Enron dataset are $val = 5\ and\ K1 = 100\ 000$.

For the SWHE parameters (Chapter 2.4.1) for $N = 8192,16384$ we use the corresponding values of $t = 37,60$ bits and $q = 218,438$ bits, respectively and security of $\lambda = 128$ bits.

For our implementation purposes of homomorphic operations, we chose Microsoft's SEAL 3.4 library [29] based on the modified FV scheme. Since it works only with integers, for all the datasets we had to convert the logarithms of all of the probabilities into integers by multiplying them with a constant K. When $K = 255$ we didn't have any accuracy loss due to the integerization and rounding process for the server-centric classification. For the user-centric classification that value rose to $K = 430$ due to incorporating the STC guard into our protocol against the STC attack given in [10]. Those values for the constant K are consistent with those found in literature for the Naïve Bayes model which reported that multiplying the logarithms of the probabilities with an 8-10 bits constant is enough to avoid any loss of classification accuracy [69].

Fig.5.9 gives the computation cost for *PPTMDO* (Algorithm 5.1). Since the computation in Phase I is done simultaneously at all EDO, for EDOs we take the average cost with respect to the number of EDOs involved. As it was expected, the average cost at EDOs and at EDS remains pretty much the same (constant) among different number of EDOs, while the cost at TACS linearly increases with the number of EDOs since TEAS has to aggregate (sum up) all of the ciphertexts send from the EDOs. For the communication cost, we have n ciphertext of the same size transmitted from each DO to TEAS, one from TECS to E2DS and one from E2DS back to TEAS, for a total of $(n + 2) \cdot ciphertext_{size}$ transmitions. The

$ciphertext_{size}$ size is calculated as $ciphertext_{size} = 2 \cdot N \cdot q$ $bits$ [56], where N is the polynomial modulus and q is the coefficient modulus.

Regardless of the dataset type or size, the number and type of the expensive homomorphic operations is the same in PPTMDO. So, as expected, when experimenting with datasets in [84-85], [86] and [87], we got the same communication costs and roughly the same computation cost as they are shown in Fig.5.9.

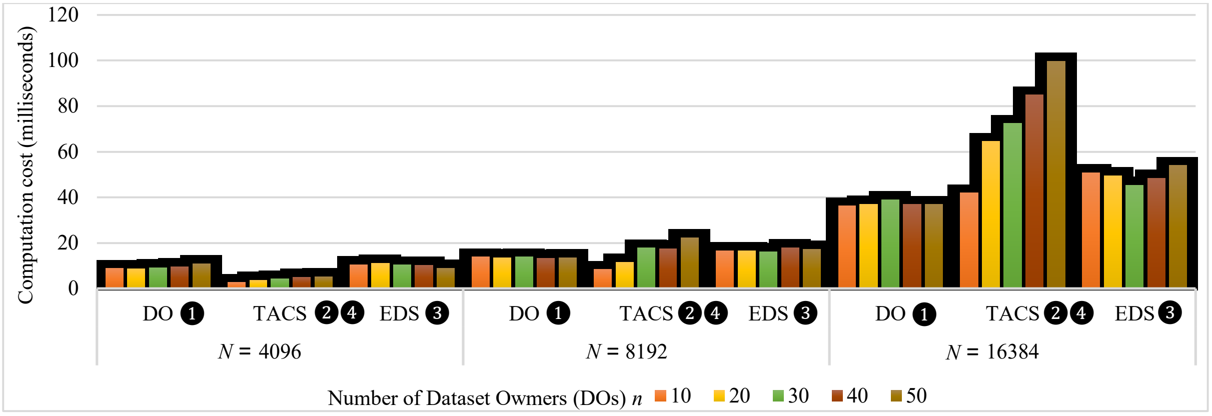


Figure 5.9. Computation cost of the participants in *PPTMDO* (Algorithm 5.1) for different polynomial sizes and number of EDOs for datasets in [84-85], [86] and [87]

Table 5.2 gives the cumulative (total) computation and communication costs for each participant while running Algorithm 5.2. for the Enron email [37] and SMS dataset [84-85]. It report results for different polynomial sizes N and numbers of EDOs n. The size of a single ciphertext is $2 \cdot N \cdot q$ bits. In Algorithm 5.2, each participant transfers one ciphertext each, for a total (cumulative) communication cost of $(n + 2)$ ciphertext transmissions (n EDOs, TEAS and E2DS), which are reported in the corresponding columns in Table 5.2 in MB.

Table 5.2. Algorithm 5.2 costs for different polynomial sizes *N* and EDO numbers *n*

n \ *N*	**Comput. (ms) *N*=8192**			**Commun. (MB)**	**Computation (ms)*N*=16384**			**Commun. (MB)**
	EDO	**TEAS**	**E2DS**		**EDO**	**TEAS**	**E2DS**	
Enron email dataset [37]								
10	1690	149	65	5.1	803	557	53	20.5
20	1382	212	62	9.3	582	646	73	37.6
30	1044	198	70	13.6	530	1581	52	54.7
40	941.0	203	71	17.8	413	1966	55	71.8
50	283.3	235	63	22.4	392	1752	48	88.9
SMS spam corpus dataset [84-85]								
10	16.8	7.8	4	5.1	36.5	34.9	11.9	21.9
20	16.7	9.5	4.2	9.4	35.8	42.1	1.7	40.2
30	16.4	13.9	3.7	13.7	36.1	53.4	10.3	58.5
40	16.3	17.4	4.1	18.0	36.5	65.3	10.8	76.8
50	16.1	19.8	3.9	22.3	37.8	71.0	11.0	95.0

Table 5.3 gives experimental comparisons for the PP training costs among different schemes and datasets among non-textual datasets, where our schemes are represented by *secT*, (Algorithm 5.2)

Table 5.3. Cumulative experimental results among all participants for PP training of non-textual datasets for n = 5 EDOs. Our scheme is represented by Algorithm 5.1.

Cost \ **Scheme**	Yang et.al [32]	Liu et. al. [23]	**Our**	Liu et. al. [33]	**Our**
Dataset	Breast Cancer Wisconsin Data Set [86]			Acute Inflammations [87]	
Computation.	≈1.8 s	2951.8 min	**22.15 ms**	8.848 sec.	**22.47 ms**
Communication	≈ 7.76 MB	267.4 MB	**763 KB**	968 KB	**763 KB**

Table 5.4 gives experimental comparisons for the PP training costs among different schemes applicable to textual datasets and our schemes are represented by *secT*, (Algorithm 5.2)

Table 5.4. Secure training comparisons among different schemes

Scheme	Comp. cost	Comm. cost	ML algorithm	Class. Acc.
Enron email dataset [37]				
[14]*	11.1 days	120 GB	Deep Learning	86.3%
[14]**	5.04 days	120 GB	Deep Learning	86.3%
Ours	**10.16 min**	**316 MB**	MNB	**99.1%**
SMS spam corpus dataset [84-85]				
[15]	21.57 ms	763 KB	NB	93.1%
Ours	**18.20 ms**	**709 KB**	NB	**93.1%**
MNIST [88]				
[88]	55.5 days	Not report.	Deep Learning	96.3%

*utilizing 104 cores of Intel Xeon processors with 2.2 GHz and 482 GB of RAM
**Improved version of the same scheme over the same hardware resources

5.6. Security analysis and proofs

While proving the security of our protocols given in this Chapter, we have in mind the definitions, concepts and Theorems given in Chapter 2.4.2

Theorem 5.1: *PPTMDO* (Algorithm 5.1) is a secure multi-party protocol (SMC) under the semi-honest model

Proof: Here we compute the probabilistic function $f(Pk, \{DB_i\}_{i=1}^{n}, \phi, sk) = f\left(\{f_{DO_i}\}_{i=1}^{n}, f_{TACS}, f_{EDS}\right) = (\{\phi\}_{i=1}^{n}, TM\_c, \phi)$ using the protocol $PPTMDO$, where ϕ means no input or output for the corresponding participant, respectively. For the outputs of the corresponding protocol we have $output^{PPTMDO} = \left(\{output_{DO_i}^{PPTMDO}\}_{i=1}^{n}, output_{TACS}^{PPTMDO}, output_{EDS}^{PPTMDO}\right) = (\{\phi\}_{i=1}^{n}, TM\_c, \phi)$. For the views of all of the DOs we have: $\{V_{DO_i}^{PPTMDO}\}_{i=1}^{n} = \{(Pk, DB_i, r_{DO_i})\}_{i=1}^{n}$. For the views of TACS and EDS we have $V_{TACS}^{PPTMDO} = (PK, r_{TACS}, [TV(k)]_{k=1}^{n}, rndTM\_c\ )$, $V_{EDS}^{PPTMDO} = (Pk, sk, r_{EDS}, rndGTV\_c)$, respectively. Since the DOs don't receive any message or don't have any output, for them we give the trivial simulator $\{S_{DO_i}^{PPTMDO}\}_{i=1}^{n} = \{(Pk, DB_i, \widetilde{r_{DO_i}})\}_{i=1}^{n}$, where r_{DO_i} and $\widetilde{r_{DO_i}}$ are from the same distribution. For the simulator of TACS we have $S_{TACS}(Pk, f_{TACS}) = (Pk, \widetilde{r_{TACS}}, [\widetilde{TV(k)}]_{k=1}^{n}, \widetilde{rndTM}\_c)$, where $\widetilde{r_{TACS}}$ has the same distribution as r_{TACS}, while $[\widetilde{TV\_c_k}]_{k=1}^{n}$ are randomly generated ciphertexts which are indistinguishable from their $[TV\_c_k]_{k=1}^{n}$ counterparts due to the semantic security of the RLWE schemes. Since TACS as an output has the TM_c then we have $rnd\widetilde{TM}\_c = TM\_c - rndLogsOFInvProbs\_p$, thus $\{S_{TACS}(Pk, f_{TACS}), f(\{DB_i\}_{i=1}^{n}, \phi, sk)\} \cong_C \{V_{TACS}^{PPTMDO}$, $output^{PPTMDO}$). For the simulator of EDS we have $S_{EDS}(Pk, sk, f_{EDS}) = (Pk, sk, \widetilde{r_{EDS}}, rnd\widetilde{\_GT}V\_c)$, where random r_{EDS} and $\widetilde{r_{EDS}}$ are from the same distribution, while $rnd\widetilde{\_GT}V\_c$ is a random ciphertext, thus $\{S_{EDS}(sk, f_{EDS}), f(\{DB_i\}_{i=1}^{n}, \phi, sk)\} \cong_C \{V_{TACS}^{PPTMDO},\ output^{PPTMDO}$) ■

Corollary 5.1: If up to $n-1$ DOs collude, the *PPTMDO* is still secure under the semi-honest model.

Proof: Without loss of generality let's assume that the colluding DOs are $i = 2, \ldots, n$ and let the common view of them be $V_{\overline{DO}}^{PPTMDO} = \left(Pk, \overline{DB}, r_{\overline{DO}}\right)$, where $\overline{DB} = \bigcup_{i=2}^{n} DB_i$. Let the view of the non-colluding DO be $V_{DO_1}^{PPTMDO} = \left(Pk, DB_1, r_{DO_1}\right)$. Since the DOs don't get any output from the function that needs to be calculated, their simulator is the trivial one (outputting only the private inputs and the random number generator from the same distribution as the views). Views and simulators for TACS and EDS are the same as in Theorem 5.1. ■

Theorem 5.2: *secT* (Algorithm 5.2) is secure under the semi-honest model.

Proof: For $1 \le k \le n$, $V_{EDO_k}^{\mathrm{secT}}(\lambda, \bar{x}) = \emptyset$ is the view, $x_{EDO_k}^{secT} = \{PreProcess(DS_k), SF\}$ is the private input and $O_{EDO_k}^{\mathrm{secT}}(\lambda, \bar{x}) = \emptyset$ is the output of EDO$_k$. The trivial simulator is $S_{EDO_k}^{\mathrm{secT}} = \emptyset$. For TEAS $V_{TEAS}^{\mathrm{secT}}(\lambda, \bar{x}) = \{(TV\_c_k)_{k=1}^{n}, rndTM\_c\}$ is the view, $x_{TEAS}^{secT} = \emptyset$ is the input and $O_{TEAS}^{\mathrm{secT}}(\lambda, \bar{x}) = TM\_c$ is the output. For the simulator we construct random RLWE ciphertexts, thus $S_{TEAS}^{secT}\left(\lambda, x_{TEAS}^{secT}, O_{TEAS}^{secT}(\lambda, \bar{x})\right) = \left\{\left(\widetilde{TV\_c_k}\right)_{k=1}^{n}, rnd\widetilde{TM}\_c\right\} \cong_c V_{TEAS}^{secT}(\lambda, \bar{x})$. For E2DS $V_{E2DS}^{\mathrm{secT}}(\lambda, \bar{x}) = \{rndTV\_c\}, S_{E2DS}^{\mathrm{secT}} = \left\{rnd\widetilde{TV}\_c\right\} \cong_c V_{TEAS}^{\mathrm{secT}}(\lambda, \bar{x})$ ■

Corollary 5.2: *secT* is secure under the semi-honest model when out of n, $n-3$ EDOs fail or $n-2$ EDOs collude ■

Theorem 5.3: Our end-to-end protocol (unprocessed datasets till the final trained model), is secure under the semi-honest model.

Proof. The end-to-end protocol, denoted as $E2E$, sequentially calls *GRPV* (Algorithm 4.1), *secFS-S1* (Algorithm 4.2)*, secFS-S2* (Algorithm 4.3) and *secT* (Algorithm 4.3), while their security was proven in Theorems 4.1-4.3 and Theorem 5.2, respectively. We invoke Theorem 2.1 to prove the security of *E2E*. ■

Corollary 5.3: $E2E$ protocol is secure under the semi-honest model when out of n, $n-3$ EDOs fail or $n-2$ EDOs collude.

Proof: We use Corollaries 4.1, 4.2 and 5.2, then invoke Theorem 2.1. ■

Chapter 6

SECURE AND PRIVATE MACHINE LEARNING CLASSIFICATIONS

Definition 6.1: Classification is the process of assigning a label to an unlabeled query q according to a trained model TM using an algorithm $\mathcal{C}$, thus $C_{TM}(q) = \mathcal{C}(TM, q)$.

In this chapter, we introduce the strict security, privacy and efficiency requirements we set up for the classification stage. In order to do so we propose several novel building blocks based on arithmetic circuits used frequently by ML classification algorithms, which we put in two groups: the ones belonging to general purpose one and the ones belonging to secure linear algebra. They all work in SIMD fashion (enabled by the SIMD properties of SWHE schemes proposed in Chapter 2.4.1), thus allow for a single instruction (algorithm, block) to be executed oved multiple data (objects, instances). In the general purpose building blocks, we introduce blocks, such as secure sum, secure comparison, secure comparison of all data slots, secure sorting, secure top-K, secure argmax, secure ciphertext permutation and secure ciphertext replication. Among others, in group of building blocks belonging to the secure linear algebra we introduce secure inner (dot) product, secure matrix-vector product, secure matrix-matrix product, secure matrix transpose, secure cascading matrix-matrix product, etc. We then utilize those building block to for our secure classification protocols which deal with non-textual, textual, multi-label multi-output datasets as well as secure classifications that can be expressed in terms of linear algebra. In the process our algorithms show flexibility in terms of being server centric or client centric, depending on where the bulk of the operations are done. To the best of our knowledge, for the first time in literature for the NB classifier and for the multiple user (query) scenario where multiple users (queries) simultaneously process their queries in secure fashion, we deal with malicious users that arbitrarily deviate from the protocol (algorithm) with the aim of fully or partially retrieving data which they are not supposed to or with the aim of totally sabotaging the algorithm for other participants. Our theoretical comparison and extensive experimental evaluations give an edge to our algorithms from several times to orders of magnitude with respect to the state of the art in term of computation and communication costs.

6.1. Introduction

Main requirements for our secure classification algorithms are:

- Privacy of the trained model
- Privacy of the user query for both query features and their corresponding values (frequencies, counts, etc.)
- Security and privacy of intermediate results
- Privacy of the output, i.e. the final classification result
- No loss of accuracy with respect to the plain classifier
- Have other properties related to secure classification mentioned in Table 3.3

6.2. System architecture, adversary models and protocol-flows-at-a-glance

Participants: **1) TEAS (The Edge Aggregating Server)**: a server used to do the bulk of the proposed protocols' homomorphic computation. **2) E2DS (The Edge Encryption Decryption Server):** It's the only participant in the system that has a pair of public keys with SWHE properties (Chapter 2.4.1). All the data that that are homomorphically evaluated in our protocols are encrypted using E2DS' public key, thus it's the only one that can decrypt them. **3) EC (Edge Client)**: has an unclassified query that he wishes to classify in secure and private manner. Since the trained model TM_c doesn't change frequently, EC keeps the trained model in encrypted form using E2DS public key with SWHE properties. .All of them are illustrated in Fig.6.1.

Adversary models: All the participants are assumed to be in the passive **semi-honest** (honest but curious) **model**, which means that they follow the protocol but on the background they try to infer some private data which they are not supposed to. A formal definition of the semi-honest model is given in Chapter 2.4.2. We assume that TEAS and E2DS don't collude. Also, during the secure classification stage we assume a more active EC that performs the STC-attack proposed in [10]. Furthermore, for the server based classification and for the multi-user (query) classification we deal with malicious users. The motivations for participant to behave in the described manners are given in [8-18]. All the participants have a pair of public/secret keys used for secure communications. Additionally, E2DS has a pair of public/secret key with SWHE properties (Chapter 2.4.1). All of these key pairs form the *KeySet* set (Chapter 5 and 6). While we designed our protocols having in mind primarily binary textual datasets, they are also applicable to non-textual ones and can be easily generalized to multi-class scenarios. This choice was done for benchmark purposes with the related research.

Protocol flows at-a-glance. ***secC* (secure classification)**: Shown in Fig.6.1. EC multiplies its query q_p with the trained model TM_c ⓬ and sends the result to TEAS ⑫ for homomorphic processing and randomization ⓭.The randomized query result is send to E2DS while the randomizing data to EC ⑬. E2DS decrypts the randomized query result ⓮ and sends it back to the EC ⑭**,** which in turn de-randomizes the query result to get the final classification ⓯. Depending on where the bulk of the execution is done, which ML classifier we are dealing with, the number of ECs in our algorithms as well as other factors and scenarios, we offer several flavors of secure classification algorithms which slightly change their order of execution, but the main idea remains the same. They are elaborated in details in their corresponding sub-chapters of this chapter.

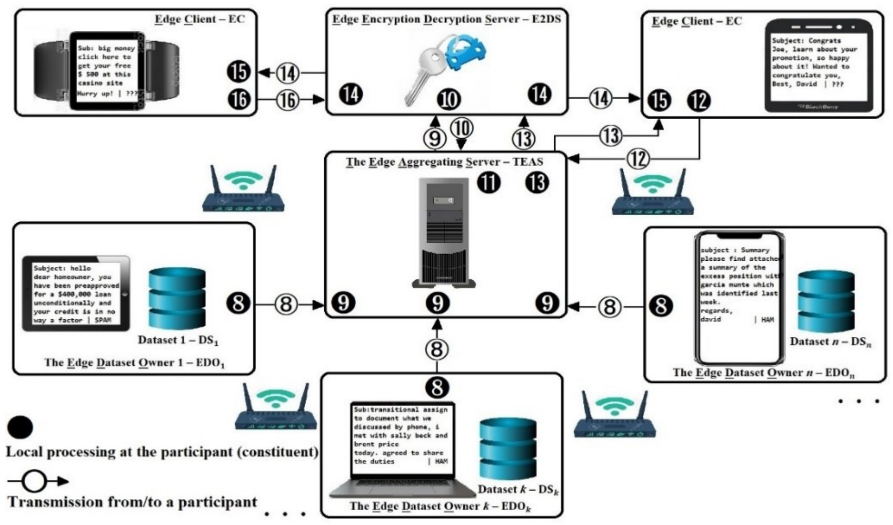

Edge Client – EC
Edge Encryption Decryption Server – E2DS
Edge Client – EC
The Edge Aggregating Server – TEAS
Dataset 1 – DS$_1$
The Edge Dataset Owner 1 – EDO$_1$
Dataset n – DS$_n$
The Edge Dataset Owner n – EDO$_n$
Dataset k – DS$_k$
The Edge Dataset Owner k – EDO$_k$
Local processing at the participant (constituent)
Transmission from/to a participant

Fig. 6.1. Protocol flows for our secure classification algorithms

6.3. General purpose secure building blocks

6.3.1. Secure sums of blocks of *d* slots

Given in Algorithm 6.1. All of the known schemes in literature the deal with finding the sum of d numbers encoded and encrypted in SIMD fashion according to a SWHE scheme (Chapter 2.4.1), assume that d is a power of. If it is not the case, then slots of dummy zeros are padded up until it is the case two [18], [89-90]. This results in waste of slots and in throughput due to the padded dummy zeros since they take the place of beneficial (real) data that can be encoded into those slots. In Algorithm 6.1 we overcome this drawback since we don't make the assumption of *d* being a power of two, hence we don't lose any slots. In this way we can pack and simultaneously process (find the sums of) *N/d* sets of integers with *d* elements (integers) in SIMD fashion where *d* is not necessarily a power of two.

In order to find the sum of d encoded slots, we use the binary representation of d. For this purpose in line 1 we find the number of bits of d, denoted as $nrBits$, and s which has one in its most significant bit and zeros at the other bits, thus it's a power of two. In log$s$ iterations (rotations and additions) we find the sum of s slots and if in a certain iteration the corresponding bit of d is one, than we store the corresponding intermediate result of sums in a temporary ciphertext, denoted as $tmp\_c[i]$ (lines 2-5), To the resulting sum of s slots we add each stored temporary ciphertext after rotating them with the corresponding offset (lines 7-9).

ALOGRITHM 6.1: ***secSum*** **(Secure Sums of blocks)**

INPUT: $\boldsymbol{input\_c, d}$

$\boldsymbol{d}$: the number of slots per block for which we find the sum, not necessarily a power of 2

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 $result\_c = input\_c$; $s = \lfloor log_2^d \rfloor$; $nrBits = bitSizeOf(d)$
2 for $i = 0$ to $s - 1$ //inclusive
3     $if(bit(i, d) == 1)$
4         $tmp\_c[i] = result\_c$
5     $result\_c = result\_c + Rot(result\_c, -2^i)$
6 for $i = nrBits - 2$ to 0
7     $if(bit(i, d) == 1)$ //the LMB is the zeroth one
8         $result\_c = tmp\_c[i] + Rotate(result\_c, -2^i)$
9 $mask\_p = secSumMask(d)$
10 $result\_c = result\_c \times mask\_p$
```

In this manner we have the sum of *d* slots at the beginning (first slot) of the corresponding block. Finally, if we want to remove the intermediate results, we multiply the resulting ciphertext with a plaintext mask which, starting from the first slot, has ones (1s) after each d slots and all other slots have value of zero (lines 9-10). In Fig.6.2a) we illustrate *secSum* for *d*=6, or in binary *d*=6=$(110)_2$.

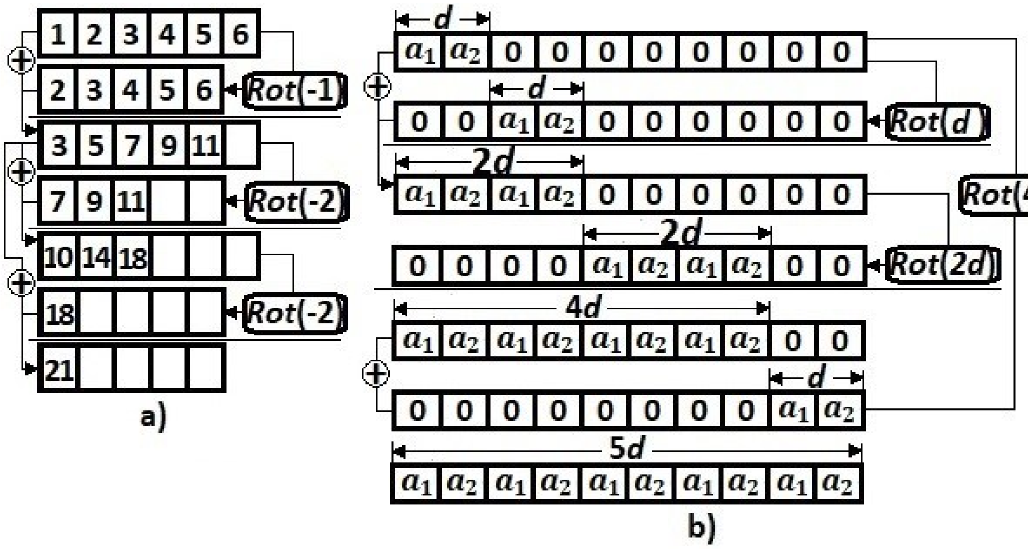


Fig. 6.2. Illustration of a) *secSums* for *d*=6=$(110)_2$ b) *CRep* for d=2 and *r*=5=$(101)_2$

Improvement 6.1. The Algorithm 6.1 does not waste slots, but it can have a relatively high cost in terms of numbers of rotations/additions per *d* slots when the number of ones in the bit representation of *d* is relatively high, a property which is not often desired. As a consequence, we can have a trade-off between the wasted slots and the average number of rotations/additions per d slots**.** In general, experimental results show that the approach of Algorithm 6.1 is the best when d is between 0.5 and 0.75 the value of its closest power of two that is greater than d. This makes Algorithm 6.1 to be done in logarithmic time with respect to *d*.

Improvement 6.2. The multiplication with the mask (lines 9-10) can be skipped and merged with the subsequent algorithm, which comes after *secSum*.

6.3.2. Ciphertext Replication

Given in Algorithm 6.2, it replicates for *r* times an input ciphertext which is assumed to have *d* data slots at its begging and the upcoming $d \cdot r$ slots are all zeros. The approach in [18] assumes that *r* is a power of two, and if it is not the case then dummy zeros are appended until it's the case. Using a similar approach as we did in Algorithm 6.1, in Algorithm 6.2. we provide the secure replication algorithm for which the replication rate r is not necessarily a power of two. Initially we find the number of bits in bit representation of *r,* then *s* in similar fashion as it was done in algorithm 6.1 (line 1). We proceed with replicating the input ciphertext for *s* times in log*s* iterations and in each iteration we keep (save) the temporary result if the bit of *r* corresponding to that iteration is one (lines 2-5).

ALOGRITHM 6.2: ***CRep*** **(<u>C</u>iphertext <u>Rep</u>lication)**

INPUT: $\boldsymbol{input\_c, d, r}$
$\boldsymbol{input\_c}$: a ciphertext that will be homomorphic. replicated
$\boldsymbol{d}$: number of data slots in $input\_c$ (starting from the first slot (the one with index zero))
$\boldsymbol{r}$: the replication rate

OUTPUT: $\boldsymbol{result\_c}$
$\boldsymbol{result\_c}$: $input\_c$'s d data slots replicated for r times

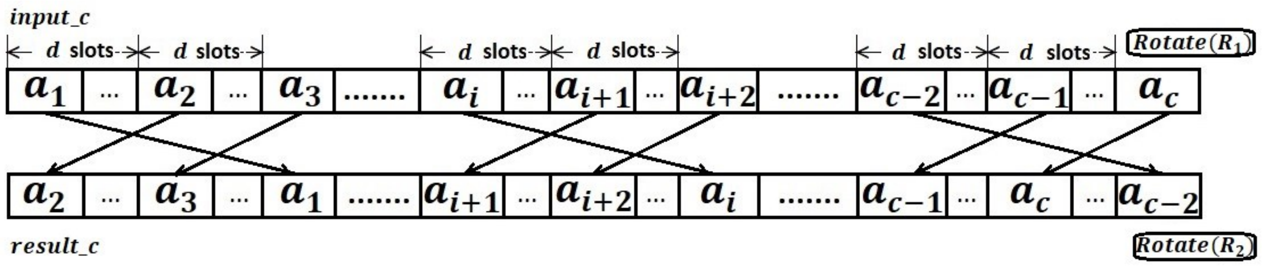
```
1 $result\_c = input\_c$; $s = \lfloor log_2^r \rfloor$; $nrBits = bitSizeOf(r)$
2 for $i = 0$ to $s - 1$ //inclusive
3     $if(bit(i, d) == 1)$
4         $tmp\_c[i] = result\_c$
5     $result\_c = result\_c + Rotate(result\_c, 2^i \times d)$
6 $dist = 2^s \times d$
7 for $i = nrBits - 2$ to 0
8     $if(bit(i, d) == 1)$ //the LMB is the zeroth bit
9         $result\_c = result\_c + Rotate(tmp\_c[i], dist)$
10        $dist = dist + 2^i \times d$
```

Finally, to the replicated ciphertext of *s* times we add each stored temporary ciphertext after rotating them with the corresponding offset (lines 7-10). We illustrate *CRep* for *r*=5=$(101)_2$ and *d*=2 in Fig.6.2b)

Improvement 6.3. We use the same logic (approach) as it was done in Improvement 6.1 for the trade-off between the average rotations/additions per *r* and the case when r is padded with dummy zeros to be a power of two.

6.3.3. Secure Random Ciphertext Permutation and its inverse

Given in Algorithm 6.3. The input ciphertext is organized in such a way that, starting from the first slot (slot with index 0), we have a total of c data slots separated by d slots from each other (Fig.6.3). This algorithm firstly rotates the input ciphertext for R_1 slots. Afterwards it divides the input ciphertext into blocks of m data slots and then permutes each data slot inside the block according to a random vector $k = \{k_1, \dots, k_m\}$ (Fig.6.4). Finally, does another rotation for R_2 slots. R_1 and R_2 are random multiples of d (Fig. 6.3). Vector k tells by how much each of the m slots of every block should be rotated inside its block. The pseudocode is given in Algorithm 6.3 and a detailed illustration of only the random block permutation (lines 3-6) for $m = 3$ and $k = \{2, -1, -1\}$ is given in Fig.6.4. Algorithm 6.3 is an improvement of the *SRCPer* algorithm given in [18] which instead of using m masks as it done in Algorithm 6.3, it uses one mask and $m - 1$ computationally costly rotations of the input ciphertext to get the same effect as we get in lines 3-5.



input_c
d slots
d slots
d slots
d slots
d slots
d slots
Rotate(R_1)
a_1 ... a_2 ... a_3 .... a_i ... a_{i+1} ... a_{i+2} .... a_{c-2} ... a_{c-1} ... a_c
a_2 ... a_3 ... a_1 .... a_{i+1} ... a_{i+2} ... a_i .... a_{c-1} ... a_c ... a_{c-2}
result_c
Rotate(R_2)

Fig. 6.3. Illustration of *SRCPer* for $m = 3$, $k = \{2, -1, -1\}$. k, m, R_1 and R_2 are random.
d is the number of slots between two neighboring data slots.

ALGORITHM 6.3: ***SRCPer*** **(Secure Random Ciphertext Permutation) and** ***invSRCPer*** **(inverse SRCPer)**

INPUT: $\boldsymbol{input\_c, k, R_1, R_2, c, d}$

$\boldsymbol{k} = \{\boldsymbol{k_1}, \dots, \boldsymbol{k_m}\}$: a random vector of m elements, gives the rotation index for each of the m data slots inside the block

$\boldsymbol{R_1}, \boldsymbol{R_2}$: random numbers by which the slot rotations are done in the beginning and the end. They are multiples of n_s

$\boldsymbol{c}$: the number of data slots (slots that carry data for us)

$\boldsymbol{d}$: the number of slots between two neighboring data slots

OUTPUT: $\boldsymbol{result\_c}$

$\boldsymbol{result\_c}$: is the finally permuted vector

```
1 $masksSRCPerVec\_p[] = SRCPerMasks(k, c, d)$ //generates m masks
2 $input\_c = Rotate(input\_c, R_1)$
3 for $i = 1$ to m
4     $tmp\_c[i] = input\_c \times masksSRCPerVec\_p[i]$
5     $tmp\_c[i] = Rotate(tmp\_c[i], k_i)$
6 $result\_c = AddMany(tmp\_c[])$ //log$m$ additions
7 $result\_c = Rotate(result\_c, R_2)$
```

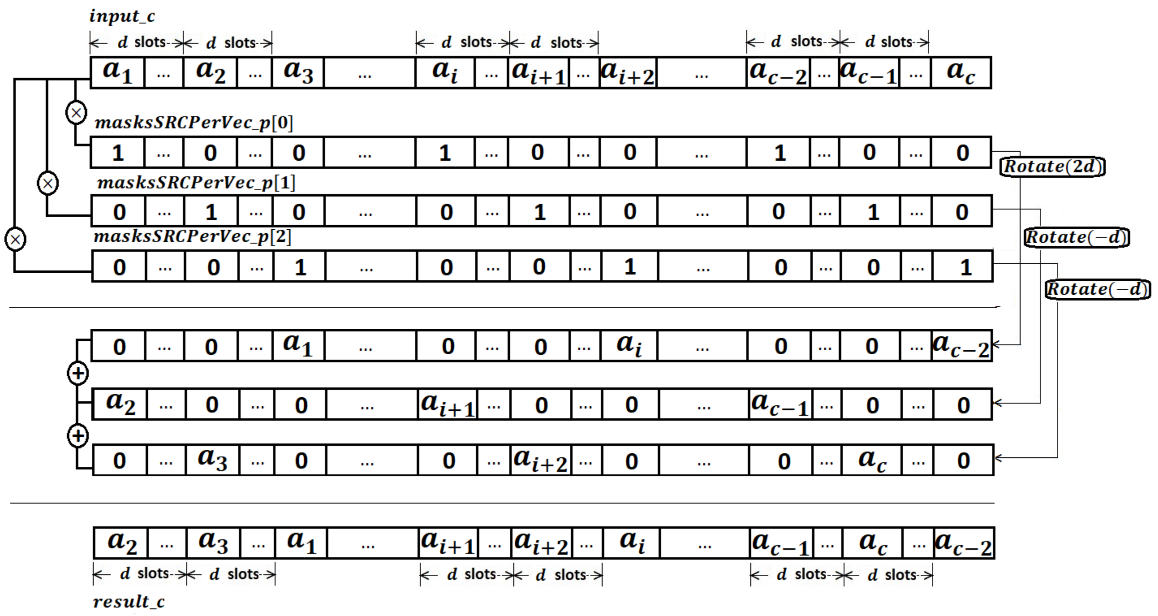


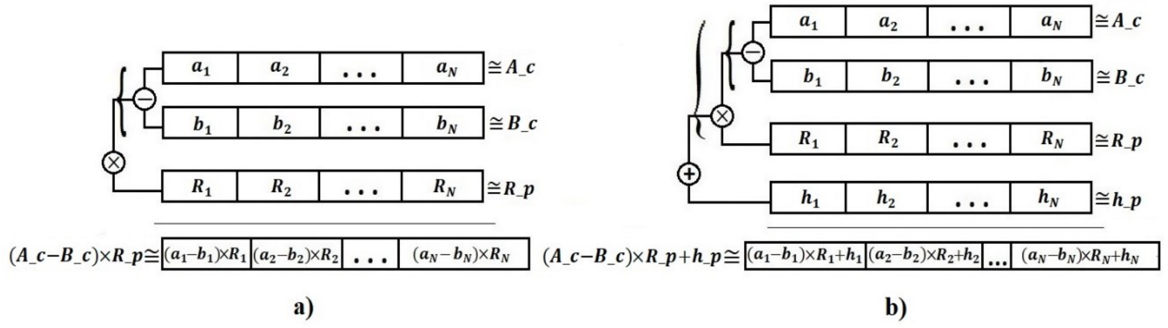
Fig.6.4. Detailed illustration (masks, multiplications, addition portion) of the block permutations of *SRCPer* for block size $m = 3$ and rotation index vector $k = \{2, -1, -1\}$ done in SIMD fashion.

The inverse of the *SRCPer* algorithm, called *invSRCPer*, is the same as *SRCPer*, but with R_1 substituted with $-R_2$ in line 2, and $-R_2$with $-R_1$ in line 7. Also, throughout the inverse protocol vector $k = \{k_1, \dots, k_m\}$ is substituted with $invK = \{k_1', \dots, k_m'\}$, s.t. $k_{i+k_i}' = -k_i$, for $1 \le i \le m$.

In order to reduce the communication cost, *SRCPer* and *invSRCPer* can be modified in a way that instead of a ciphertext, as an input it can take an integer which would represent the input index of a data slot we are interested in. In this manner we can find out the new index of this data slot at the end of the execution of *SRCPer* or *invSRCPer*

6.3.4. Secure SIMD Comparison

The comparison technique that [77-79] use to compare two SIMD encrypted ciphertexts A_c and B_c is $A\_c \cdot R\_p - B\_c \cdot R\_p$, where R_p is a random plaintext. In order to reduce the number of multiplications we propose to do it as $(A\_c - B\_c) \cdot R\_p$ (Fig.6.5a). If the result is positive, then $A > B$ and vice-versa. However, if we compare A_c and B_c several times with this technique, then there is a possibility of an adversary to factor the terms $(A - B)$ and/or R_p, which is a leakage, so we advise using it only for one-time comparisons. To overcome this weakness, we propose comparing by $(A\_c - B\_c) \cdot R\_p + h\_p$ in SIMD fashion as shown in Algorithm 6.4 and illustrated in Fig.6.5b), where R_p and h_p are random s.t. $R\_p > 0$ and $|h\_p| < R$. If the final term (result) is positive then $A > B$, and vice-versa. If the polynomial size is N, then we do N comparisons in SIMD fashion.



a_1 a_2 ... a_N $\cong A\_c$
b_1 b_2 ... b_N $\cong B\_c$
R_1 R_2 ... R_N $\cong R\_p$
$(A\_c - B\_c) \times R\_p \cong$ $(a_1-b_1)\times R_1$ $(a_2-b_2)\times R_2$ ... $(a_N-b_N)\times R_N$
a)
a_1 a_2 ... a_N $\cong A\_c$
b_1 b_2 ... b_N $\cong B\_c$
R_1 R_2 ... R_N $\cong R\_p$
h_1 h_2 ... h_N $\cong h\_p$
$(A\_c - B\_c) \times R\_p + h\_p \cong$ $(a_1-b_1)\times R_1+h_1$ $(a_2-b_2)\times R_2+h_2$ ... $(a_N-b_N)\times R_N+h_N$
b)

Fig. 6.5 Simultaneous a) secure one-time comparison b) secure comparison (*secComp, sC*) of N integer pairs in SIMD fashion

ALGORITHM 6.4: ***secComp*** **(Secure Comparison)**

INPUT: $\boldsymbol{input1\_c, input2\_c}$
$\boldsymbol{input1\_c, input2\_c}$: the input ciphertext containing data to be compared in each slot. One can be plaintext

OUTPUT: $\boldsymbol{result\_c}$
$\boldsymbol{result\_c}$: contains index (component) wise secure comparison results of the input ciphertext

```
1 $(R\_v, h\_v) = rndSecCompVectors()$; // $R\_v = \{(R_i)_{i=1}^{N}\}$, $R\_v = \{(h_i)_{i=1}^{N}\}$ s.t. $R_i > 0$ and $|h_i| < R_i$
2 $(R\_p, h\_p) = EncEncr(R\_v, h\_v)$
3 $result\_c = ((input1\_c - input2\_c) \times R\_p) - h\_p$
```

6.3.5. Secure comparison of all data slots (SCADS)

SCADS securely compares all of the data slots using the secure comparison (*secComp*) Algorithm of Chapter 6.3.4. Similarly to the *SRCPer* algorithm, the data that we want to compare are at each $\boldsymbol{d}$-th slot, starting from slot 0. In total we have $\boldsymbol{c}$ such data slots. The pseudocode is given in Algorithm 6.5 and the corresponding illustration in Fig.6.6. We give here an improved version of SCADS from [18] where instead of rotating the resulting ciphertext by one slot to the right as it is done in line 5, in [18] they rotate it to the right by a rotation for a number of slots which is greater than one, which in turn is computationally costlier for several times.

ALGORITHM 6.5: ***SCADS*** **(Secure Comparison of All Data Slots)**

INPUT: $\boldsymbol{input\_c, c, d}$
$\boldsymbol{c}$: is the total number of data slot values that needs to be compared. In total we should do $c/2$ SIMD comparisons
$\boldsymbol{d}$: the distance in number of slots between two neighboring slots

OUTPUT: $result\_c$
$result\_c$: contains the comparisons between all of the data slot (slots that are multiple of d, starting from slot 0)

```
1 $tmp\_c = input\_c$
2 for $i = 1$ to $c/2$ do
3     $tmp\_c = Rotate(tmp\_c, -d)$   //rot. left for $d$ slots
4     $result\_c = (input\_c - tmp\_c) + result\_c$
5     $result\_c = Rotate(result\_c, 1)$
6 $(R\_v, h\_v) = rndSecCompVectors()$; // $R\_v = \{(R_i)_{i=1}^{N}\}$, $R\_v = \{(h_i)_{i=1}^{N}\}$ s.t. $R_i > 0$ and $|h_i| < R_i$
7 $(R\_p, h\_p) = EncEncr(R\_v, h\_v)$
8 $result\_c = (result\_c \times R\_p) + h\_p$
```

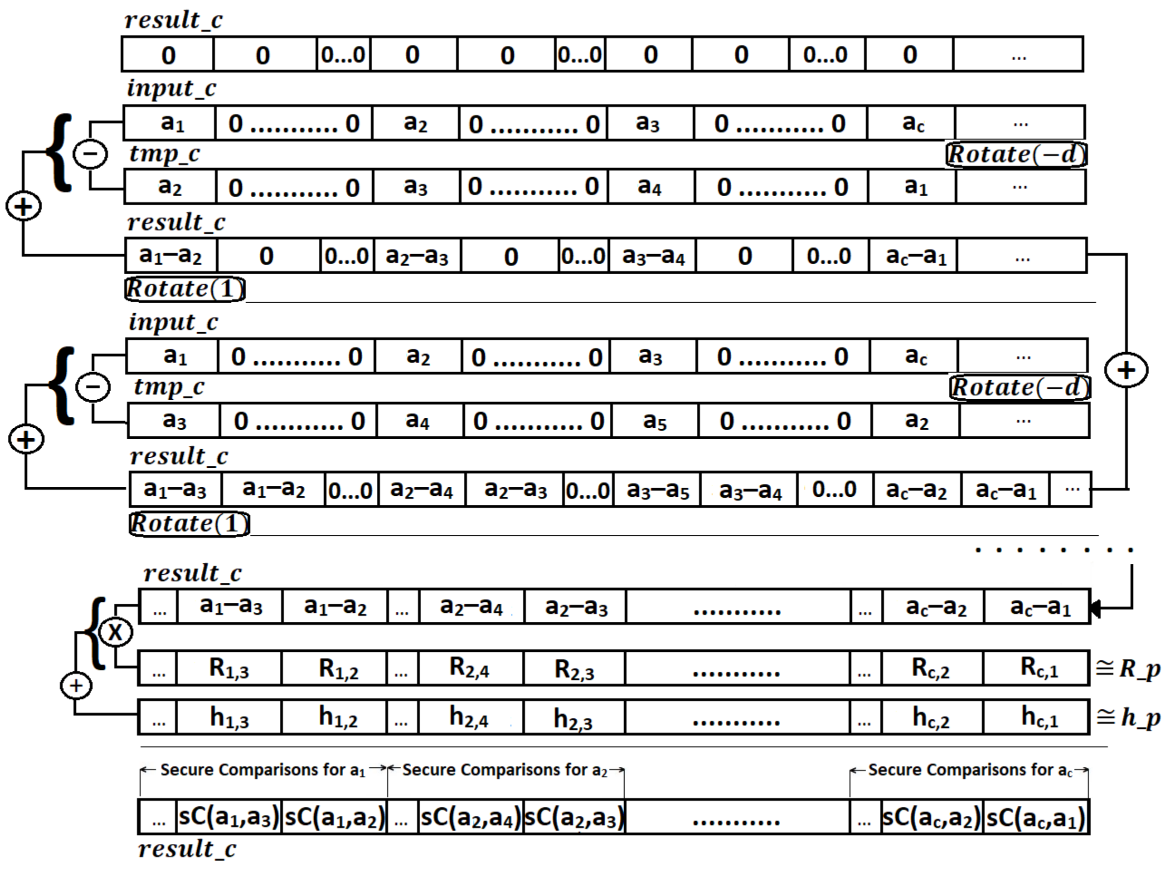

result_c
input_c
tmp_c
Rotate(−d)
Rotate(1)
≅ R_p
≅ h_p
Secure Comparisons for a1
Secure Comparisons for a2
Secure Comparisons for ac

Fig.6.6. Illustration of *SCADS* in SIMD fashion.

6.3.6. Secure sorting

It is given in Algorithm 6.6. Similarly, to the input in Algorithm 6.5, here also Party A has a ciphertext with c data slots, which, starting from the first slot (slot with index 0) are apart of each other for d slots. Party A performs SCADS on the input and send it to party B which has the decryption key. Upon decrypting, Party B learns the sorted order of the input A, but not their values.

```
ALGORITHM 6.6: secSorting (Secure Sorting)
INPUT: $input\_c, c, d$
$c$: the number of data slots (slots that carry integer data)
$d$: the number of slots between two neighboring data slots
OUTPUT: $sortedIndexes$ ($maxIndex, topKIndexes$)
$sortedIndexes\_v$: the indexes of the input ciphertext if it was sorted according to the data values
Party A:
1  $SCADS\_c = SCADS(input_c, c, d)$
2  send $SCADS\_c$ to Party B
Party B:
3  $SCADS\_v = Decrypt\_Decode(SCADS\_c)$
4  $sortedIndexes\_v = sortedIndexesAccodingToComparisonsOfAllData(SCADS\_v)$
```

6.3.7. Secure argmax and secure top-K

In Algorithm 6.3.7 we present a secure two party protocol (2PC) that finds the top k indexes of c encrypted integers. Party A has the encrypted integer array encoded and constructed in a similar fashion as it was the case with the input ciphertexts at the *SCADS*, *SRCPer* and *secSorting* algorithms. Party B has the secret key. At the end of the protocol Party B finds the indexes of the data slot that have the top k integers of the input ciphertext and nothing else (neither its value, nor the sorted order of the integer array or anything else). Party A learns nothing. When k is 1 it is secure argmax.

```
ALGORITHM 6.7: secArgmax and secTopK
INPUT: $\boldsymbol{input_c, c, d, k}$
$\boldsymbol{c}$: the number of data slots (slots that carry integer data)
$\boldsymbol{d}$: the number of slots between two neighboring data slots
$\boldsymbol{k}$: top k integers we are interested in (the ones with the top k highest values)
OUTPUT: $\boldsymbol{maxIndex\ and\ topKIndexes}$
$\boldsymbol{maxIndex\_v}$: the index of the original input ciphertext with the greatest value
$\boldsymbol{topKIndexes\_v}$: Indexes of the input ciphertext with the top-K values
Party A:
1 $permutedInput\_c = SRCPer(input\_c, k, R_1, R_2, c, d)$
2 $perSCADS\_c = SCADS(permutedInput\_c, c, d)$
3 send $perSCADS\_c$ to Party B
Party B:
4 $perSCADS\_v = Decrypt\_Decode(perSCADS\_c)$
5 $rndMaxIndex = findMaxIndex(perSCADS\_v)$ //finds the index for which the comparis. are all positive
  //$rndTopKIndexes\_v = findTopKIndexes(perSCADS\_v)$
  //top k indexes for which the comparis. are all positive
6 $rndMaxIndex\_v = constrVector(rndMaxIndex)$
  //all elements of the vector are 0s, $rndMaxIndex$ is 1
  //$rndTopKIndexes\_v = constrVector(rndTopKIndexes\_v)$
  //all elements of the vector are 0s, $rndTopKIndexes\_v$ are 1
7 $rndMaxIndex\_c = EncEncr(rndMaxIndex\_v)$
  //$rndTopKIndexes\_c = EncEncr(rndTopKIndexes\_v)$
  send $rndMaxIndex\_c$ to Party A // send $rndTopKIndexes\_c$ to Party A
Party A
8 $result\_c = invSRCPer(rndMaxIndex\_c, k, R_1, R_2, c, n_s)$
9 send $[result]$ to Party B
Party B:
10 $maxIndex\_v = Decrypt\_Decode(result\_c)$ // $topKIndexes\_v = Decrypt\_Decode(result\_c)$
```

The secTopK (secArgmax) Algorithm (Algorithm 6.7) goes as follows: Party A randomly permutes, afterwards compares all the data slots of the input ciphertexts and sends the result to A (lines 1-3). Party B decrypts it, finds the permuted index of the top k integers, constructs a ciphertext where everything is zero, except the permuted index of the top k integers, which are 1 (ones) and sends this to A (lines 4-7). Party A does the inverse permutation of this ciphertext and sends it back to B (lines 8-9). Finally, party B decrypts it and finds the real index of the top k (maximum) integer. At the end of secTopK Party B learns only the indexes with the top k values, but not their order or actual values, which makes it have strong security and privacy properties. secTopK (secArgmax) is a one-round algorithm, which makes it non-interactive and efficient in terms of communication cost. We should note that we gave the secTopK (secArgmax) algorithm only for comparison purposes done in Chapter 6.9, while in our privacy preserving classification algorithms we use a similar sequence (logical order) of commands, but scattered among three entities (TEAS, E2DS and the EC). We should also note that we can easily come up with different flavors of our secTopK (secArgmax) protocol in terms of which party gets to know the final outcome (index) of the protocol, just as it is the case with the secure comparison and secure argmax protocols in [53].

6.4. Secure building blocks for linear algebra

In [89-90] they pad their input with dummy slots with values zero so their inputs are a power of two. Due to utilizing the corresponding *secSum* (Algorithm 6.1) and *CRep* (Algorithm 6.2) which don't assume that their inputs are always powers of two, in our secure linear algebra algorithms we don't have this assumption. Two ways of encoding an integer matrix into an integer vector are given in Fig.6.7. Row-wise encoded matrices are denoted by having an R as a superscript in their name, i.e. mat^R, while column-wise encoding ones by having a C, i.e. mat^C (Fig.6.7). In this manner in a single plaintext or ciphertext of polynomial degree N we can pack $q = \frac{N}{d_1 \cdot d_2}$ matrices, where d_1 and d_2 are the dimension (number of rows and columns, respectively) of the matrix. Similarly, we can pack $q = \frac{N}{d}$ integer vectors of dimension d.

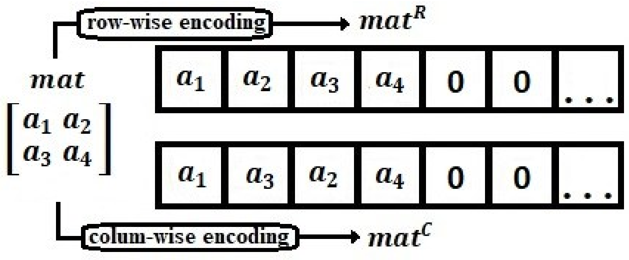

row-wise encoding
mat^R
mat
$\begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix}$
a_1 a_2 a_3 a_4 0 0 ...
a_1 a_3 a_2 a_4 0 0 ...
colum-wise encoding
mat^C

Fig.6.7. Row and column-wise encoding of integer matrices into vectors

6.4.1. Secure Dimension replication

As an input takes an encoded&encrypted matrix with dimensions d_1 and d_2 and replicates each of the input's first dimension's elements (rows) for r times in sequential order (Fig.6.8). It is given in Algorithm 6.8. Initially in line 1 a mask for all of the d_1 dimension is constructed, such that the mask of the i-th dimension has all of the slots set to zero except slots from $i \cdot d_1$-th till $(i+1) \cdot d_1 - 1$ – th slots which are set to one. The bulk of the algorithm is done in d_1 iterations such that for the i-th iteration the input matrix ciphertext is multiplied by the corresponding mask and the resulting ciphertext is rotated for $i \cdot d_2 \cdot (r-1)$ slots to the right (lines 2-5). The resulting ciphertexts of all of the iterations are added into one ciphertext (line 6) which is then replicated by r times calling Algorithm 2 (line 7) to get the desired output. Fig. 6.8 illustrates *secDRep* for $d_1 = 2, d_2 = 2, r = 4$.

ALOGRITHM 6.8: ***secDRep* - secure Dimension Replication**

INPUT: $\boldsymbol{mat\_c, d_1, d_2, r}$
$\boldsymbol{mat\_c}$: an encoded&encrypted matrix
$\boldsymbol{d_1, d_2}$: first and second dimension of mat_c
$\boldsymbol{r}$: replication rate

OUTPUT: $\boldsymbol{matDRep\_c}$
$\boldsymbol{matDRep\_c}$: $input\_c$'s d_1 dimension replicated for r times

```
1 $masksVec\_p[] = secDRepMasks(d_1, d_2)$
2 $tmp\_c[0] = mat\_c \times masksVec\_p[0]$
3 for $i = 1$ to $d_1$
4   $tmp\_c[i] = mat\_c \times masksVec\_p[i]$
5   $tmp\_c[i] = Rotate\ (tmp\_c[i],\ i \times d_2 \times (r-1))$
6 $matDRep\_\_c = AddMany(tmp\_c[])$
7 $matDRep\_\_c = CRep(matDRep\_\_c\ , d_2, r)$
```

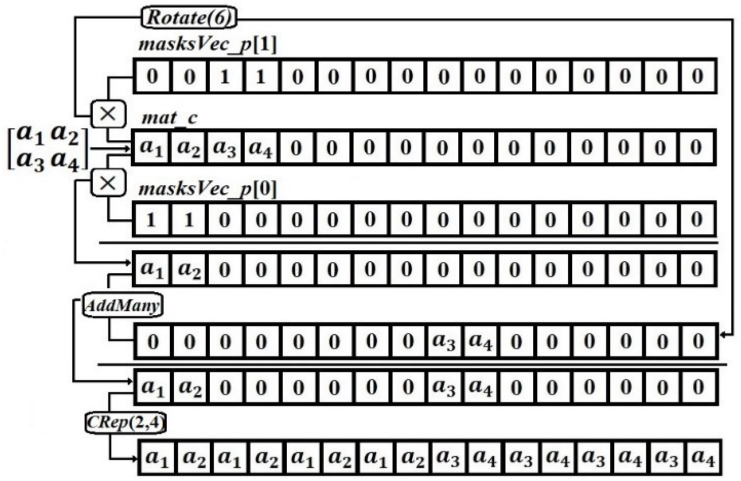

Rotate(6)
masksVec_p[1]
0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0
mat_c
a1 a2 a3 a4 0 0 0 0 0 0 0 0 0 0 0 0
[a1 a2; a3 a4]
masksVec_p[0]
1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
a1 a2 0 0 0 0 0 0 0 0 0 0 0 0 0 0
AddMany
0 0 0 0 0 0 0 0 a3 a4 0 0 0 0 0 0
a1 a2 0 0 0 0 0 0 a3 a4 0 0 0 0 0 0
CRep(2,4)
a1 a2 a1 a2 a1 a2 a1 a2 a3 a4 a3 a4 a3 a4 a3 a4

Fig. 6.8. Illustration of $secDRep(\cdot)$ for $d_1 = 2, d_2 = 2$ and $r = 4$

6.4.2. Secure dot (inner) product

Given in Algorithm 4. It takes two vectors of dimension d, multiplies them (line 1) and calls $secSum(\cdot)$ to get their final dot product (line 2).

ALOGRITHM 6.9: ***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 $result\_c = vec1\_c \times vec2\_c$
2 $result\_c = secSum(result\_c, d)$
```

6.4.3. Secure matrix-vector product

Given in Algorithm 6.10. As input it takes a row-encoded matrix ciphertext and a vector ciphertext and translates their corresponding matrix-vector multiplication into a dot product (line 2) by firstly replicating the vector for d_1 times (line 1).

ALOGRITHM 6.10: ***secMatVec* - secure Matrix Vector Product**

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

OUTPUT: $\boldsymbol{result\_c}$
$\boldsymbol{result\_c}$: the result of the mat-vec product after each d_2 slots

```
1 $vec\_c = CRep(vec\_c, d_1, d_2)$
2 $result\_c = secDotP(mat\_c, vec\_c, d_2)$
```

6.4.4. Secure matrix-matrix product

Given in Algorithm 6.

ALOGRITHM 6.11: ***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. $d_1 \times d_2$
$\boldsymbol{mat^C\_c}$: column-encoded (packed) matrix(es) with dim $d_2 \times d_3$

OUTPUT: $\boldsymbol{result\_c}$
$\boldsymbol{result\_c}$: the result of the mat-mat product after each d_2 slots

```
1 $mat^R\_c = CRep(mat^R\_c, d_1 \times d_2, d_3)$
2 $mat^C\_c = secDRep(mat_c^C, d_3, d_2, d_1)$
3 $result\_c = secDotP(mat^R\_c, mat^C\_c , d_2)$
```

As an input it takes one row-encoded and one column-encoded matrix. After replicating the row-encoded matrix (line 1) and replicating the dimensions of the column-encoded matrix (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.

6.4.5. Secure ciphertext compression

Given in Algorithm 6.12**.** After secure matrix-vector and matrix-matrix product, the resulting ciphertext has its data slots scattered (separated) from each other according to the corresponding dimension(s) of their input. As an input it takes a ciphertext that has $nrOfBlocks$ number of blocks to be compressed, each with size $blockSize$ separated from each other by d slots. 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 iteration 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.6.9a) for input parameters $nrOfBlocks = 4$, $blockSize = 1, d = 1$ and in Fig.6.9b). for input parameters $nrOfBlocks = 4$, $blockSize = 1, d = 3$.

ALOGRITHM 6.12: ***CCompress* – Ciphertext Compress**

INPUT: $\boldsymbol{input\_c, blockSize, d, nrOfBlocks}$
$\boldsymbol{input\_c}$: a sparsely encoded (packed) ciphertext
$\boldsymbol{blockSize}$: number of slots per block
$\boldsymbol{d}$: distance between two neighboring blocks in terms of slots
$\boldsymbol{nrOfBlocks}$: number of blocks to be compressed

OUTPUT: $\boldsymbol{result\_c}$
$\boldsymbol{result\_c}$: the compressed ciphertext

```
1 $emptySlots = d$
2 $result\_c = input\_c$
3 $for\ i = 0\ to\ \log_2(nrOfBlocks)$
4     $tmp\_c = Rotate\ (result\_c\ , -d)$
5     $result\_c = result\_c + tmp\_c$
6     $emptySlots = emptySlots - blockSize$
7     $d = 2 \times d$
8     $blockSize = 2 \times blockSize$
9     $if(emptySlots == 0)$
10        $result\_c = result\_c \times maskCCompress\_p[blockSize, d]$
11        $emptySlots = blockSize$
```

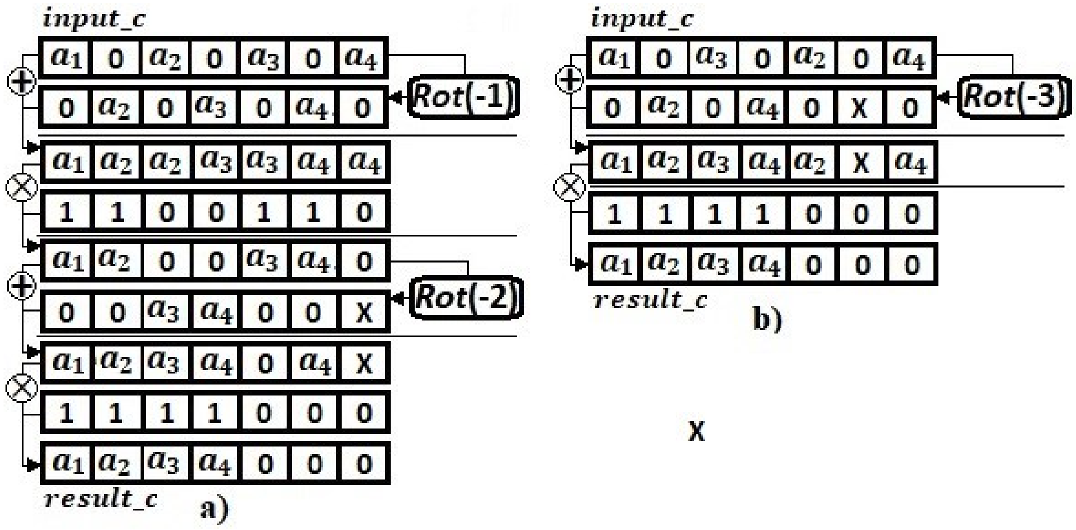


Fig.6.9 Illustration of $CCompress(\cdot)$ for input parameters a) $nrOfBlocks = 4$, $blockSize = 1, d = 1$ and b) $nrOfBlocks = 4$, $blockSize = 1, d = 4$

6.4.6. Secure matrix transpose

Given in Algorithm 6.13, it securely transposes an input matrix with dimensions $d_1 \times d_2$. Using the corresponding d_2 masks, the input ciphertext is multiplied by them and each of them has the columns of the input 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 we firstly convert the input matrix into sparsely encoded ciphertext (lines 1-4), to which we apply $CCompress(\cdot)$ to get the transposed matrix result. The illustration of $secMatT(\cdot)$ is given in Fig.6.10 for $d_1 = 2$ and $d_2 = 2$.

ALOGRITHM 6.13: ***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 $for\ i = 0\ to\ d_2$
2     $tmp\_c[i] = mat\_c \times maskMatT\_p(i, d_1, d_2)$ //starting from slot I you put d1 1s in distance of d2 from eo
3     $tmp\_c[i] = Rotate(tmp\_c[i], i \times (d_1 \times d_2 - 1))$
4 $tmpRes\_c = AddMany(tmp\_c[])$
5 $matT\_c = CCompress(result\_c, 1, d_2 - 1, d_1 \times d_2)$ //initially block size is 1, and have $d_1 \times d_2$ in dis d2 from eo
```

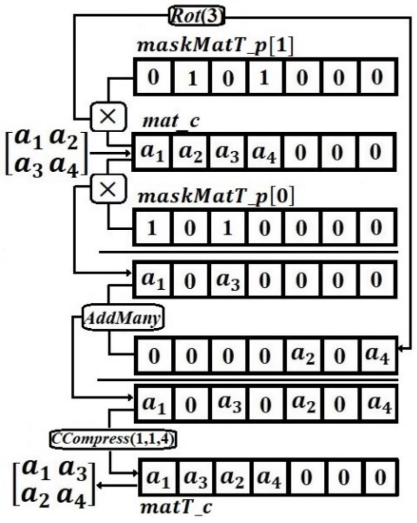


Fig.6.10 Illustration of $secMatT(\cdot)$ for $d_1 = 2$ and $d_2 = 2$.

6.4.7. Secure matrix transpose and dimension replication

Given in Algorithm 6.14. For efficiency purposes, instead of transposing the matrix using $secMatT(\cdot)$ and then replicating its dimensions using $secDRep(\cdot)$, $secMatTDrep(\cdot)$ on the fly does them both, thus replicates the dimensions of the transposed input matrix. It is illustrated in Fig. 6.11.

ALOGRITHM 6.14: ***secMatTDRep*** **–secMatT and Dimension Replication**

INPUT: $\boldsymbol{mat\_c, d_1, d_2, r}$
$\boldsymbol{mat\_c}$: a matrix with dimensions $d_1 \times d_2$
$\boldsymbol{r}$: replication rate

OUTPUT: $\boldsymbol{result\_c}$
$\boldsymbol{matTDRep\_c}$: mat_c transposed and its dimensions replicated by r

```
1 $for\ i = 0\ to\ d_2$
2   $tmp\_c[i] = mat\_c \times maskMatT\_p(i, d_1, d_2)$ //starting from slot I you put d1 1s in distance of d2 from eo
3   $if(r > d_2)$
4     $tmp\_c[i] = Rotate(tmp\_c[i], i \times (d_1 \times r - 1))$
5   $else$
6     $tmp\_c[i] = Rotate(tmp\_c[i], i \times (d_1 \times d_2 - 1))$
6 $tmpRes\_c = AddMany(tmp\_c[])$
7 $tmpRes\_c = CCompress(result\_c, 1, d_2 - 1, d_1)$ //initlly block size is 1, and have $d_1$ in dis d2 from eo
8 $if(r > d_2)$
9   $tmpRes\_c = tmpRes\_c \times maskMatTDRep(d_1, d_1 \times (r - 1))$
10  $matTDRep\_c = CRep(tmpRes\_c, d_1, r)$
11 $else$
12  $tmpRes\_c = tmpRes\_c \times maskMatTDRep(d_1, d_1 \times (d_2 - 1))$
13  $tmpRes\_c = CRep(tmpRes\_c, d_1, r)$
14  $matTDRep\_c = CCompress(result\_c, r \times d_1, d_1 \times (d_2 - r), d_2)$
```

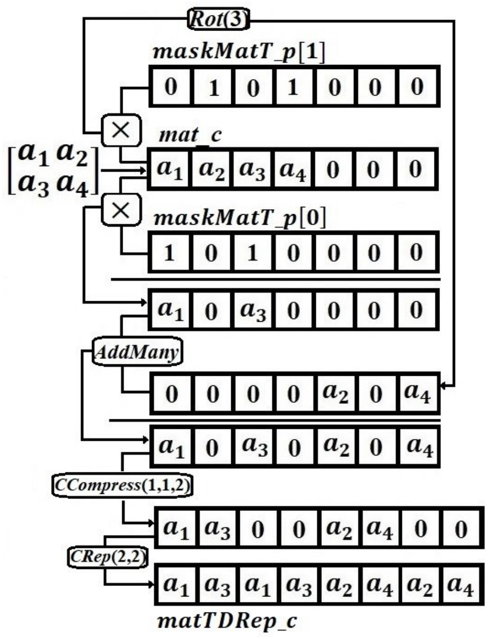

Rot(3)
maskMatT_p[1]
0 1 0 1 0 0 0
mat_c
$\begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix}$
a_1 a_2 a_3 a_4 0 0 0
maskMatT_p[0]
1 0 1 0 0 0 0
a_1 0 a_3 0 0 0 0
AddMany
0 0 0 0 a_2 0 a_4
a_1 0 a_3 0 a_2 0 a_4
CCompress(1,1,2)
a_1 a_3 0 0 a_2 a_4 0 0
CRep(2,2)
a_1 a_3 a_1 a_3 a_2 a_4 a_2 a_4
matTDRep_c

Fig. 6.11. Illustration of $secMatTDRep(\cdot)$ for $d_1 = 2, d_2 = 2$ and $r = 2$

6.4.8. Secure cascading matrix vector product

Given in Algorithm 6.15. After *secMatVec*, 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 multiplication we compress the data slots using the *CCompress* algorithm (line 2).

ALOGRITHM 6.15: ***secMatVecCas* - secure Matrix Vector Cascading Product**

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

OUTPUT: $\boldsymbol{result\_c}$
$\boldsymbol{result\_c}$: the compressed result of the mat-vec product

```
1 $tmp\_c = secMatVec(mat^R\_c, vec\_c, d_1, d_2)$
2 $result\_c = CCompress(tmp\_c, 1, d_2 - 1, d_1)$
```

6.4.9. Secure cascading matrix-matrix product

Given in Algorithm 6.16. After *secMatMat*, 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 multiplication we compress the data slots using the *CCompress* algorithm (line 2).

ALOGRITHM 6.16: ***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 dim. $d_1 \times d_2$
$mat^C\_c$: column-encoded (packed) matrix(es) with dim $d_2 \times d_3$
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 $result\_c = CCompress(tmp\_c, 1, d_2 - 1, d_1 \times d_3)$
```

6.4.10. Secure matrix-matrix product – version 2

Given in Algorithm 6.17. While Algorithm 6.4 takes one row-encoded and one column-encoded matrix ciphertext, this one takes both of them as row encoded matrix ciphertexts. It firstly replicates the dimensions of the first matrix using *CRep* (line 1), on the fly transposes and replicated the second matrix and finally uses *secDotP* to find their product with a single SIMD multiplication.

ALOGRITHM 6.17: ***secMatMat_v2*** **- secure Matrix Matrix – version 2 Product**

```
INPUT: $mat^R\_c, mat^R\_c, d_1, d_2, d_3$
$mat1^R\_c$: row-encoded (packed) matrix(es) with dim. $d_1 \times d_2$
$mat2^R\_c$: row-encoded (packed) matrix(es) with dim $d_2 \times d_3$
OUTPUT: $result\_c$
$result\_c$: the result of the mat-mat product after each $d_2$ slots
1 $mat1^R\_c = CRep(mat1^R\_c, d_1 \times d_2, d_3)$
2 $mat2^R\_c = secMatTDRep(mat2^R\_c, d_2, d_3, d_1)$
3 $result\_c = secDotP(mat1^R\_c, mat2^R\_c, d_2)$
```

6.4.11. Secure cascading matrix-matrix product – version 2

Given in Algorithm 6.18.

ALOGRITHM 6.18: ***secMatMatCas_v2*** **- secure Matrix Matrix Cascading– version 2 Product**

```
INPUT: $mat^R\_c, mat^R\_c, d_1, d_2, d_3$
$mat1^R\_c$: row-encoded (packed) matrix(es) with dim. $d_1 \times d_2$
$mat2^R\_c$: row-encoded (packed) matrix(es) with dim $d_2 \times d_3$
OUTPUT: $result\_c$
$result\_c$: the result of the mat-mat product after each $d_2$ slots
1 $tmp\_c = secMatMat\_v2(mat1^R\_c, mat2^R\_c, d_1, d_2, d_3)$
2 $result\_c = CCompress(tmp\_c, 1, d_1 \times d_2 - 1, d_1 \times d_3)$ //from column-wise to row wise direct conversion
```

Just as it was the case with *secMatMat*, after *secMatMat_v2* the resulting ciphertext has its data sparsely encoded. We use *CCompress* again to compress these data so the resulting ciphertexts is without any gaps (empty slots) between the data, thus making it ready for another round of matrix operations.

6.4.12. Secure ciphertext compression – version 2

Given in Algorithm 6.19**.** This is a variant of *CCompress* where at the input parameters the $blocksize$ (number of slots per block) is greater than d (the distance between the neighboring blocks), which makes the algorithm slightly different than *CCompress.*

ALOGRITHM 6.19: ***CCompress2*** **–** **<u>C</u>iphertext <u>Compress</u> – ver. <u>2</u>**

INPUT: $\boldsymbol{input\_c, blockSize, d, nrOfBlocks}$
$\boldsymbol{input\_c}$: a sparsely encoded (packed) ciphertext
$\boldsymbol{blockSize}$: number of slots per block s.t. $blockSize > d$
$\boldsymbol{d}$: distance between two neighboring blocks in terms of slots
$\boldsymbol{nrOfBlocks}$: number of blocks to be compressed

OUTPUT: $\boldsymbol{result\_c}$
$\boldsymbol{result\_c}$*:* the compressed ciphertext

```
1 $result\_c = input\_c$
2 $for\ i = 1\ to\ \log_2(nrOfBlocks)$
3     $tmp\_c[1] = result\_c \times maskCC2\_p(0, blockSize, d)$
4     $tmp\_c[2] = result\_c \times MaskCC2\_p(1, blockSize, d)$
5     $tmp\_c[2] = Rotate(tmp\_c[2], -d)$
6     $result\_c = tmp\_c[1] + tmp\_c[2]$
7     $d = 2 \times d$
8     $blockSize = 2 \times blockSize$
```

6.4.13. Secure Frobenius Product

Given in algorithm 6.20**.** Finds the Frobenius (dot, inner) product of two matrices with the same dimensions. Frobenius product is used in Convolutional Neural Networks (CNN) [91].

ALOGRITHM 6.20: ***secFrobP*** **- <u>sec</u>ure <u>Frob</u>enius <u>P</u>roduct**

INPUT: $\boldsymbol{mat1\_c, mat2\_c, d_1, d_2,}$
$\boldsymbol{mat1\_c}$: encoded (packed) matrix(es) with dim. $d_1 \times d_2$
$\boldsymbol{mat2c}$: encoded (packed) matrix(es) with dim. $d_1 \times d_2$

OUTPUT: $\boldsymbol{result\_c}$
$\boldsymbol{result\_c}$*:* the Frobenius inner product between $mat1\_c$ and $mat2\_c$

```
1 $result\_c = secDotP(mat1\_c, mat2\_c, d_1 \times d_2)$
```

6.4.14. Secure ciphertext packing

Packs several ciphertexts into one. The pseudocode is given in Algorithm 6.20A, the illustration in Fig.8. As an input it takes a vector of p ciphertexts ($inputVector[]$) and the number of (useful) data slots (n_L) which, starting from the first slot, each ciphertext of the input vector has,. Instead of a vector of ciphertexts, as an input it can also take one ciphertext (the $[input]$), which is replicated for p times. Lines 1, 2 and 6 can be skipped if all of the elements (ciphertexts) of the input vector at their first n_L slots have their data, and the reaming slots are filled up with zeros. It is assumed that all the input ciphertexts (elements) of the $inputVector\_c[]$ have $p \cdot n_L$ slots.

ALGORITHM 6.20A: ***CPack***

INPUT: $\boldsymbol{inputVector\_c[], n_L, p}$

$\boldsymbol{inputVector\_c[]}$**:** a vector of p ciphertexts, each ciphertext has $p \cdot n_L$ slots, at each of them only the first n_L slots have data that we are interested in

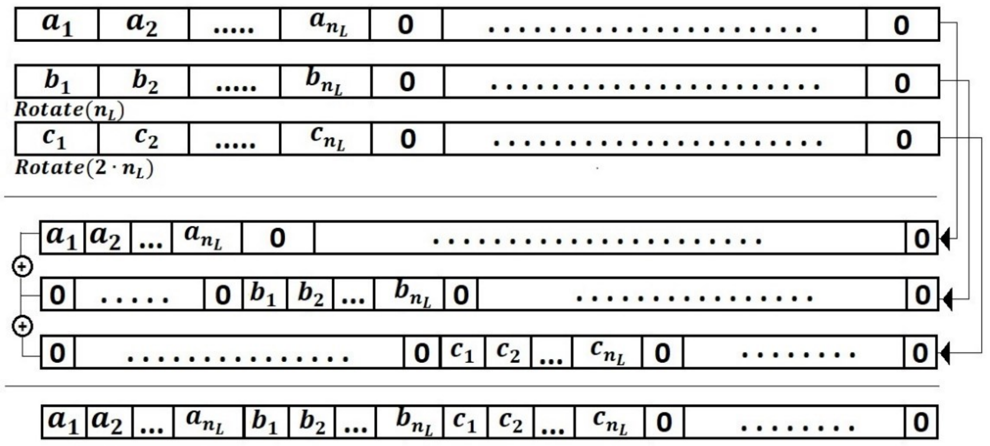
$\boldsymbol{n_L}$**:** starting from the first slot (slot 0), is the number of consecutive data slots that each input ciphertext has

$\boldsymbol{p}$: the number of elements (ciphertexts) of $inputVector[]$

OUTPUT: $\boldsymbol{result\_c}$

$\boldsymbol{result\_c}$**:** the packed ciphertext that contains all of the $inputVector[]$ ($[input]$). It has at least $p \cdot n_L$ slots

```
1 $mask\_v = \{1,1,\dots,1,0,\dots,0\}$     //first $n_L$ slots are ones (1s), the rest are zeros. In total it has $p \cdot n_L$ slots
2 $mask\_p = Encode(mask\_v)$
  $result\_c = inputVector\_c[1]$     //only for the single $[input]$ case
3 for $i = 2$ to $p$ do
4     $inputVector\_c[i] = inputVector\_c[i] \times mask\_p$
5     $inputVector\_c[i] = Rotate(inputVector\_c[i], i \times n_L)$
6 $result\_c = addMany(inputVector\_c[])$     //add the input ciphertexts (a total of log$p$ additions)
7 return $result\_c$     /
```



a_1 a_2 ..... a_{n_L} 0 0
b_1 b_2 ..... b_{n_L} 0 0
$Rotate(n_L)$
c_1 c_2 ..... c_{n_L} 0 0
$Rotate(2 \cdot n_L)$
a_1 a_2 ... a_{n_L} 0 0
0 ..... 0 b_1 b_2 ... b_{n_L} 0 0
0 .... 0 c_1 c_2 ... c_{n_L} 0 0
⊕
⊕
a_1 a_2 ... a_{n_L} b_1 b_2 ... b_{n_L} c_1 c_2 ... c_{n_L} 0 0

Figure 6.12. Illustration of the ciphertext packing (*CPack*) algorithm

6.5. Secure classifications based on NB, kNN, DT and RF

In this Chapter we provide the NB based classification for non-textual datasets, and also for kNN, decision trees as well as random forests. While doing so we have in mind the security goals for secure classification set in chapter 1. Depending on where the bulk of the operations are done, all of them can be written as server or client (user) centric. We provide both of them only for NB case.

6.5.1. Secure classifications for non-textual queries based on NB

Uses the background information given in Chapter 2.1.1. Depending on where the bulk (most) of the processing is done, in this Chapter we give (provide) two types of privacy preserving classification schemes: the server-centric and the client (user)-centric. In this sense, for each query we offer to the system the flexibility of choosing one of the schemes depending on the current workload at the server or the client (user) side. In both of them TACS holds the encrypted trained model (denoted as $\boldsymbol{TM\_c}$) and the user has an unclassified query $\boldsymbol{X}$. Both of the schemes satisfy the security requirements mentioned in Chapter 1 and they both deal with passive participants in the semi-honest model. Furthermore, the client-centric scheme also deals with a user that can apply the active "substitution-then-comparison" (STC) attack proposed in [10]. In Chapter 6.8 we deal with an active malicious user during the server-centric classification that can arbitrarily deviate from the protocol. Although the classification schemes have three participants, both of them are easily convertible to secure two party protocols (2PC) where the server has the trained model and the user an unclassified query.

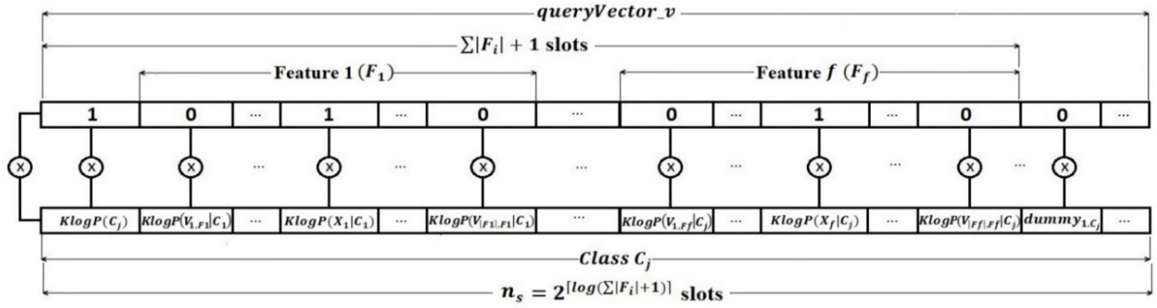


Fig.6.13. (14). SIMD per class view of the query vector with zeros and ones in corresponding places according to the query feature vector X

ALGORITHM 6.21: ***PPClassServCen*** **(Privacy Preserving Classification - server centric)**

INPUT: $\boldsymbol{X = \{X_1, X_2 \dots X_f\}, F, C, TM\_c}$

$\boldsymbol{X = \{X_1, X_2 \dots X_f\}}$: unclassified query feature vector owned by the User, s.t. $X_i \in F_i$ for $1 \le i \le f$

$\boldsymbol{F}$: $F = \{F_1, F_2, \dots, F_f\}$, where $F_i = \{V_{1,Fi}, V_{2,Fi}, \dots, V_{|Fi|,Fi}\}$. F_i, st. $1 \le i \le f$ (as explained in Chapter 2.1.1)

$\boldsymbol{C}$: The set of classes $C = \{C_1, C_2, \dots, C_c\}$ (as explained in Se Chapter 2.1.1)

$\boldsymbol{TM\_c}$ an already SIMD encrypted trained Naïve Bayes model stored at TEAS

OUTPUT: $\boldsymbol{C_{TM}(X)}$

$\boldsymbol{C_{TM}(X)}$:the classification of the query vector X according to the NB trained model TM_c

```
Phase I – EC:
1  $queryVector\_v.insert(1, 0)$   //insert 1 at the first slot for the class probability in the empty $queryVector$
2  for $i = 1$ to $f$                                                     //for each feature
3    for $m = 1$ to $|F_i|$                      //for each value of the current feature (feature $F_i$)
4      if $X_i == V_{m,Fi}$ then               //if $X_i$ is equal to the current $V_{m,Fi}$ feature value of feature $F_i$
5        $queryVector\_v.insert(1)$                 //then insert (put, push) one (1) to the $queryVector$
6      else                                                             //otherwise
7        $queryVector\_v.insert(0)$       //insert zero. At the end, the $queryVector$ should look like Fig.14
8  for $j = 2$ to $c$    //afterwards replicate the $queryVector$ for $c - 1$ times, where $c$ is the number of classes
9    $queryVector\_v.insert(queryVector\_v)$     //query vector should look like the upper vector of Fig.15
10 $queryVector\_c = EncEncr(queryVector\_v)$    //$n_L = c \cdot n_s$ slots, where $n_s = 2^{\lceil \log(\Sigma|F_i|+1) \rceil}$
11 send $k, R_1, R_2, queryVector\_c$ to TEAS            //$k, R_1, R_2$ are random, needed for SRCPer (line 17)
Phase II - TEAS:
12 receive $k, R_1, R_2, queryVector\_c$                 //receives the encrypted query sent by the user
13 $//results\_c[] = recieveMultipleEncryptedQueries(p)$
     //receives $p$ encrypted query vectors from different users
14 $//queryVector\_c = CPack(results\_c[], n_L, p)$ //packs them into a ciphertext using CPack, $n_L = c \cdot n_s$
15 $result\_c = queryVector\_c \times [TM\_c]$                    //as shown in Fig.14-15
16 $result\_c = secSum(result\_c, n_s)$ //finds the class prob. for each class $C_j, 1 \le j \le c, n_s = 2^{\lceil \log(\Sigma|F_i|+1) \rceil}$
17 $result\_c = SRCPer(result\_c, k, R_1, R_2, c, n_s)$    //permut. of the data slots for the class probabilities
18 $result\_c = SCADS(result\_c, c, n_s)$ //compare all of the posterior class probab., $c$ is the nr. of classes
19 send $[result]$ to EDS
Phase III – E2DS:
20 receive $result\_c$
21 $result\_v = DecrDec(result\_c)$
22 $rndMaxInd = findMaxIndex(result\_v)$ //the index for which all comparisons are positive (Fig.7)
23 send $rndMaxInd$ to User        //sends this $rndMaxIndex$ in clear (as a pure rand. integer) to the EC
Phase IV – EC:
24 receive $rndMaxInd$ //the invSRCPer in line 25 is done by taking an integer index as an input (Section V)
25 $C_{TM}(X) = invSRCPer(rndMaxInd, k, R_1, R_2, c, n_s)$//de-rand. to find the orig. index of $rndMaxIndex$
```

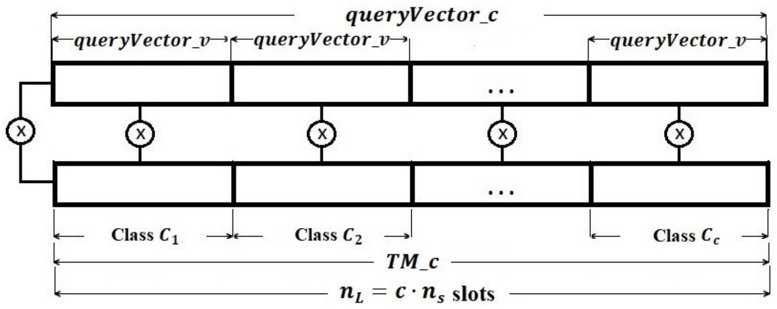


Fig.6.14. Multiplying $queryVector\_c$ with TM_c

The pseudocode for the server-centric NB classification of non-textual datasets case is given in Algorithm 6.21. As an input it takes the trained model TM_c kept at TEAS and the EC's feature vector $X = \{X_1, X_2, \dots, X_f\}$, where $X_i \in F_i$, for $1 \leq \text{i} \leq \text{f}$. In Phase I EC(s) construct(s) the query vector to look like the upper vector in Fig.6.13 (lines 1-7), i.e. for each of the ordered feature set F_i s.t. $1 \leq i \leq f$ and $1 \leq m \leq |F_i|$, if $X_i == V_{m,Fi}$ we put 1 (one) to that corresponding slot, otherwise everything else is zero. In the beginning of the query we have the slot associated with class probabilities $P(C_j)$ for $1 \leq j \leq c$, and its value is always 1 (one) (Fig.6.13). The user then replicates the same vector for $c - 1$ times (lines 8-9), where c is the number of classes, and afterwards encodes and encrypts this vector to get the final $[encryptedQueryVector]$ (Fig.6.14) which is send to TEAS (lines 10-11) together with the random $k = \{k_1, \dots k_m\}, R_1$ and R_2 needed for the *SRCPer* algorithm. In Phase II TEAS receives the $queryVector\_c$ (line 12). In order to increase the throughput, TEAS might receive multiple (say p) encrypted query vectors, and if so he first runs the *CPack* (Algorithm 6.2, lines 13-14) to pack them into a single ciphertext. Here the size of a single encrypted query vector is $n_L = c \cdot n_s$ (a single class has $n_s = 2^{\lceil \log(\sum |F_i|+1) \rceil}$ slots, thus a single query has $n_L = c \cdot n_s$ slots). We should note that if there are p multiple queries involved, the trained model should be replicated p times (shown during the training phase – line 10). Then TEAS multiplies the (packed) $queryVector\_c$ with the (packed) trained model TM_c (line 15, Fig.6.13-6.14), and afterwards runs the *secSum* (Algorithm 6.1) (line 16) to find the posterior class probabilities at the beginning of each of the $n_s - th$ slots, starting from slot zero. After randomly permuting the data slots containing the posterior probabilities (line 17) and securely comparing all of them with each other (line 18), TEAS sends the final result to E2DS (line 19).

It Phase III E2DS receives, decrypts and decodes the final results to get the randomized index of the class with highest posterior probability before sending it to the User(s) in plain (lines 20-22). Finally, EC(s) in Phase IV remove the randomization by running the *invSRCPer* (Chapter 6.3.3) to get the final classification(s) (lines 24-26).

6.5.2. Secure classifications based on kNN

The 2PC is given in Algorithm 6.22. No prior training is done since kNN doesn't need it. In our scenario, an EDO owns a dataset that he wishes to be used for secure classifications and (an) EC(s) user has(have) queries that he (they) wish to classify having in mind the security requirements during the classification stage set in Chapter 1. The EDO's dataset has NT transactions (records), each with f attributes and the corresponding class for that record. Thus a record r_i looks like $Y^{r_i} = \left\{Y_1^{r_i}, \dots, Y_f^{r_i}, C^{r_i}\right\}$, where $1 \leq i \leq NT$ and C^{r_i} is the r_i's class. The EDO randomly permutes his NT transaction (records) according to a random permutation π, and encodes the feature values of each record in a sequential order in a plaintext(s) denoted as $perDS\_p[]$, with a corresponding plaintext where he encode his classes, denoted as $classes\_p$. In a single plaintext of polynomial N we can encode $q = N/f$ features, thus we need $nrPlainTexts = \lceil NT/q \rceil$ plaintexts at the EDO to encode only the feature values of each record in the dataset. To this ends, if a single plaintext is not enough for $perDS\_p[]$, then at the first plaintext of $perDS\_p[]$ EDO encodes the first q permuted records, at the second plaintext of $perDS\_p[]$ encodes the second q queries etc., as it is illustrated in Fig. 6.15. For the $classes\_p$ plaintext at the first f slots we put the classes for the permuted records which are in distance of q to each other, thus the first f spot are for classes $C^{\pi(r_1)}, C^{\pi(r_{q+1})}, \dots, C^{\pi(r_{f-1})}$, the second f spots are for $C^{\pi(r_2)}, C^{\pi(r_{q+1})}, \dots, C^{\pi(r_{f+1})}$, etc. (Fig.6.15). The EC replicates his query for q times, encodes and encrypts to get X_c and sends it to the EDO (lines 1-2). If all of the dataset can be put in a single $perDS\_p[]$ plaintext, EDO then subtracts the $perDS\_p$ with the X_c, squares the result and finds the sum of the corresponding f slots in SIMD fashion according to one of the distances proposed in Chapter 2, which in this case is the Euclidian distance (lines 4-6). If the dataset doesn't fit in a single $perDS\_p[]$ plaintext, then the same process is repeated for all of the plaintexts, in the process utilizing multiple cores of the processor (lines 8-13). Making the necessary rotations and additions $perDS\_p[]$ plaintexts, EDO makes the calculated packed distances of each dataset record with the query into $result\_c$ to correspond to the record's classes in $classes\_p$ (lines 4-12). In order to guard against any eventual data leakage, in SIMD fashion EDO multiples each distance with the same random R proceed by adding another random h (line 13) and sends it to EC (line 14). EC decrypts the randomized result, constructs a vector by putting ones at the slots of the k distances with the minim value, encodes and encrypts it to get $queryC_{kNN}(X\_v)\_c$ and sends it to EDO (lines 15-18). EDO multiplies

$queryC_{kNN}(X\_v)\_c$ with the $classes\_p$ and send the result back to EC (lines 19-20). Finally, EC decrypt this result which contains k classes corresponding to the top-K classifier.

ALGORITHM 6.22: ***secKNN*** **(<u>sec</u>ure <u>K</u> <u>N</u>earest <u>N</u>eighbors)**

INPUT: $\boldsymbol{perDS\_p}[], \boldsymbol{classes\_p}, \boldsymbol{f}, \boldsymbol{d}, \boldsymbol{k}, \boldsymbol{X\_v}$

$\boldsymbol{perDS\_p}[]$: permuted according to π and encoded records of the dataset DS residing at the server, who is also the owner of it

$\boldsymbol{classes\_p}$: the vector of classes of the corresponding permuted records $classes\_v = \left\{\left(C^{\pi(r_i)}\right)_{i=1}^{NT}\right\}$, $classes\_p = Encode(classes\_v)$

$\boldsymbol{f}$: the dimension of the dataset (number of features, i.e. number of slots needed for each record)

$\boldsymbol{k}$: the number of the closest neighbors by which the classification is done

$\boldsymbol{X\_v}$: the client's query vector of dimension f replicated by $q = N/f$ times, where q is also the number of records in a single plaintext

OUTPUT: $\boldsymbol{maxIndex\ and\ topKIndexes}$

$\boldsymbol{C_{kNN}(X)}$: the final classification of q_v according to kNN and dataset DS:

```
EC:
1 $X\_c = EncEncr(X\_v)$
2 send $X\_c$ to EDO
EDO:
3 $nrPlaintextsForDS = \lceil (NT \times f)/N \rceil$; $R\_p = Encode\{R, \ldots, R\}$; $h\_p = Encode\{h, \ldots, h\}$
4 $if(nrPlaintextsForDS == 1)$
5     $result\_c = square(perDS\_p - X\_c)$
6     $result\_c = secSum(C_{kNN}(X\_v)\_c, d)$
7 else
8     for $i = 0$ to $nrPlaintextsForDS - 1$ //done in parallel among several processor cores
9         $tmp\_c[i] = square(perDS\_p[i] - X\_c)$
10        $tmp\_c[i] = secSum(tmp\_c[i], f)$
11        $tmp\_c[i] = Rotate(tmp\_c[i], i)$
12    $result\_c = AddMany(tmp\_c[])$
13 $result\_c = result\_c \times R\_p + h\_p$
14 send $result\_c$ to EC
EC:
15 $result\_v = DecrDec(result\_c)$
16 $queryC_{kNN}(X\_v)\_v = getTopKValues(result\_v)$
   //ones in the indexes containing the smallest $k$ values, 0's elsewhere
17 $queryC_{kNN}(X\_v)\_c = EncEncr(queryC_{kNN}(X\_v)\_v)$
18 send $queryC_{kNN}(X\_v)\_c$ to EDO
EDO:
19 $C_{kNN}(X\_v)\_c = queryC_{kNN}(X\_v)\_c \times classes\_p$
20 send $C_{kNN}(X\_v)\_c$ to EC
EC:
21 $C_{kNN}(X\_v)\_v = DecrDecC_{kNN}(X\_v)\_c$
22 $C_{kNN}(X) = classifyAccordingToClosestKClasses(C_{kNN}(X\_v)\_v)$
```

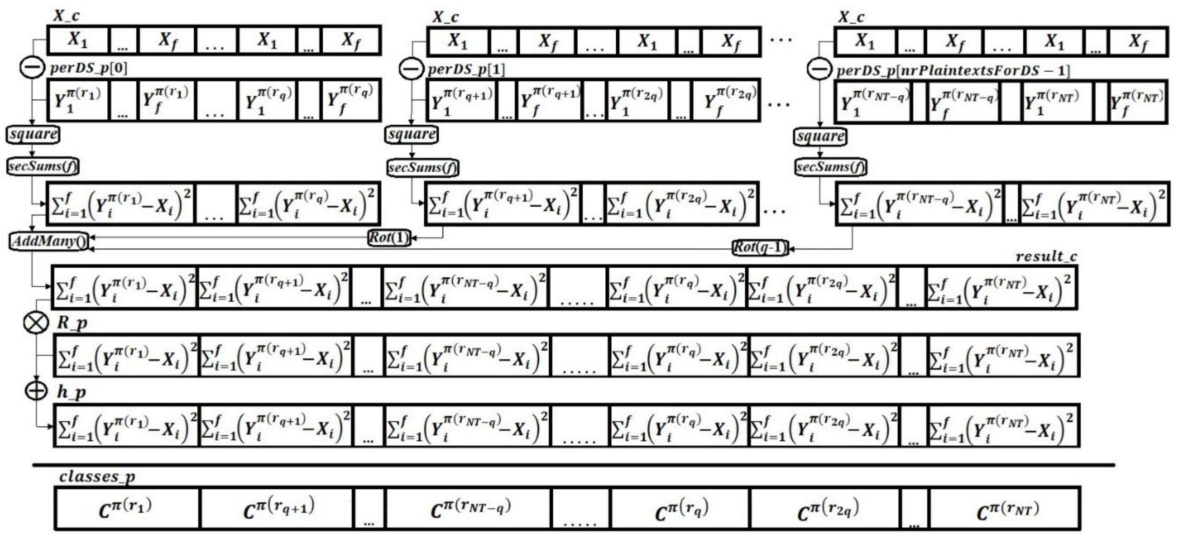


Fig. 6.15. Illustration of construction of $perDS\_p[]$ and $classes\_p$ for the *secKNN*.

6.5.3. Secure classifications based on DT and RF

Given in algorithm 6.23. If the trained model consisted of a decision tree is not binary, it can be easily converted to one [9,73, 92]. Unlike [92] which reveals the depth of the tree, in our algorithm in order to hide the depth of the binary tree 1) we can add a dummy root which on both sides has the exact replica of the original binary tree, and/or 2) we can add one or more levels at the leaves by putting a couple of dummy nodes (one on each side of every leaf), which will point to the same class that the corresponding leaf (now parent node) used to point. The obtained binary tree then can be encoded into a vector in a way that, start from the root, each node's value of each level is put (encoded) into the vector in sequential order, as it is shown in Fig. 6.16. This represents the trained model TM_p plaintext. In the $classes\_p$ plaintext in sequential order, from left to right, we put the classes to which the leaves point. Using *SRCPer* (Chapter 6.3.3) in plain, the server publishes the permuted order of the TM_p by which the clients should encode their feature values. According to this published order, EC encodes and encrypts the values of its features to get X_c and sends it to the Server (lines 1-2). The Server then homomorphically performs the *invSRCPer* over X_c using the same parameters it used while performing the *SRCPer* in plain, and in SIMD fashion securely compares X_c with TM_c using *secComp* (Chapter 6.3.4) and sends the result to EC (lines 3-5). EC decrypts and decodes the result and based on the comparison results it construct a vector will all zeros, except for the slot which belongs to the final classification which is one. Encodes and encrypts this vector to get $ones\_c$ and send it to the Server (lines 6-9). The server multiplies $ones\_c$ with $classes\_p$ and sends the result to EC (lines 10-11). EC decrypts and decodes it to get the final classification result. For increased throughput we can classify up to $q = N/f$ queries, where N is the polynomial modulus.

For the RF cases we have several DT encoded one after the other at $TM\_p, classes\_p$, while the general idea and protocol flow remains pretty much the same.

ALGORITHM 6.23: ***secDT_RF*** **(secure Decision Tree and Random Forest)**

INPUT: $\boldsymbol{TM\_p, classes\_p, X\_v, f}$
$\boldsymbol{TM\_p}$: the decision trees or random forests' ciphertext kept privately at the server (owner)
$\boldsymbol{classes\_p}$: the vector of classes of the corresponding tree(s)
$\boldsymbol{X\_v = \{X_1, \dots, X_f\}}$**:** the EC's query vector of dimension *f*. We can have $q = N/f$ such queries

OUTPUT: $\boldsymbol{maxIndex\ and\ topKIndexes}$
$\boldsymbol{C_{DT}(X)}$: the final classification of q_v according to $DT\ or\ RF$ a

```
Server:
using SRCPer in plain, publishes the permuted order by which the clients should encode their feature values
EC:
1 $X\_c = EncEncr(X\_v)$
2 send $X\_c$ to Server
Server:
3 $X\_c = invSRCPer(k, R_1, R_2, N, 1)$
4 $result\_c = secComp(X\_c, TM\_p)$
5 send $result\_c$ to EC
EC:
6 $result\_v = DecrDec(result\_c)$
7 $ones\_v = putOnesTo\ TheClassesSpots(result\_v)$
8 $ones\_c = EncEncr(ones\_v)$
9 send $ones\_c$ to Server
Server:
10 $C_{DT}(X\_v)\_c = ones\_c \times classes\_p$
11 send $C_{DT}(X\_v)\_c$ to EC
EC:
12 $C_{DT}(X\_v) = DecrDec(C_{DT}(q\_v)\_c)$
```

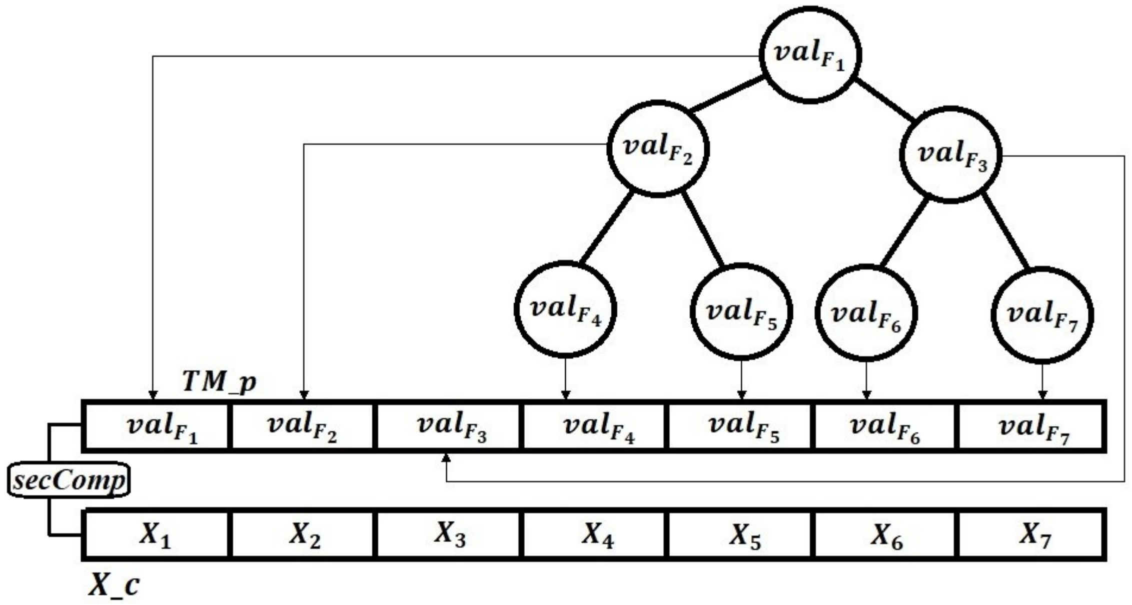


Fig. 6.16. Encoding the values of each level's node into the trained model plaintext TM_p for DT and RF classifier when $f = 7$

6.6. Secure MNB and NB classifications for binary and multi-label multi-output textual datasets

In Algorithm 6.24 we provide the MNB secure classification scheme for binary textual queries (datasets). It can be seen as a natural continuation of Algorithm 5.2. The necessary information background and notations are given in Chapter 2.1.2. In Phase XII EC encodes its query q_v, which has 1 in the first slot (index), proceeded by the query frequencies of each word in the query ordered according to the words arrangement in the selected features sets SF (line 1, Fig.6.17). After multiplying the encoded query with the trained model TM_c it sends the result to TEAS (lines 2-3, Fig. 6.17). In Phase XIII, in accordance to (2.7), TEAS homomorphically finds the sums of each of the $(m+1)$ slots in $log_2(m+1)$ rotations and additions with the result residing in the first slot (lines 5-6). To do this $(m+1)$ should be a power of two. If it is not the case, then we can pad extra slots with dummy values (usually zeros). Then, for secure comparison purposes (Chapter 6.3.4), multiplies this result with a random R_p followed by adding h_p s.t. R_p and h_p are constructed having in mind the secure comparisons requirements for them in Section IVC. To protect the result from the STC (substitute-then-compare) attack from [10], TEAS adds an extra random

$h2\_p = Encode(h\_v)$ to the result, sends the randomized result to E2DS and the $h2\_v$ to EC (lines 7-10). In Phase XIV E2DS decrypts the randomized classification result and sends it to EC (lines 11-12). Finally, in Phase XV EC subtracts $h2\_v$ from the randomized classification result and, according to (2.7), gets the final classification based on the sign of the result (lines 14-15). If the query q_v instead of the frequencies contains the counts ($N_q(w_i)$) of words appearing in the query, then instead of MNB, we're dealing with the NB for textual classifications.

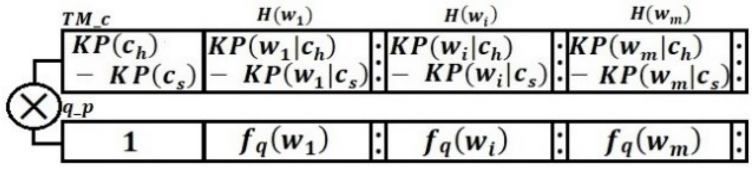

TM_c
H(w_1)
H(w_i)
H(w_m)
KP(c_h) − KP(c_s)
KP(w_1|c_h) − KP(w_1|c_s)
KP(w_i|c_h) − KP(w_i|c_s)
KP(w_m|c_h) − KP(w_m|c_s)
q_p
1
f_q(w_1)
f_q(w_i)
f_q(w_m)

Fig.6.17. SWHE SIMD multiplication of the trained model TM_c obtained in Algorithm 5.2 with query q_p.

ALGORITHM 6.24*: secC* (<u>sec</u>ure <u>C</u>lassification)

INPUT: $SF, TM\_c, X\_v, (pk, sk)$
(pk, sk): key pairs of E2DS with SWHE properties
$SF = \{H(w_1), \ldots, H(w_m)\}$: the set of hashes of m selected features with the highest IG
TM_c: the binary trained model ciphertext residing at EC
$X\_v = \{1, f_q(w_1), \ldots, f_q(w_m)\}$: the EC's query vector

OUTPUT: $C_{TM}(q)$
$C_{TM}(X)$**:** the q_v's final classification

```
PHASE XII - EC:
1 $X\_p = Encode(X\_v)$
2 $C_{TM}(X)\_c = TM\_c \times X\_p$ //so the TM_c already resides at EC (tell the reason behind it)
3 send $C_{TM}(X)\_c$ to TEAS
PHASE XIII-TEAS
4 $tmp\_c = C_{TM}(X)\_c$
5 for $i = 0$ to $\lceil log(m+1) \rceil$
6     $C_{TM}(X)\_c = C_{TM}(X)\_c + Rotate(C_{TM}(X)\_c, -2^i)$
7 $(R\_v, h\_v, h2\_v) = rndVectorsforComp()$; $(R\_p, h\_p, h2\_p) = Encode(R\_v, h\_v, h2\_v)$
8 $rndC_{TM}(X)\_c = ((C_{TM}(X)\_c \times R\_p) + h\_p) + h2\_p$ //to protect from the STC attack
9 $send\ rndC_{TM}(X)\_c$ to E2DS
10 send $Enc_{pk_{EC}}(h2\_v)$ to EC
PHASE XIV-E2DS
11 $rndC_{TM}(X)\_v = DecrDec(rndFinClas\_c)$
12 send $Enc_{pk_{EC}}(rndC_{TM}(X)\_v)$ to EC
PHASE XV-EC
13 $res\_v = Dec_{sk_{EC}}(rndC_{TM}(X)\_v) - Dec_{sk_{EC}}(h2\_v)$
14 if $(res\_v \geq 0)$   return $C_{TM}(q) =$ "ham"
15 else return $C_{TM}(X) =$ "spam"
```

Having in mind the notations and the corresponding background information in chapter 2.1.2, in Algorithm 6.25 we provide a multi-label multi-output secure classification algorithm for textual queries (datasets). Algorithm 6.25 can be seen as naturally following Algorithm 5.3. and, in general lines, it has the same logic as Algorithm 6.24, but expanded to deal with multi-label multi-output queries.

ALGORITHM 6.25*: secC-MLMO* (secure Classification for Multi-Label Multi-Output queries)

INPUT: $\boldsymbol{SF^{MLMO}} = \left\{\{\boldsymbol{SF^l}\}_{\boldsymbol{l=1}}^{|\boldsymbol{L}|}\right\} = \left\{\left\{\{\boldsymbol{H}(\boldsymbol{w_i})\}_{\boldsymbol{i=1}}^{\boldsymbol{m^l}}\right\}_{\boldsymbol{l=1}}^{|\boldsymbol{L}|}\right\}, \boldsymbol{m^l}, \boldsymbol{TM^{MLMO}\_c}, \boldsymbol{q\_v^{MLMO}}, (\boldsymbol{pk}, \boldsymbol{sk})$

$(\boldsymbol{pk}, \boldsymbol{sk})$: key pairs of E2DS with SWHE properties
$\boldsymbol{SF^{MLMO}}$ the set of $|L|$ set hashes, each of m^l selected features with the highest IG
$\boldsymbol{TM^{MLMO}\_c}$: the MLMO trained model ciphertext residing at EC
$\boldsymbol{q\_v^{MLMO}} = \left\{\left\{\left\{1, \left\{f_q^l\left(w_i^l\right)\right\}_{i=1}^{m^l}\right\}_{c=1}^{|C^l|}\right\}\right\}_{l=1}^{|L|}$: the EC's MLMO query vector, for the classes of the same label, it's replicated for $|C^l|$times

OUTPUT: $\boldsymbol{C_{TM}^{MLMO}(X)}$
$\boldsymbol{C_{TM}^{MLMO}(X)}$**:** the q_v^{MLMO}'s final classification

```
PHASE XII - EC:
1 $X\_p^{MLMO} = Encode(q\_v^{MLMO})$
2 $C_{TM}^{MLMO}(X)\_c = TM^{MLMO}\_c \times X\_p^{MLMO}$ //so the TM_c already resides at EC (tell the reason behind it)
3 send$C_{TM}^{MLMO}(X)\_c$ to TEAS
PHASE XIII-TEAS
4 $m = \underset{1 \leq l \leq |L|}{argmax}(m^l)$
5 for $i = 0$ to $\lceil log(m+1) \rceil$
6     $C_{TM}^{MLMO}(X)\_c = C_{TM}^{MLMO}(X)\_c + Rot(C_{TM}^{MLMO}(X)\_c, -2^i)$
7 $rndC_{TM}^{MLMO}(X)\_c = SRCPer(C_{TM}^{MLMO}(X)\_c)$// creates $permutData\_v$ needed for the permutations
8 $rndC_{TM}^{MLMO}(X)\_c = SCADS(rndC_{TM}^{MLMO}(X)\_c)$
9 $send\ rndC_{TM}^{MLMO}(X)\_c$ to E2DS
10 send $Enc_{pk_{EC}}(permutData\_v)$ to EC
PHASE XIV-E2DS
11 $rndC_{TM}^{MLMO}(X)\_v = DecrDec(rndC_{TM}^{MLMO}(X)\_c)$
12 send $Enc_{pk_{EC}}(rndC_{TM}^{MLMO}(X)\_v)$ to EC
PHASE XV-EC
13 $C_{TM}^{MLMO}(X) = invSRCPer(Dec_{sk_{EC}}(rndC_{TM}^{MLMO}(X)\_c), Dec_{sk_{EC}}(permutData\_v))$
```

6.7. Secure classifications based on linear algebra operations

In Chapter 2.3.3 we give the necessary notations and background information related to ML classification based on linear algebra. Utilizing the secure algebra building blocks introduced in Chapter 6.4, in Algorithm 6.26 and 6.27 we give a general secure classification scheme which is applicable to any ML classification algorithm that can be expressed in terms of linear algebra operations, particularly in vector and matrix operations. Depending on where the bulk of the operations are being done, they are either client (like algorithm 6.26) or Server centric (Algorithm 6.27)

Secure linear algebra based ML classification algorithm flows at-a-glance. We give both the client and the server centric flavors. The server owns a trained model which he wants to keep private, while the client wants to use it for secure ML classifications based on linear algebra operations having in mind the security, privacy and efficiency requirements given in Chapter 1.

Client Centric (Fig. 6.18): 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 with SWHE properties (Chapter 2.4.1). The client(s) construct(s) his/their queries $X^{(i)}$(❶), encode and (depending on the circumstances, might also) encrypt them to get the packed queries in a single plaintext (S_p) or ciphertext (S_c) (❷). Using the encrypted trained model M they do computations over encrypted data to get the classification result $C_M(S)$ in encrypted form (❸). After randomizing the result they get the encrypted $rndC_M(S)$ and send it to the server (❹). After decrypting and decoding it, the server obtains the randomized result in plain, $rndC_M(S)\_v$, and sends it back to the client(❺). Finally, the client de-randomizes $rndC_M(S)\_v$ to get the final classification for the query(ies), $C_M(S)$ (❻). More details are given in Algorithm 6.26.

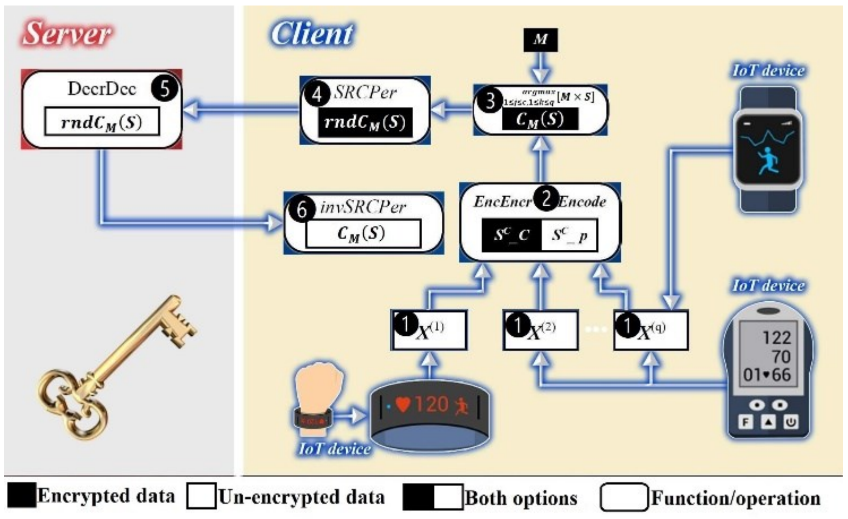


Fig.6.18. Secure linear algebra ML classification algorithm flow-client centric

Server centric (Fig.6.19)**:** the trained model M resides at the server un-encrypted. All encryptions are done using the client's public key with SWHE properties (Chapter 2.4.1). The client(s) construct(s) his/their queries $X^{(i)}$(❶), encode and encrypt them and send them to the Server (❷). The server adds the encrypted queries up and does the computations over encrypted queries to get the classification result $C_M(S)$ in encrypted form (❸). Afterwards randomizes the result to get the encrypted $rndC_M(S)$ and send it to the client(s) (❹). After decrypting and decoding it, the server obtains the randomized result in plain, $rndC_M(S)\_v$ (❺), and based on it constructs, encode and encrypts a new $rndC_M(S)$ which is send back to the server (❻). The server removes the randomization to get the final encrypted classification $C_M(S)$, which is send to the clinet(s) that decrypt it to get the final classification for the(ir) query(ies), $C_M(S)$. More details are given in Algorithm 6.27.

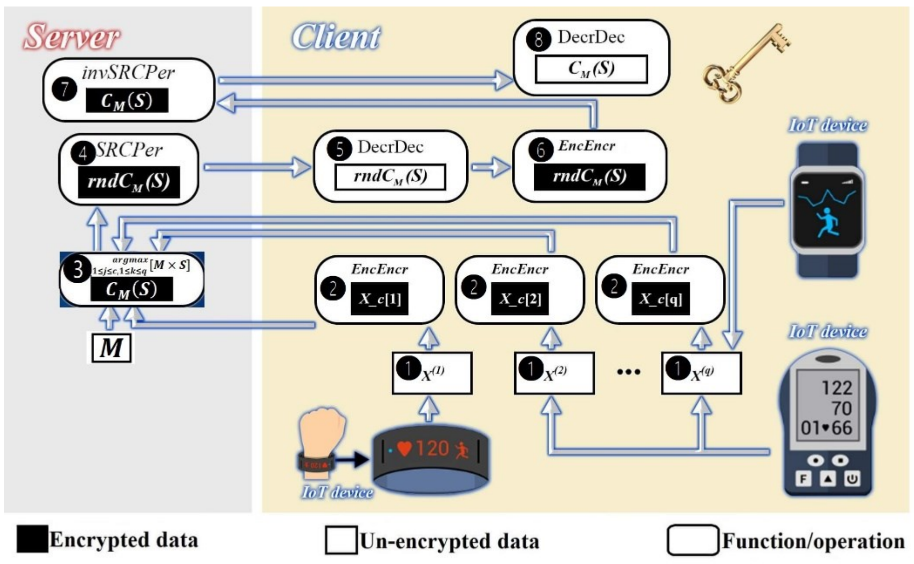


Fig.6.19. Secure linear algebra ML classification algorithm flow-server centric

The client centric version (Algorithm 6.26) as an input takes the row-encoded trained model ciphertext and the queries of $q = \frac{N}{c \cdot (f+1)}$ users in a scenario where we have f features and c classes (Chapter 2.3.3). IoT devices/clients encrypt their queries and rotate them, in a way that when homomorphically added up, they form a column-wise encrypted matrix (lines 1-4). Then homormorphcally classify their queries with the trained model according to (2.21) (lines 5-6). In line 6 *SCADS*(·) -Chapter 6.3.5-, is used for the *argmax* purposes of (3). The input ciphertext of *SCADS*(·) looks exactly what $secMatMat(\cdot)$ returns in line 5. After randomization with *SRCPer*(·) (Chapter 6.3.3), the results are send to the server (line 7-8). The server decrypts and sends back the randomized results in plain (lines 9-10). The clients de-randomize them to get the final classifications (line 11). *SRCPer*(·) and it's inverse, *invSRCPer*(·), proposed in Chapter 6.3.3, are used for randomization and de-randomization purposes (lines 7, 11). The values we give to the random parameters that *SRCPer*(·) and *invSRCPer*(·) take are $R_1 = 0,\ R_2 = 0$, while the random input vector k has $m = c$ elements and it is used to randomly permute inside the block the positions of the comparisons results for each of the c data slots (corresponding to c classes) for each of the q blocks (queries), simultaneously.

ALOGRITHM 6.26: ***secMLClass*** **– (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 **M**odel

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

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

OUTPUT: $\boldsymbol{C_M(S)\_v}$

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

```
Client:
1  for $i = 1$ to $q$
2      $X\_c[i] = EncEncr(X^{(i)})$
3      $X\_c[i] = Rotate(qVec\_c[i], i \times (f+1))$
4  $S^C\_c = AddMany(X\_c[])$
5  $tmp\_c = secMatMat(M^R\_c, S^C\_c, c, f+1, q)$
6  $C_M(S)\_c = SCADS(tmp\_c, c, f+1)$
7  $rndC_M(S)\_c = SRCPer(C_M(S)\_c, k, R_1, R_2, c, f+1)$
8  send $rndC_M(S)\_c$ to Server
Server:
9  $rndC_M(S)\_v = DecrDec(rndC_M(X)\_c)$
10 send $rndC_M(S)\_v$ to Client
Client:
11 $C_M(S)\_v = invSRCPer(rndC_M(S)\_v, k, R_1R_2, c, f+1)$
```

Algorithm 6.27 (server centric) in general is similar with Algorithm 6.26 (client centric), with the difference that in Algorithm 6.27 all encryptions are done using client's public key with SWHE properties, whereas in Algorithm 6.26 using servers public key with SWHE properties. What is more important, the bulk of the heavy homomorphic computations at the server centric algorithm are done at the server, whereas at the client centric algorithm they are done at the client. While the client centric algorithm is done in one round, the server centric one is done in two rounds, having in the process slightly heavier computation cost.

ALOGRITHM 6.27: ***secMLClass*** **(secure ML Classifications – Server Centric)**

INPUT: $\boldsymbol{M^R\_c}, \boldsymbol{S} = \{\boldsymbol{X^{(i)}}\}_{\boldsymbol{i=1}}^{\boldsymbol{q}}, \boldsymbol{c}, \boldsymbol{f}$
$\boldsymbol{M^R\_c}$: row-wise encrypted Trained **M**odel
$\boldsymbol{S} = \{\boldsymbol{X^{(i)}}\}_{\boldsymbol{i=1}}^{\boldsymbol{q}}$,: the set of q user queries, $q = \frac{N}{c\cdot(f+1)}$
$\boldsymbol{c}, \boldsymbol{f}$**:** the number of **c**lasses and **f**eatures, respectively

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

```
Client(s)
1  for $i = 1$ to $q$
2      $X\_c[i] = EncEncr(X^{(i)})$
3      $X\_c[i] = Rotate(qVec\_c[i], i \times (f+1))$
4      send $X\_c[i]$ to Server
Server:
5  $S^C\_c = AddMany(X\_c[])$
6  $tmp\_c = secMatMat(M^R\_c, S^C\_c, c, f+1, q)$
7  $C_M(S)\_c = SCADS(tmp\_c, c, f+1)$
8  $rndC_M(S)\_c = SRCPer(C_M(X)\_c, k, R_1, R_2, c, f+1)$
9  send $rndC_M(S)\_c$ to Client(s)
Client(s)
10 $rndC_M(S)\_v = DecrDec(rndC_M(S)\_c)$
11 $rndC_M(S)\_v = putOnesToTheMaxClassOfAllQueries()$
12 $rndC_M(S)\_c = EncEncr(rndC_M(S)\_v)$
13 send $rndC_M(S)\_c$ to Server
Server:
14 $C_M(S)\_c = invSRCPer(rndC_M(S)\_v, k, R_1R_2, c, f+1)$
15 send $C_M(S)\_c$ to Client(s)
Client(s):
16 $C_M(S)\_v = DecrDec(rndC_M(S)\_c)$
```

Note: for the DNN with l layers, the trained model M^R_c in Algorithms 6.26 and 6.27 is made of l matrices denoted as M^{Ri}_c, where $1 \leq i \leq l$, the activation functions f^i, where $1 \leq i \leq l$, are polynomial ones (usually square or linear functions) and S^{Ci} is the output of the previous layer, where for the input in the first layer we have $S^{C0} = S^C$. In this sense, to abide to (2.22), for a DNN with l layers line 5 in Algorithm 6.26 (i.e. line 6 in Algorithm 6.27) should be changed to $secMatMatCas(f^i(M^{Ri}\_c, S^{Ci}\_c, c, f+1, q))$ which is executed for l times. Algorithm $secMatMatCas(\cdot)$ is explained in Chapter 6.4.

Improvement 6.1: For algorithms 6.26 and 6.27, if the users know their order, instead of encoding their data at the beginning of the ciphertext's slots, they can directly put them to their corresponding place, i.e. EC i put his $f + 1$ data from slot $(i - 1)(f + 1)$ till $i(f + 1)$ slots. In this way they both avoid the costly operation of rotations in line 3.

Improvement 6.2. Unlike it was the case up until now in literature, for efficiency purposes, no need for slots in Algorithms 6.20-6.27 to be powers of two anymore due to introducing Algorithms 6.1 and 6.2.

Improvement 6.3. Whenever possible in Algorithms 6.20-6.27 we introduce the poly-switching technique proposed in [55]. We also use multiple-cores for increasing furthermore the throughput of processed queries by simultaneously classifying them among multiple processor cores. For this each core should have a copy of the trained model in the core's local cache memory. Those couple of techniques alone give an improved computational and communicational cost for several times.

Improvement 6.4. The 3PC algorithms that exist in some of the Algorithms in 6.20-6.25 can be converted to 2PC. Furthermore, having in mind Improvement 5.1 and knowing that TM_c rarely changes, it can be send to the user only once for the client-centric classification, thus amortizing this cost for all of the subsequent classifications to follow.

Improvement 6.5: Besides Algorithms 6.26-6.27, for high throughput, in Algorithms 6.20-6.25 we can also simultaneously process several queries by replicating the trained model and packing several queries in the query vector. E.g. for *secC* (Algorithm 6.24) we can process up to $q = \left\lfloor \frac{N}{(m+1)} \right\rfloor$ queries by obtaining a replicated TM_c during $secT$ (Algorithm 5.2), which can be done without extra costs, and encoding (packing) q queries to q_p during $secC$.

6.8. Dealing with malicious users during classifications

Unlike the semi-honest model (Chapter 2.4.2), malicious users are active adversaries that arbitrarily deviate from the protocol with the aim of retrieving partially or totally the data that they are not supposed to or with the aim of sabotaging the protocol.

In algorithm 6.28 we deal with a malicious EC user for the Algorithm in 6.21. thus it is valid for non-textual data dealing with NB classifiers. One of the attacks that such a malicious user might come up with during the server-centric classification stage in Algorithm 6.21 is to put 1s (ones) in only two slots corresponding to the same feature-value but different classes, i.e. put ones to $KlogP(V_{m,Fi}; Cj)$, s.t. m and F_i are the same but C_j is different) or put ones to two different class probabilities ($KlogP(C_j)$ and all the other slots are set to zeros (Fig.6.13 and 6.14). In this way, while running *PPClassServCen* (Algorithm in 6.21), the user can find which of the two probabilities is greater than the other. Furthermore, if instead of ones, in the same fashion the user puts some random values R_1 and R_2 into two slots corresponding to the same feature-value but different classes, then after executing *PPClassServCen* for several times with different random values for R_1 and R_2, ultimately the user can find the ratio of those two probabilities. In both cases we have a leakage that goes against the strict classification goals mentioned in Chapter 1. Algorithm 6.28 deals with such active malicious users.

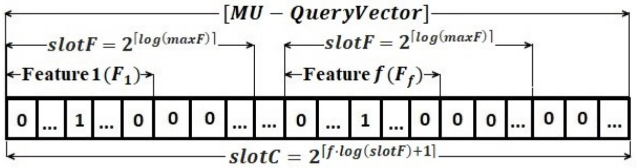

$[MU - QueryVector]$
$slotF = 2^{\lceil log(maxF) \rceil}$
$slotF = 2^{\lceil log(maxF) \rceil}$
Feature 1 (F_1)
Feature f (F_f)
0 … 1 … 0 0 0 … … 0 … 1 … 0 0 0 … 0 0 …
$slotC = 2^{\lceil f \cdot log(slotF)+1 \rceil}$

Fig.6.20. SIMD construction of the $MU - QueryVector\_c$ with padded zeros added for the need of the *secSum* Algorithm

We assume that TEAS and E2DS are still in the semi-honest model and they don't collude. To avoid any attack, we should make sure that the malicious user behaves properly while executing the protocol, especially while constructing the $queryVector$, hence the q$ueryVector_c$ (Fig.6.13 and 6.14), which for the case of the malicious user will be slightly altered and named as $MU - QueryVector\_c$ and $MUencryptedQueryVector\_c$, respectively. By proper behavior from the malicious user we mean that for each feature F_i

s.t. $1 \leq i \leq f$, exactly one slot per feature should have 1 (one) inserted at the corresponding feature–value slot and all others slots should bet set to zeros, just as it is explained in Phase I of Algorithm 6.21 (Fig.6.13 and 6.14). To make sure that this is the case we run *secSum* (Algorithm 6.1) to simultaneously and privately find the sum of block of slots corresponding to each feature and check whether each of those sums are 1 or not. Based on this outcome, the other participants (TEAS and E2DS) decide whether to continue or abort the protocol. In order to do this, due to the needs of the old version of *secSum*, we will have to allocate $slotF = 2^{\lceil\log(maxF)\rceil}$ slots per feature, where $maxF$ is the feature with the biggest cardinality (number of elements), i.e $maxF = \max(|F_i|)$ for $1 \leq i \leq f$. Since we have f features, and in order to find the posterior probabilities for each class we should again use the *secSum* algorithm for the second time, then for each class we need $slotC = 2^{\lceil f \cdot \log(slotF)+1\rceil}$ slots, where the term+1 (extra one slot) is the slot for the class probability. All the extra added slots are padded with (have values) of 0 (zero). Malicious user's $MU - QueryVector\_c$ is illustrated in Fig.6.20. The corresponding pseudocode that builds this $MU - QueryVector - c$ is given in lines 1-9 for Phase I of Algorithm 6.29. In Phase II when TEAS gets this query, firstly it runs the *secSum* algorithms to find the sum of each feature block and sends the result to E2DS for checking (lines 10-12). Then constructs a plaintext named $ones\_p$ which, starting from the first slot, has ones in every $slotC$ slot and everything else is zero (lines 13-14). Afterwards TACS firstly rotates $MU - QueryVector\_c$ to the right by one slot (upper vector of Fig.6.21), replicates it for c times by calling the *CPack* algorithm from Chapter 6.4.14 and adds the $ones\_p$ plaintexts to it to get the final $MU - EncyrptedQueryVector_$c as shown in Fig.6.22 (lines 15-17), which has $slotQ = c \cdot slotC$ slots ($slotC$ slots for each of the c classes).

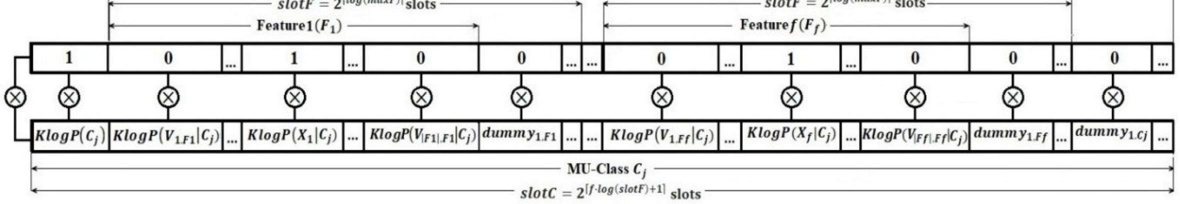


Fig.6.21. $MU - QueryVector\_c$ and portion of the $MU - TM\_c$ depicting slots related to class C_j

ALGORITHM 6.28: ***MU-PPClassServCen*** **(PP Classification With Malicious User - server centric)**

```
INPUT: $\boldsymbol{X} = \{\boldsymbol{X_1}, \boldsymbol{X_2} \dots \boldsymbol{X_f}\}, \boldsymbol{F}, \boldsymbol{C}, \boldsymbol{TM\_c}$
$\boldsymbol{X} = \{\boldsymbol{X_1}, \boldsymbol{X_2} \dots \boldsymbol{X_f}\}$: unclassified query feature vector owned by the User, s.t. $X_i \in F_i$ for $1 \le i \le f$
$\boldsymbol{F}$: $F = \{F_1, F_2, \dots, F_f\}$, where $F_i = \{V_{1,Fi}, V_{2,Fi}, \dots, V_{|Fi|,Fi}\}$. $F_i$, st. $1 \le i \le f$ (as explained in Section III-A)
$\boldsymbol{C}$: The set of classes $C = \{C_1, C_2, \dots, C_c\}$ (as explained in Section III-A)
$\boldsymbol{TM\_c}$: an already SIMD encrypted trained Naïve Bayes model stored at TEAS
OUTPUT: $\boldsymbol{C_{TM}(X)}$
$\boldsymbol{C_{TM}(X)}$: the classification of the query feature vector X according to $TM\_c$
Phase I – EC User:
1  $maxF = findMaxFeatSize(F)$   //find the feature with the biggest cardinality (number of elements)
2  $slotF = 2^{\lceil log(maxF) \rceil}$; $slotC = 2^{\lceil f \cdot log(slotF)+1 \rceil}$; $slotQ = c \cdot slotC = c \cdot 2^{\lceil f \cdot log(slotF)+1 \rceil}$
3  $queryVector\_v.insert(slotQ, 0)$   //insert $slotQ = c \cdot slotC$ 0s (zeros) to the $queryVector\_v$
4  for $i = 1$ to $f$   //for each feature
5      for $m = 1$ to $|F_i|$   //for each value of the current feature (feature $F_i$)
6          if $X_i == V_{m,Fi}$ then   //if $X_i$ is equal to the current $V_{m,Fi}$ feature value of feature $F_i$
7              $queryVector\_v[i \cdot maxF + m] = 1$   //then insert one to index $i \cdot maxF + m$
8  $MU - QueryVector\_c = Encode\_Encrypt(queryVector\_v)$ //SIMD encod. then encrypt., Fig.A.1
9  send $k, R_1, R_2, MU - QueryVector\_c$ to TACS //send the malicious users' random k, $R_1$, $R_2$ to TEAS
Phase II – TEAS:
10 receive $k, R_1, R_2, MU - QueryVector\_c$
11 $sumResult\_c = secSum(MU - QueryVector\_c, slotF)$ //finds the sum of the (1s) for each feature
12 send $sumResult\_c$ to EDS   //lines 13-17 can be done in parallel with Phase III
13 $ones\_v = \{1,0,\dots,0,1,0,\dots,\}$ //a vector that has $c$ ones after each $slotC$ slots, starting from the first slot
14 $ones\_p = Encode(ones\_v)$ //constructing a plaintext, rather than a ciphertext for performance reasons
15 $MUQueryVector\_c = Rotate(MU - QueryVector\_c, 1)$//rotation for 1 place to the right to make place
//for 1 in the begin. is needed to be multiplied with the class-conditional prob. (upper vector Fig.A.2)
16 $MUEncryptedQueryVector\_c = CPack(MUQueryVector\_c, slotC, c)$ //replicates the
//$MUQueryVector\_c$ for $c$ times to get the upper vector of Fig.A.3, without the 1s (ones) in the beginning of
//each class slot
17 $MUEncryptedQueryVector\_c = MUEncryptedQueryVector\_c + ones\_p$
//here we add the 1s (ones) at the beginning of each class slot to finally get the upper vector shown in Fig.A.3
Phase III – E2DS:
18 receive $sumResult\_c$   //$sumResult\_c$ was obtained from line 11
19 $sumResult\_v = Decrypt\_Decode(sumResult\_c)$
        //HasOnes(·) checks whether there are ones at the beginning of each features' slot at $sumResult\_c$
20 if($HasOnes(sumResult\_v)$)   //if it is the case
21     send ($outcome = true$) to TEAS   //send true to TEAS so we can continue with the protocol
22 else   //otherwise
23     send ($outcome = false$) to TEAS   //send false to abort the protocol
Phase IV – TACS:
24 recieve $outcome$
25 if(outcome == false) abort the protocol
26 else
27 return $PPClassServCen(MU - TM\_C, MUEncryptedQueryVector\_c, k, R_1, R_2)$ from line 13
//PPClassServCen (Algorithm 6.21) is executed from line 13, during the execution $n_L = slotQ, n_c = slotC$,
//as shown in Fig. A.3
//$queryVector\_c$ is replaced by $MUEncryptedQueryVector\_c$ and $TM\_c$ replaced by $MU - TM\_C$
```

In Algorithm 6.21 the replication is done at the user side (EC), but here we do it at TEAS since the malicious user might put a different query vector (Fig.6.13 and 6.14) for each class. Lines 13-17 can be done in parallel with Phase III.

In Phase III E2DS receives, decrypts and checks whether the result of the *secSum* done at TEAS is proper (it should have one at the begging of each feature slot, i.e. ones after each $slotF$ slots) (lines 18-23). If that's the case then TEAS is informed to proceed, otherwise it should abort.

In Phase IV, if TEAS is signaled to abort, it does so (line 25). If not, TACS proceeds by executing $PPClassServCen$ (Algorithm 6.21) from line 13, and while doing so the $queryVector\_c$ is replaced by $MUEncryptedQueryVector\_c$, and the TM_c is replaced by $MU - TM\_c$ so that their corresponding slot constructions are illustrated in Fig.6.21 for one class and in Fig.6.22 for the whole construction (all classes).

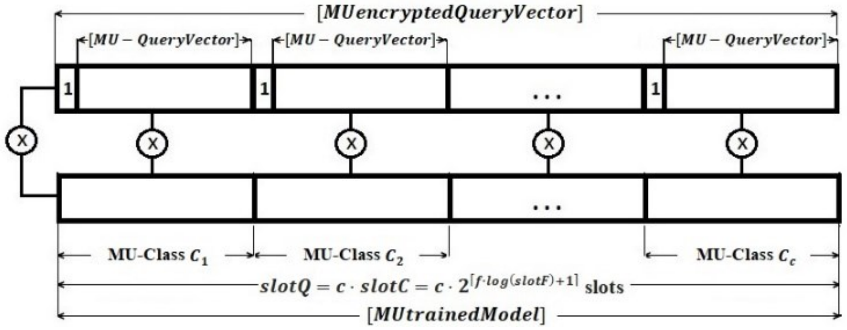


Fig.6.22. $MUencryptedQueryVector\_c$ multiplies $MU - TM\_c$

Improvement 6.6: Unlike it was the case till now in the literature, no need for slots to be powers of two anymore due to Algorithms 6.1 and 6.2.

In Algorithm 6.29 we deal with malicious user(s) (client(s)) for the cases of secure ML classifications based on linear algebra operations, which were presented in Algorithms 6.26 and 6.27. In those scenarios a malicious EC user instead of putting zeros into slots that are not meant (not designated) for him, in order to disrupt the secure classifications for the other EC's, the malicious EC can put dummy values other than zeros, which will sabotage the protocol for the other EC's by having them get inaccurate classifications. In order to protect from such malicious EC users, before adding (summing, packing) up the EC's queries into one query, each EC's query is firstly multiplied by a plaintext mask which has f 1s (ones) at the slots which are designated for that particular EC and zeros elsewhere (lines 1-4). Afterwards it can be continued with both Algorithms 6.26 and 6.27 from line 4.

ALOGRITHM 6.29: ***secMLClass-MU*** **(secure ML Classifications – Malicious Users)**

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

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

```
Client(s)
1 $maskSecMLClass\_p[] = masksForSecMlClass()$
2 for $i = 0$ to $q$
3     $X\_c[i] = EncEncr(X^i)$
4     $X\_c[i] = X\_c[i] \times maskSecMlClass\_p[i]$
//continue with line 4 for both Algorithm 6.26 and 6.27
```

6.9. Theoretical and Experimental evaluations and comparisons

In this Chapter we provide the theoretical and experimental evaluations and comparisons among building blocks and secure classifications algorithms (protocols) from different schemes (research papers). We should note that the experimental evaluations of our proposed secure comparison protocol - *secComp*, hence of our secure comparison of all data slots – SCADS as well (since it is built on top of *secComp*), show that it doesn't offer a perfect hiding of the difference of the two numbers that are being compared (Section 5.9.1). Yet, our theoretical analysis show that when SCADS is used in combination with the our secure and private ML classification protocols it offers a total privacy of the trained model and the user query. We do this by giving a polynomial time reduction of the hardness of getting the trained model and the user query to the hardness of LWE (Section II). This is due to the fact that the matrix-vector product of the trained model matrix *TM* and the user query vector *X* of our Machine Learning classification protocols help us "convert" (polynomially reduce) SCADS into an LWE problem, as it is proven in Section 6.9.2.

6.9.1. Theoretical analysis and comparisons

Since secure comparison and secure argmax (secure top-K) are among the most important and most used building blocks in secure ML classification algorithms, in Table 6.1 and 6.2 we provide and compare their security, privacy and efficiency properties among different schemes. In this Section we use the notations we mentioned in Chapter 2. Briefly, N is the number of slots, c is the number of classes, f number of features and $|F_i|$ is the number of elements (cardinality) of feature's value set F_i for $1 \leq i \leq f$, b is the number of bits a single ciphertext encoded in one slot has and n is the number of Edge Dataset Owners (EDOs). During the theoretical comparisons, for both the computation and communication purposes, except for schemes [61-62], we mostly take into consideration only the costliest terms which usually are due to the cryptographic techniques such as homomorphic encryption (HE), oblivious transfer (OT), private information retrieval (PIR), etc. In the process in bold we give a reference to the papers of corresponding cryptographic technique together with the number of invocations of the scheme or any of its' subroutine(s) (e.g. multiplication, addition, rotation for FHE or SWHE). Furthermore, for the scheme at [67], letters O and Ω represent the *big-*O and the Ω notation, respectively.

In Tables 6.1 and 6.2 we provide theoretical comparisons for the computation and computational costs for secure comparison (*secComp*) and secure argmax (*secArgmax*) protocols among different state-of-the-art schemes, respectively. At those secure schemes one of the parties has the encrypted data (two integers or an array of integers for the argmax case) and the other one has the secret key. At the end one finds the index or the maximum of two integers (or of an array of integers for the argmax case) while the other party usually learns nothing. During the secure argmax protocol, almost all of the schemes, several times invoke the corresponding secure comparison scheme of the same paper. In this manner all the computation and communication costs should be correspondingly added to both parties for each secure comparison invocation.

In our proposed scheme, during the *secArgmax* protocol, Party A executes once all of the *SRCPer*, *invSRCPer* and *SCADS* protocols described in Chapter 6.3. *SRCPer* has m plain multiplications, $(m + 2)$ rotations and $\log m$ additions, where m is the number of data slots in a block and it's a small integer (usually not greater than 6 or 7). The same applies for *invSRCPer*. SCADS has c rotations, $(c/2 + \log c + 1)$ additions and 1 plain multiplication. Thus we have $(2m + 1)$ plain multiplications, $(2m + 4 + c)$ rotations and $(2\log m + c/2 + \log c + 1)$ additions for the overall computation cost at Party A, which is shown in Table 6.2.

Table 6.3 gives the theoretical comparisons for the computation and communication costs among different schemes during the PP NB classification stage. For our scheme we put Algorithm 6.21. In the process we tend to use the described schemes in the most efficient and optimized way they can be utilized (especially the scheme described in [69]). However, we do this without losing the generality by making any assumption on the number of features f, classes c or cardinalities of F_i for $1 \leq i \leq f$. Also, during the PP classification, almost all of the schemes invoke the corresponding secure argmax or, when they deal with binary classification only (such as Gao et. al. [71]), the corresponding comparison protocol, which should be kept in mind while estimating the overall computation and communication cost for both (or all) of the participants. Furthermore, some of them several times call other cryptographic protocols or their subroutines such as OT, PIR, Pailler [49] etc., which should also be considered when estimating the overall communication and computation cost for both (or all) parties.

For the server-centric classification scheme of Park et.al. [69], for the Assign module we need 1 multiplication + 1 addition while for *SlotCopy* we need $|F_j| - 1$ rotations and $|F_j| - 1$ additions. Since for each feature we repeat both of the process once, and also having in mind the $|F_j|$ rotation for the *MaskGen* module, in total we have $TX_{query} = \mathrm{f} \cdot (1 \text{ multipliplication} + (2|F_j| - 1) \text{ rotations } + |F_j| \text{ additions})$.

Table 6.1. Theoretical comparison of the secure comparison (*secComp*) algorithm among different schemes

Property / Scheme	Place	Communication	Computation	Interac. (rounds)	Avoids Leaking any information
Bost et.al [53] Li et.al. [70] Gao et. al. [71]	Party A	**GM** [50]: 2 ciphers **DGK** [95]**:** 1 invocation	**Pailler**[49]: 1 decryption **GM** [50]: 1 encryp..+2 mult. **DGK** [95]**:** 1 invocation	3 rounds	YES
	Party B	**Pailler**[49]: 1 cipher **GM** [50]: 1 cipher **DGK** [95]**:** 1nvocation	**Pailler**[49]: 1 encr. + 3 mult. **GM** [50]: 1 encr. + 1 mult. **DGK** [95]**:** 1 nvocation		
Park et.al [69]	A	**KLY** [96]: 1 invocation	**KLY** [96]: 1 invocation	1 round	YES
	B	None	**KLY** [96]: 1 invocation (i.e. 1 decryption only for Party B)		
Sun et.al [73]	Party A	None	**BGV-improved** [73]: 1 decryption	1round	If run several times, the difference of the two numbers might be exposed
	Party B	**BGV-improved** [73]: 1 ciphertext	**BGV-improved** [73]: 2 multiplicat. + 3 additions		
Khedr et.al [74] Pereira [76]	No secure comparison at all. The class probabilities are totally leaked since their exact values can be decrypted by the user				
Liu et.al [66]	A	**Pailler**[49]: 3 ciphertexts	**Pailler**[49]: 1decr. + 1 enc. + 2 encr. **or** 2 multip.	1 round	YES
	B	**Pailler**[49]: 3 ciphertexts	**Pailler**[49]: 6 const. expon. + 11 multip.		
Liu et.al [67]		**BGV** [55]: 1 ciphertext	**BGV** [55]: $(log b + b)\cdot$(rotat.+addition) + $b\cdot$(multip. + const. multip.)	1 round	YES
Kjamilji et. al [72]	A	None	**FV** [56]: 1 decryption	1 round	If run several times, the difference of the two numbers might be exposed
	B	**FV** [56]: 1 ciphertext	**FV** [56]: 1 multiplication + 1 additions for potentially N SIMD comparisons		
Yasamura et.al [77] Wood et.al. [78] Wood et. al. [79]	A	None	**BGV-like:** 1 decryption	1 round	The difference of the two numbers is totally exposed
	B	**BGV-like:** 1 ciphertext	**BGV-like:** 1multiplication + **1** addition		
Our scheme	Party A	None	**FV:** [27]: 1 decryption	1 round	Partially
	Party B	**FV** [56]: 1 ciphertex. for potentially N SIMD comparisons	**FV:** [56]: 1 multiplication + 2 additions for potentially N SIMD comparisons		

Table 6.2. Theoretical comparison and properties of the secure argmax (*secArgmax*) algorithm among different schemes

Property \ Scheme	Bost et.al [53] Li et.al. [70]		Park et.al [69]		Sun et.al [71]		[74]	Liu et.al [66]		Liu et.al [67]	Gao et. al. [71]	Kjamilji et. al [72]		Yasamura et. al. [77] Wood et.al. [78], [79]		**Our scheme**	
Place	**A**	**B**	**A**	**B**	**A**	**B**		**A**	**B**			**A**	**B**	**A**	**B**	**A**	**B**
Communication	$c-1$ times of each of the below: **Corresponding SecComp**: 1 **Paillier [49]:** 2 ciphertexts	c-1 times of each of the below: **Corresponding SecComp**: 1 **Paillier**[49]: 3 ciphertexts	None	**BGV-like** [55]: 1 ciphertext transmission .	**Corresponding SecComp:** $(c-1)$ invocat. with party B	**Corresponding SecComp:** $(c-1)$ invocate.with party B	The scheme deals with multinomial Naïve Bayes, which has two classes. So no need for secure argmax	**Corresponding SecComp:** $(c-1)$ invocat. with party B	**Corresponding SecComp:** $(c-1)$ invoca. with party B	This scheme does not support multiclass classification since. It does not have the argmax protocol	The scheme doesnt deal with multiclass classification, but it can easily be extended to support the one in [15]	**FV** [56]: $c(c-1)/2$ +1 ciphertexts	**FV** [56]: c ciphertexts	**BGV-like:** $c-1$ ciphertexts	**BGV-like**: $c-1$ bits	**FV** [56]: 2 ciphertexts	**FV** [56]: 1 ciphertext
Computation	$c-1$ times of each of the below: **Corresponding SecComp**: 1 **Paillier**[49]: 2 encryptions + 5 multiplications + 3 exponentiations	c-1 times of each of the below: **Corresponding SecComp**:: 1 **Paillier**[49]: 5 multiplications	**BGV-like** [55]: 1 decryption	**Correspon. SecComp:** $(c-1)/2$ **BGV-like** [55]: $(c-1)^2$ multipl.	**Corresponding SecComp:** $(c-1)$ invocations with party B	**Corresponding SecComp:** $(c-1)$ invocations with party B		**Corresponding SecComp:** $(c-1)$invocations with party B	**Corresponding SecComp:** $(c-1)$invocations with party B			**FV** [56]: $\frac{c(c-1)}{2} + (c-1)$ additions and $\frac{c(c-1)}{2} + c$ multiplications	**FV** [56]:]: $\frac{c(c-1)}{2} + 1$ decryptions	**BGV-like**: $c(c-1)$ multiplications + $c(c-1)$ additions	**BGV-like:** $c-1$ decryptions	**FV** [56]: $(2m+1)$ plain multiplicat. + $(2m+4+c)$ rotations + $(2logm + c/2 + logc + 1)$ additi.	**FV** [56]: 2 decryptions + 1 encryption
Interactions (nr. of rounds)	$3 \cdot (c-1)$ rounds		1 round (SIMD)		$c-1$ rounds			$c-1$ rounds				2 rounds		$c-1$ rounds		1 round (SIMD fashion)	
Avoids Leaking any information	YES		YES		Partially leaks the order (sequence) of the numbers, not their values			YES				YES		Partially leaks the order (sequence) of the numbers, and all of their differences		Partially	

Table 6.3 Theoretical comparison for the costs of the PP Naïve Bayes classification algorithm among different schemes

S	**Place**	**Computation**	**Communication**
[53], [77], [78]	**User**	**Pailler** [49] **(BGV-like** [55]**)**: cf multipications **Corresponding argmax:** 1 invocation with the server	**Corresponding argmax:** 1 invocation with the server
	Server	**Pailler** [49] **(BGV-like** [55]**)**: $c(1 + \Sigma\|Fi\|)$ encryptions (done once, so amortized among many users) **Corresponding argmax:** 1 invocation with the client	**Pailler** [49] **(BGV-like** [55]**)**: $c(1 + \Sigma\|Fi\|)$ ciphertext transmissions **Corresponding argmax:** 1 invoc. with client
[69] (serv.)	**User**	**BGV-like** [55]: 1 encryption for the query **Corresponding argmax:** 1 invocation with the server	**BGV-like** [55]: 1 ciphertext for the user query
	Server	**BGV-like** [55]: $T_{tot} = TX_{query} + T_{pack} + T_{rem}$ **Corresponding argmax:** 1 invocation with the user	**BGV-like** [55]: 1 ciphertext with the final result
Sun et.al [71]	**User**	**Corresponding argmax:** 1 invocation with the server	f integers (the query in plaintext) **Corresp. argmax:**1 invocation with the server
	Server	**BGV-improved** [73]: $c \cdot f$ mult. or f mutl. (SIMD vers.) **Corresponding argmax:** 1 invocation with the client	**Corresponding argmax:** 1 invocation with the client
Khedr et.al [74]	**User**	**KGV** [74]: $m \cdot h$ encrypt., 2 decrypt. (probs), where m is the nr. of words in the email, h is the hash size in bits	**KGV** [74]: $m \cdot h$ ciphertexts (encrypted bit hashes for all the unclassified email words)
	Server	**KGV** [74]: $d \cdot h$ encrypt. (amortized for all users), $m \cdot d \cdot h$ additions., $m \cdot d \cdot h$ multiplic., where d is the nr. of words in the database (word-bag), h is bit-hash	**KGV** [74]: 2 ciphertexts (the 2 probabilities)
Liu et.al [66]	**User**	**Pailler** [49]: $\Sigma\|Fi\|/2$ encryptions **Corresponding argmax:** 1 invocation with the server	**Pailler** [49]:]: $\Sigma\|Fi\|/2$ transmissions **Corresponding argmax:**1 invocation with the
	Server	**Pailler** [49]: $\Sigma\|Fi\|$ exponent. + $\Sigma\|Fi\|$ multiplic. **Corresponding Secure Sum** [33]: f invocations **Corresponding argmax:** 1 invocation with the client	**Correspond. Secure Sum** [66]: f invocations **Corresponding argmax:** 1 invocation with the client
Liu et.al [67]	**User**	**BGV** [55]: 1 encryption	**BGV** [55]: 1 ciphertext transmission
	Server	**BGV** [55]: $O(fb3 + max\|Fi\|b2)(t\_add + t\_rot) + O(fb2 + max\|Fi\|c + max\|Fi\|b)t\_ml + O(fb3 + max\|Fi\|c + cb2)t\_cml$	**BGV** [55]: 1 ciphertext transmission
Li et.al. [70]	**User**	**Pailler** [49]: c encryptions., $c(1+f)$ decrypti. c multip. **PIR** [75]**:** f invocations **Corresponding argmax:** 1 invocation with the server	**PIR** [75]**:** f invocations **Corresponding argmax:**1 invocation with the server
	Server	**Pailler** [49]: c encryptions, $c(1 + \Sigma\|Fi\|)$ multip. **PIR** [75] **:** f invocations **Corresponding argmax:** 1 invocation with the client	**Pailler** [49]: $2c$ cipher transmissions (1024 bit each) **PIR** [75]**:** f invocations
Gao et. al. [71]	**User**	**Pailler** [11]: $2f$ multiplications **Corresp. secure comp.**: 1 invocation with the server **Parallel Oblivious Transfer** [71]**:** 2 invocations	**Corresp. secure comp.:** 1 invocation with the server **Parallel Oblivious Transfer** [71]**:** 2 invocations
	Server	**Pailler** [11]: $c(1 + \Sigma\|Fi\|)$ encryptions **Corresponding secure comparison** with the client **Parallel Oblivious Transfer** [71]**:** 2 invocations	**Corresp. secure comp.:** 1 invocation with the server **Parallel Oblivious Transfer** [71]**:** 2 invocations
Kjamilji etal[72]	**User**	**FV** [56]: 2 encryptions + 1 addition **Corresponding argmax** with the server	**FV** [56]: 2 ciphertexts **Corresponding argmax** with the server
	Server	**FV** [56]: c (1 plain multiplication + 1 addition + $log(\Sigma\|Fi\|)$ multiplications) **Corresponding argmax** with the server	**Corresponding argmax** with the server
Our scheme	**User**	**Corresponding argmax** with the server	**FV** [56]: 1 ciphertext **Corresponding argmax** with the server
	Server	**FV** [56]: 3 mulip., $log(\Sigma\|Fi\|) + 2$ additions, 1 decrypt. **Corresponding argmax:** 1 invocation with the client	**FV** [56]: 1 ciphertext **Corresponding argmax:** 1 invocation with the client

For maximum performances we will have to pack those outputs into one ciphertext, thus we need extra $T_{pack} = (f-1)(\text{rotation} + \text{addition})$. Assuming that the T and S tables are also packed correspondingly into $\left\lceil \frac{N}{b \cdot u \cdot c \cdot \sum_i F_i} \right\rceil$ ciphertexts, we have $T_{rem} = \left\lceil \frac{N}{b \cdot u \cdot c \cdot \sum_i F_i} \right\rceil \left(|F_j| \text{ multiplications} + |F_j| \text{ additions}\right) + (log_{1.5}(f+1) + \lceil logq + 2 \rceil)$ multiplications for the remaining part, where the term $(log_{1.5}(f+1) + \lceil logb + 2 \rceil)$ comes from the *WallaceTree* and *K-S adder*, b is the number of bits per encrypted integer and u is the number of slots between two neighboring bits. Thus for the server-centric classification in [69], in total we need $T_{tot} = TX_{query} + T_{pack} + T_{rem}$ of BGV-like operations and the corresponding secure argmax scheme, which is the value of [69] provided in Table 6.3.

Table 6.4 shows the homomorphic complexity and circuit depth (the number of consecutive multiplications) of some of our secure linear operations compared with the best reported results of the related research schemes in [89-90], since they are known to be among the best. All of our algorithms have a $O(logd)$ (logarithmic) complexity wrt. to the matrix (vector) dimensions, except for *secMatMat* which is linear since it uses *secDRep*(·), which is linear itself.

Table 6.4. Complexity and comparisons of secure linear algebra operations

Algorithm	**ADD**	**CMUL**	**ROT**	**MUL**	**DEPTH**
secSum	*logd*	1	*logd*	0	1 CMUL
CRep	*logd*	0	*logd*	0	0
secDRep	*2logd*	*d*	*d+logd*	0	2 CMUL
secDotP	*logd*	1	*logd*	1	1 MUL+1 CMUL
secMatVec	*2logd*	1	*2logd*	1	0
secMatVec (C)*	***logd***	**2**	***logd***	**0**	**2 CMUL**
secMatVec [89]	*d*	*d*	*d-1*	0	2 CMUL
secMatMat	***4logd***	***d + 1***	***d+3logd***	**1**	**1 MUL+2 CMUL**
secMatMat [90]	*6d*	*4d*	$3d+5\sqrt{d}$	d	1 MUL+2 CMUL

*For benchmark purposes with [89] the vector is packed and in plain
**it is assumed that the matrixes are squared. All logs are in base 2
***ADD = Ciphertext Addition, CMUL = Constant (plain) Multiplication, ROT = Rotation, MUL = Ciphertext Multiplication

6.9.2. Experimental evaluations and comparisons

Table 6.5 gives the computation cost to securely sum up d integers. Due to the SIMD packing of integers into polynomials with size N, in our scheme we can simultaneously sum up $\frac{N}{d}$ sets of d integers, so all of the results for our scheme are aggregated (divided byN/d) to include this speed-up. Table 6.6 shows the results for secure comparison of two integers, and for the same reasons the results of our scheme are aggregated (divided by N) to include N simultaneous comparisons. For our scheme in Table 6.6 for the costs of our *secComp* algorithm we consider party A to actually execute our *secComp* algorithm and party B to decrypt it. Table 6.7 results of our block are aggregated to include simultaneous secure argmax of N/c pairs of c integers. Table 6.8 and 6.9 give the results for different polynomial sizes of $SRCPer,\ invSRCPer$ and $CPack,$ among different block sizes – k, and number of ciphertexts to pack – p, respectively.

Table 6.5. Computation cost for secure sum (*secSum*) of d integers among different schemes (in milliseconds)

scheme / d	**Our scheme - *secSum* (Algorithm 6.1)**			Liu et. al. [67]	Park et. al. [69]	Khedr et. al. [74]	Bost et.al.[53]
	N = 4096	*N* = 8192	*N*= 16384				
32	**0.093826**	**0.138688**	**0.443816**	256	≈106000	192	168
64	**0.192583**	**0.336359**	**1.05018**	512	≈202000	384	336
128	**0.399319**	**0.768558**	**2.48382**	1024	≈778000	768	672
256	**0.910869**	**1.763741**	**5.648328**	2048	NA	1536	1344
512	**1.992125**	**3.984806**	**12.65613**	4096	NA	3072	2688
1024	**4.2552**	**8.57455**	**28.14**	8192	NA	6144	5376

Table 6.6. Computation cost for the SIMD secure comparison (*secComp*) protocol of two encrypted integers among different schemes

Scheme	**Our scheme (*secComp*) (in milliseconds)**						[53], [70] [71] (in ms.)		[67] (in sec.)		[69] (in sec.)		[73] (in ms.)	
	***N* = 4096**		***N* = 8192**		***N* = 16384**									
Place	**A**	**B**	**A**	**B**	**A**	**B**	A	B	A	B	A	B	A	B
Cost	**0.33**	**0.24**	**0.72**	**0.42**	**2.03**	**0.69**	45.3	43.7	2.5	NR	8	NR	110	NR

*NR = Not reported

Table 6.7. Computation cost of secure argmax (*secArgmax*) of *c* integers among different schemes (in milliseconds, unless otherwise stated)

Sch.	**Our scheme – *secArgmax* (Algorithm 6.7)**						[53] [75]		[69]	[73]
	***N* = 4096**		***N* = 8192**		***N* = 16384**					
Place / ***c***	A	B	A	B	A	B	A	B	All	All
4	**0.01**	**0.01**	**0.0267**	**0.0132**	**0.09**	**0.02**	≈250	≈150	≈20 s	440 sec.
8	**0.11**	**0.02**	**0.2321**	**0.0250**	**0.76**	**0.03**	≈550	≈400	≈120 s	880 sec.
16	**0.32**	**0.04**	**0.6609**	**0.0544**	**2.25**	**0.07**	≈1100	≈800	≈900 s	1760 sec.
32	**0.98**	**0.08**	**2.0322**	**0.1115**	**6.99**	**0.16**	≈250	≈150	≈20 s	3520 sec.
64	**3.51**	**0.21**	**7.2732**	**0.2059**	**26.6**	**0.31**	≈550	≈400	≈120 s	7040 s

Table 6.8. Comparison of computational costs for *SRCPer* and *invSRCPER* for different block) sizes *k* and polynomial modulus *N* (results are in milliseconds)

k / ***N***	**2**	**3**	**4**	**5**	**6**	**7**
4096	8.66	15.07	21.18	25.05	31.71	36.86
8192	45.82	55.28	83.06	101.97	130.49	156.87
16384	239.09	377.19	566.71	680.13	904.02	997.20

Table 6.9. Comparison of computational costs of *CPack* for different ciphertext numbers *p* and polynomial modulus *N* (results are in milliseconds)

p / ***N***	**2**	**4**	**8**	**16**	**32**	**64**
4096	2.04	5.63	17.82	37.45	209.53	209.53
8192	8.01	23.08	60.87	150.9	887.60	887.60
16384	50.81	138.8	426.6	1024.56	6752.1	6852.1

Tables 6.10 and 6.11 give the computation cost of some of our secure linear algebra operations compared with the best known results from the state-of-the-art schemes.

Table 6.10. Comparisons of amortized secure linear algebra operation costs (in ms)

Dimension	4	8	16	32	64
secSum	10^{-4}	0.005	0.011	0.045	0.081
CRep	0.006	0.035	0.21	1.625	6.01
secDRep	0.15	3.906	80.4	352.5	450
secDotP	$9 \cdot 10^{-4}$	0.024	0.04	0.107	0.194
secMatVec;	0.05	0.24	1.19	5.2	22.6
secMatMat	**0.54**	**9.6**	**145**	**575**	**895**
secMatMat (*HE-MatMult*) *of* [90]	3	NR	162	NR	10^4

Table 6.11. Comparisons of amortized cost of *secMatVec* costs (in ms)

Matrix dim.	Naive [89]	Diagonal [89]	Hybrid [89]	Our*
1024 × 128	880.0	192.4	**16.2**	30.8
1024 × 16	110.3	192.4	7.8	**3.79**
128 × 16	77.4	25.4	5.3	**0.48**

* multiplication of a plaintext matrix with a packed ciphertext vector

Table 6.12 compares the best PP classification results per query as reported at the corresponding schemes. In Table 6.10 scheme we report our results for Algorithm 6.21 for datasets in [84-87], also introduced in Chapter 5.5.

Table 6.12. Per query comparison of the cumulative (among all participants) costs for the PP classification case among different schemes and datasets

Scheme	[73]	[69]	[53]	[67]	[71]	[75]	[72]	[76]	[77]	[78]	[79]	**Our**	[66]	**Our**	**Our**
Dataset	Breast Cancer Wisconsin (Original) Data Set [86]												Acute Inflammations Data Set [87]		SMS Spam [84]
Comp. cost	48 ms	70 sec.	479 ms.	349. min.	555 ms.	few min.	14 ms	35.7 ms	1.5 sec	0.62 sec.	0.40 sec.	**0.84 ms**	196 sec.	**0.9 ms**	**6.75 ms**
Comm. cost	NR	2 cip her.	72.9 KB	1.24 MB	19.3 KB	NR	109 KB	43.1 KB	4 MB	256 KB	306 KB	**13 KB**	40 KB	**13 KB**	**109 KB**

*NR = Not reported

Table 6.13. Amortized query costs for the NB classifier based on linear algebra operations

Cost / Scheme	Computation cost	Communication cost
Original Wisconsin Breast Cancer Dataset [86]		
[53]	479 milliseconds	72.5 KB
[10]	555 milliseconds	19.3 KB
[6]	2900 milliseconds	800 KB
[44]	349 minutes	1.24 MB
[73]	48.79 milliseconds	Not reported
[69]	70 seconds	2 ciphertexts
[72]	14 milliseconds	109 KB
[76]	35.74 milliseconds	43.13 KB
Our	**2.37 milliseconds**	**13.7 KB**

Table 6.14. Amortized query costs for the SVM classifier based on linear algebra operations

Cost / Scheme	Computation cost	Communication cost
Original Wisconsin Breast Cancer Dataset [86]		
[53]	204 milliseconds	35.84 KB
[44]	3100 milliseconds	7.5 MB
[73]	2.41 milliseconds	Not reported
[97]	3.47 milliseconds	0.92 KB
Our	**0.19 milliseconds**	**0.43 KB**
UNSW-NB 15 cybersecurity dataset [94]		
Our	**0.20 milliseconds**	**0.43 KB**

Table 6.15. Amortized query costs for the LR classifier based on linear algebra operations

Cost / Scheme	Computation cost	Communication cost
Original Wisconsin Breast Cancer Dataset [86]		
[53]	204 milliseconds	35.84 KB
[97]	3.55 milliseconds	0.92 KB
Our	**0.21 milliseconds**	**0.43 KB**
UNSW-NB 15 cybersecurity dataset [94]		
Our	**0.19 milliseconds**	**0.43 KB**

Tables 6.13-6.15 compare the amortized (per query) computation and communication costs of our schemes based on secure linear algebra operations (Algorithm 6.26) and the related PP classifications for NB, SVM and LR among different datasets, respectively. One of the datasets is the Wisconsin Breast Cancer dataset [86], while the other is the UNSW-NB 15

cybersecurity dataset [94], which is also a binary-class dataset, thus $c = 2$, has $f = 42$ original features, but after extensive feature selection in plain we used only $f = 15$ of them. For both datasets our PP classification scheme (Algorithm 6.26) showed no loss of accuracy due to PP classification. If a certain scheme provides several results for the same purpose due to different security parameters or improved scenarios, in Tables 6.13-6.15 we give the best results of the corresponding schemes. In all of them for our scheme we provide the implementation results of the improved version of the server centric *secMLClass* (Algorithm 6.26). For all of the datasets the trained model for NB in plain was obtained using C++ code, while for SVM and LR the trained models were obtained by WEKA [93].

Table 6.16. Amortized per query costs for (*secC*, Algorithm 6.24)

Comput. cost (ms)		**Commun. cost (KB)**		**Comput. cost (ms)**		**Commun. cost (KB)**	
EC	**E2DS**	**EC**	**E2DS**	**EC**	**E2DS**	**EC**	**E2DS**
Enron Email dataset [37], *N*=8192				Enron Email dataset [37], *N*=16384			
15.03	0.28	27.5	0	17.3	0.27	27.5	0
SMS spam dataset [84], *N*=8192				SMS spam dataset [84], *N*=16384			
6.5	0.2	13.75	0	6.95	0.72	13.75	0

Table 6.16 gives the amortized (per query) costs of the improved 2PC version of Algorithm 6.24 for different N. In this sense, for the Enron dataset [37], already presented in Chapters 4.5 and 5.5, the number of packed queries in a single ciphertext is $q = \left\lfloor \frac{N}{(m+1)} \right\rfloor = \left\lfloor \frac{N}{(2047+1)} \right\rfloor =$ 4 and 8 queries for $N = 8192$ and $N = 16384$, respectively, while for the SMS spam corpus dataset [84] it is 8 and 16 queries, respectively.

In Table 6.17 we report and compare the costs and characteristics of several related schemes (mainly related to binary textual datasets) dealing with PP classification. Since they report several costs and properties, we present the best of each one of each scheme.

Table 6.17. Amortized per query costs for PP classifications among different binary textual datasets (queries)

Scheme	**Comp. cost**	**Comm. cost**	**ML algorithm**	**Class. Acc.**
Enron email dataset [37]				
[11]	≈8 s	Not reported	NB	Not report.
[12]	350 ms	≈110 KB	MNB,NB	98.8%
[17]*	3.79s (SEAL)	40.63 MB	Deep Learning	86.3%
[17]**	0.17s (GPU)	40.63 MB	Deep Learning	86.3%
[21]	78 min	Not reported	NB	99.1%
Ours	**15.31 ms**	**27.5 KB**	MNB	**99.1%**
SMS spam corpus dataset [84]				
[20]	21 ms	Not reported	NB	**95.6%**
[18]	6.75 ms	109 KB	NB	93.1%
Ours	**3.38 ms**	**11.37 KB**	NB	93.1%
Hate speech against immigrants and women in Twitter dataset [19]				
[19]	25.579 s	Not reported	Ensemble trees	74.4%
[19]	0.953 s	Not reported	Logis. Regress.	72.4%

*26 cores of 2.1 GHz Intel Xeon Platinum processor with 188 GB of RAM
** 1 TESLA (5120 cores of 1.38 GHz) and 3 P100 (3584 cores of 1.19 GHz)

We evaluate the performances of the proposed secure comparison-*secComp* protocol over arithmetic circuits when it is used isolated (not in combination with other building blocks or protocols), which is given as $y = (a - b)r + h = xr$ +h, s.t. $r > 0$ and $|h| < r$ (Section 6.3.4). We assume that variables a and b are samples from a uniform distribution in the range of $(-2^{n-1}, 2^{n-1} - 1)$, thus $a \leftarrow A = U(-2^{n-1}, 2^{n-1} - 1)$,, $b \leftarrow B = U(-2^{n-1}, 2^{n-1} - 1)$, r is a positive sample from a discrete Gaussian distribution with mean 2^{n-1} and standard deviation of 3.2, thus $r \leftarrow R = N(2^{n-1}, 3.2)$, while h is a sample from uniform distribution in the range $(-r + 1, r - 1)$, $h \leftarrow H = U(-r + 1, r - 1)$, where n is the number of bits that the variables have. The distribution type of *r,* its mean and dispersion where chosen due to showing better experimental performances and were inspired by LWE. For the distribution of the variable x we have $x \leftarrow X = A - B$.

In Fig.6.23-6.25 we show joint probability of X and Y -$P(X, Y) -$ by plotting 10.000 points when the numbers of bits are n=2,3 and 4, respectively.

What we want to idelly see in Fig.4-6 is a projection of *Y* which is uniform and a projection of *X* which is the same for each value of *Y*. In this case, observing any value of *Y* will give the conditional entorpy for *X*, thus there would be no information gain for any observerd value of Y. However, this is not the case with Fig.4-6, thus there is some information leakage about the difference of numbers a and b.

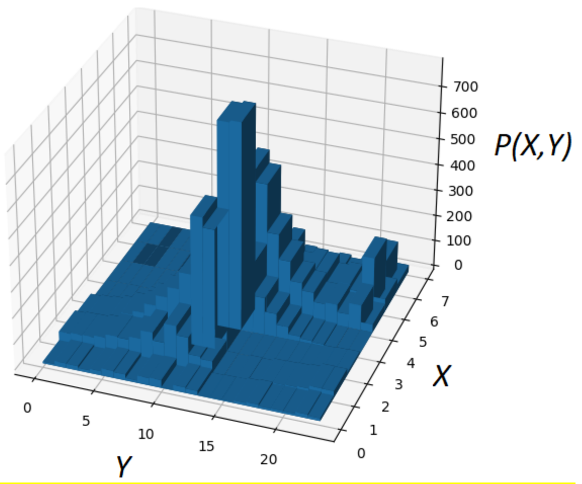


Fig.6.23 The joint probability of X and Y, $P(X,Y)$, for n=2 bits

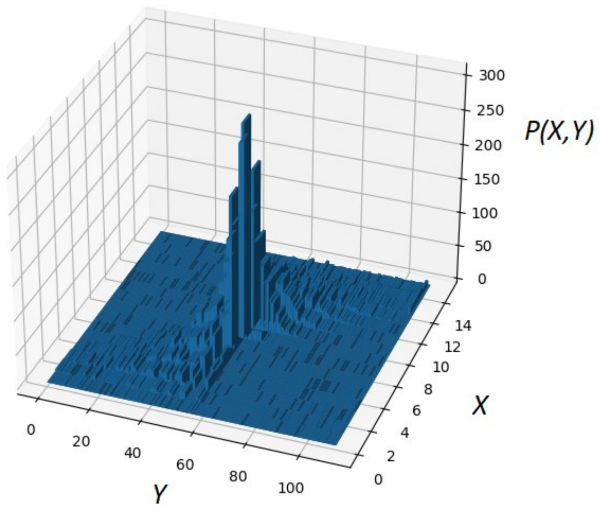


Fig.6.24. The joint probability of X and Y, $P(X,Y)$, for n=3 bits

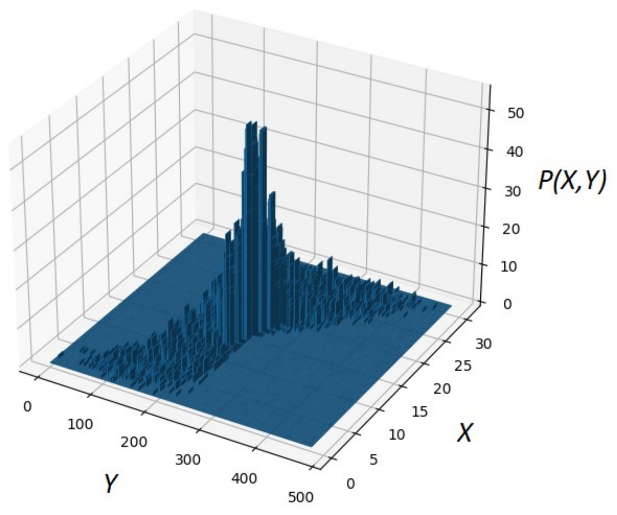


Fig.6.25. The joint probability of X and Y, $P(X,Y)$, for n=4 bits

In Fig.6.26-6.28 we show the probability of Y- $P(Y)$, by plotting 10.000 points when the numbers of bits are n=8,12 and 16 bits, respectively.

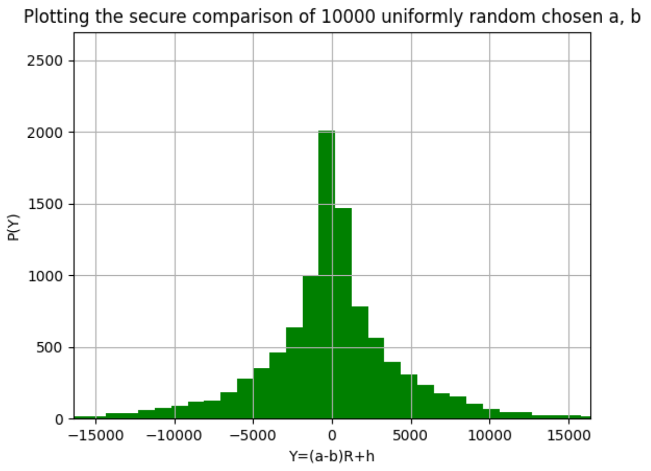


Fig.6.26.Plotting 10.000 points to draw P(Y) when n=8 bits

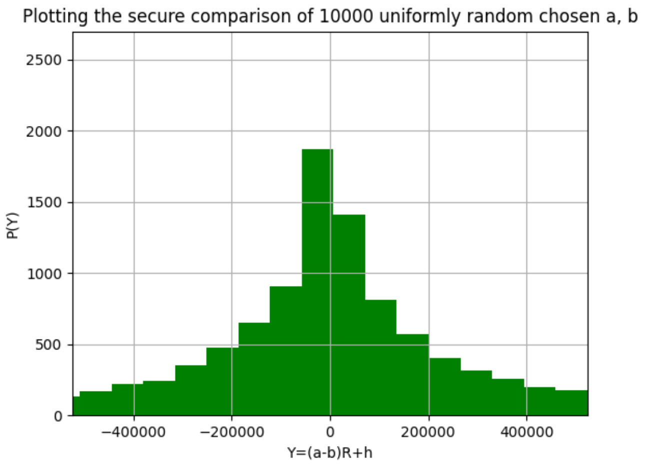


Fig.6.27.Plotting 10.000 points to draw P(Y) when *n*=12 bits

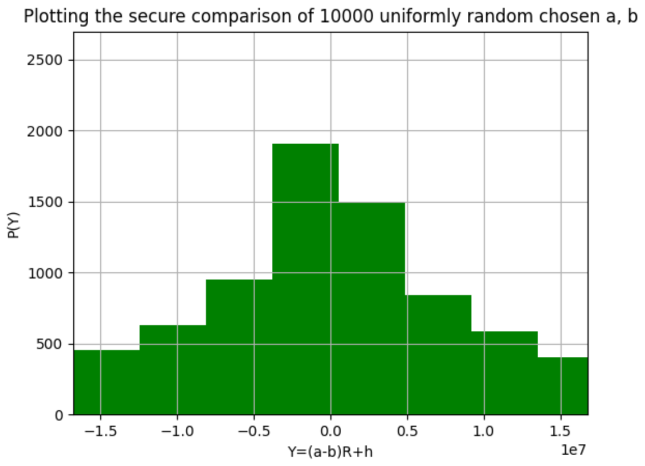


Fig.6.28.Plotting 10.000 points to draw P(Y) when *n*=16 bits

Fig.6.26-6.28 show that the security characteristics of Y improve as the number of bits gets larger by making the dispersion of Y greater. This is especialy good knowing that Algorithm 1 can be expressed with the Paillier that can have thousands of bits and the SWHE scheme which can offer close to one thousand bits when $N = 32K$.

6.10. Security analysis and proofs

Theorem 6.1: Let $a \leftarrow U_q^{n\times m}$ and let a' be obtained by a s.t. each of the rows of a is subtracted in index (component-wise) manner with all the other rows (in total we have $\frac{n(n-1)}{2}$ such subtractions, thus a' has $\frac{n(n-1)}{2}$ rows). Let $s' \leftarrow U_q^{m\times 1}$, $R \leftarrow U_q^{\frac{n(n-1)}{2}\times 1}$ and $h \leftarrow X_q^{\frac{n(n-1)}{2}\times 1}$, s.t. each element (entry, index) of R is greater than zero, thus *R>0* and $|h| < R$, thus the absolute value of each element of h is smaller than the corresponding index value of *R*. Let $c' \leftarrow U_q^{n\times 1}$. If Decision-LWE is hard (Section 2.4.2), then distinguishing between $(a', a's' \times R + h)$ and (a', c') is also hard. Here $\times$ stand for index (component-wise) multiplication.

Proof: We get a' in polynomial time of operations (subtractions). Apparently $a' \epsilon\, U_q^{\frac{n(n-1)}{2}\times n}$, since subtracting a uniform variable from another uniform variable in modular arithmetic still gives a random uniform variable. The same reasoning can be given for the multiplication of two uniform random variables in modular arithmetic. Simply put, before adding h to $a's'$, we furthermore randomized all of $a's'$ components (indexes) by multiplying them with a random uniform *R,* thus $a's' \times R + h \cong U_q^{\frac{n(n-1)}{2}\times n}$. This means that we can't distinguish between $(a', a's' \times R + h)$ and (a', c'). ■

Theorem 6.2 (Symmetry of Decision-LWE): Let $s \leftarrow U_q^{m\times 1}$, $a \leftarrow U_q^{n\times m}$, $e \leftarrow \chi_q^{n\times 1}$, $c \leftarrow U_q^{n\times 1}$. If Decision-LWE is hard**,** then distinguishing between $(s, a \cdot s + e)$ and (*s, c*) is also hard. Thus, here, instead of *a*, we share *s*.

Proof: if there is a way to find $a\, \epsilon\, U_q^{n\times m}$ from $s\, \epsilon\, U_q^{m\times 1}$, then it is even easier to find s when a is given, which contradicts Assumption 1 (Decision-LWE). ■

Theorem 6.3: Using SCADS (Algorithm 2) after secure ML classifications (equation (1)), reveals nothing about the trained model *M* or the user query *X*.

Proof: Let $X \leftarrow U_q^{m\times 1}$, $M \leftarrow U_q^{n\times m}$, $R \leftarrow U_q^{\frac{n(n-1)}{2}\times 1}$ and $h \leftarrow X_q^{\frac{n(n-1)}{2}\times 1}$, where R and h are the random integer vectors needed for SCADS (hence for secure comparison) in SIMD fashion. At the end of SCADS the output will be $M'X \times R + h$, where $M' \epsilon\, U_q^{\frac{n(n-1)}{2}\times n}$. We invoke Theorem 1 to show that the user query *X* is kept private. We invoke Theorem 2 to show that the trained model *M* is also kept private. ■

Theorem 6.4: Let $s \leftarrow U_q^{m\times 1}$, $a \leftarrow U_q^{n\times m}$, $h \leftarrow \chi_q^{n\times 1}$ $R' \leftarrow \chi_q^{n\times 1}$, s.t. each element (entry, index) of *R* is greater than zero, thus *R>0* and $|h| < R$, thus the absolute value of each element of *h* is smaller than the corresponding index value of *R*. *Let* $c \leftarrow U_q^{n\times 1}$. If Decision–LWE holds, then we also can't distinguish between $(a, (a \cdot s)^2 R' + h)$ and (*a, c*).
Proof: We invoke Theorem 1, where *R* is $(a \cdot s)R'$. ■

Theorem 6.5: Let $s_0 \leftarrow U_q^{m\times 1}$, $a_i \leftarrow U_q^{n\times m}$, where $0 \leq i \leq l$, $h \leftarrow \chi_q^{n\times 1}$ $R' \leftarrow \chi_q^{n\times 1}$, s.t. each element (entry, index) of *R* is greater than zero, thus *R>0* and $|h| < R$, thus the absolute value of each element of *h* is smaller than the corresponding index value of *R*. If Decision–LWE holds, then we can't distinguish between $(a_l, (a_l \cdot s_l)^2 R' + h)$ and (a_l, *c*). Here $s_i = (a_{l-1} \cdot s_{l-1})^2$.
Proof: We will use Mathematical induction. For $i = 0$ we invoke Theorem 3. Let's assume that the theorem holds for $i = k$. Then for $i = k + 1$ we have $s_{k+1} = (a_k \cdot s_k)^2$, which is is also uniformly random, thus $(a, (a_{k+1} \cdot s_{k+1})^2 R' + h)$ and (*a, c*) can't be distinguished according to Theorem 4.

Theorem 6.6: Using SCADS (Algorithm 2) after secure ML classifications over DNN (equation (2)), doesn't reveal anything about the trained model *M* of the DNN or the user query *X*
Proof: If the DNN has one layer, then $X \leftarrow U_q^{m\times 1}$, $M^0 \leftarrow U_q^{n\times m}$, $h \leftarrow \chi_q^{n\times 1}$ $R \leftarrow \chi_q^{n\times 1}$ and the output of the DNN will be $(M \cdot X)^2 R' + h$. We invoke Theorem 4 and Theorem 2 to proof that there is no leakage of M or X.
If the DNN has more l layers, then $X^0 \leftarrow U_q^{m\times 1}$ $M^i \leftarrow U_q^{n\times m}$, for $0 \leq i \leq l$, $h \leftarrow \chi_q^{n\times 1}$ $R \leftarrow \chi_q^{n\times 1}$. and $X^i = (X^i \cdot M^i)^2$. The output of the DNN at the l-th layes is $(M^l \cdot X^l)^2 R + h$. We invoke Theorem 5 and Theorem 2 to proof that there is no leakage of the trained model *M* and the user query *X*

Theorem 6.7: *secArgmax* (Algorithm 6.7) is a secure two-party protocol under the semi-honest model.

Proof: Here f is the deterministic function $argmax(Pk, input\_c, sk)$, where the public key Pk and the input ciphertext $input_c$are the private inputs of party A and the secret key sk is the private input of party B. For the output of the function we have $argmax(Pk, input\_c, sk) = argmax_A(Pk, input\_c, sk), argmax_B(Pk, input\_c, sk)) = (\phi, maxIndex\_v)$, where ϕ means no output. The protocol Π that securely computes $argmax$ is $secArgmax(Pk, input\_c, sk)$. For the output of the protocol Π we have $output^{\Pi}(Pk, input\_c, sk) = (output_A{}^{\Pi}, output_B{}^{\Pi}) = (\phi, maxIndex\_v)$.

The view of party A is $V_A = (Pk, input\_c, r_A, rndMaxIndex\_c)$, where Pk is the public key and r_A are random coin tosses at A. We build the simulator for party A as $S_A\big((x_A), f_A(\bar{x})\big) = S_A\big((Pk, input\_c),\ secArgmax_A(Pk, input\_c, sk)\big) =$ $S_A((Pk, input\_c), \phi) = (Pk, input\_c, \widetilde{r_A}, rndM\widetilde{axIn}dex\_c)$, where $\widetilde{r_A}$ is chosen from the same distribution as r_A and $rndM\widetilde{axIn}dex\_c$ is a random ciphertext. Due to the semantic security of the RLWE scheme an adversary cannot distinguish between $rndMaxIndex\_c$ and $rndM\widetilde{axIn}dex\_c$, hence $S_A \cong_C V_A$.

Similarly, for the view of party B we have $V_B = (sk, r_B, perSCADS\_c, result\_c)$, where sk is the secret key and r_B are random coin tosses at B. We construct the simulator for the party B as $S_B\big(x_B, f_B(\bar{x})\big) = S_B\big((sk), secArgmax_B(Pk, input\_c, sk)\big) = S_B(sk, maxIndex) =$ $(sk, \widetilde{r_B}, per\widetilde{SCAD}S\_c, re\widetilde{sul}t\_c)$, where $\widetilde{r_B}$ has same random distribution as r_B, $per\widetilde{SCAD}S\_c$ is a random ciphertext and $re\widetilde{sul}t\_c$ is done by first constructing a vector $maxIndex\_v$ from the output $maxIndex$ and then having $re\widetilde{sul}t\_c = Encode\_Encrypt(maxIndex\_v)$. Due to the semantic security of the ciphertexts, an adversary cannot distinguish between $perSCADS\_c$ and $per\widetilde{SCAD}S\_c$, as well as between $result\_c$ and $re\widetilde{sul}t\_c$, so $S_B \cong_C V_B$. ■

Theorem 6.8: *PPClassServCen* (Algorithm 6.21) is a secure multi-party protocol under the semi-honest model

Proof: The function $f\big((X, k, R), TM\_c, \phi\big) = (f_{User1}, f_{TACS}, f_{EDS}) = (C_{TM}(X), \phi, \phi)$, is computed by the protocol *PPClassServCen* = Π, which we split into two protocols called consecutively. Namely ρ_1 computes lines 1-12 and ρ_2 computes lines 13-26, so

$PPClassServCen = \Pi = \rho_1\rho_2$. For the corresponding protocol outputs we have $output^{\Pi}(X, k, R, TM\_c, \phi) = (output^{\Pi}_{User1}, output^{\Pi}_{TACS}, output^{\Pi}_{EDS}) = (C_{TM}(X), \phi, \phi)$.
$output^{\rho_1}\big((X, k, R), \phi, \phi\big) = \big(output^{\rho_1}_{User1}, output^{\rho_1}_{TACS}, output^{\rho_1}_{EDS}\big) =$
$(\phi, (k, R, queryVetor\_c), \phi)$,
$output^{\rho_2}((k, R), (k, R, queryVetor\_c,\ TM\_c), \phi) =$
$\big(output^{\rho_2}_{User1}, output^{\rho_2}_{TACS}, output^{\rho_2}_{EDS}\big) = (C_{TM}(X), \phi, \phi)$.
For the views and simulators for ρ_1 we have the trivial ones, thus $V^{\rho_1}_{User1} = (X, r_U) \cong_C (X, \widetilde{r_U}) = S^{\rho_1}_{User1}(X, \phi)$, $V^{\rho_1}_{TACS} = (TM\_c, r_{TACS}) \cong_C (TM\_c, \widetilde{r_{TACS}}) = S^{\rho_1}_{TACS}\big(TM\_c, output^{\rho_1}_{TACS}\big)$.
For ρ_2 for TACS we have the trivial view and simulator. For EDS we have $V^{\rho_2}_{EDS} = (k, R, r_{EDS}, result\_c) \cong_C \big(k, R, \widetilde{r_{EDS}}, \widetilde{result\_c}\big) = S^{\rho_2}_{EDS}$, where for $\widetilde{result\_c}$ we first construct a random vector rnd_v, then encode and encrypt it $rnd\_c = EncodeEncrypt(rnd\_v)$ and set $\widetilde{result\_c} = SCADS(rnd\_c, ([c], n_c, c))$.For the user's view and corresponding simulator we have $V^{\rho_2}_{User1} = (\phi, r_{User}, rndMaxIndex) \cong_C S^{\rho_2}_{User} = (\phi, \widetilde{r_{User}}, ind)$, and $ind = SRCPer(C_{TM}(X), k, R, c, n_c)$. We invoke Theorem 3.1 to prove that $PPClassServCen = \rho_1\rho_2$ is secure ■

Corollary 6.1: protocol ρ_1 is a secure protocol under the semi-honest model ■

Corollary 6.2: protocol ρ_2 is a secure protocol under the semi-honest model ■

Theorem 6.9: *secMLClass* (Algorithm 6.26) is a secure 2PC protocol under the semi-honest model.
Proof: The client's view is $V_C\{\lambda, S\} = V_C = \{rndC_M(S)\_v\}$, where $\lambda = \{N, q, t\}$ (as in Section III-B). Let $rnd\widetilde{C_M(S)}\_v = SRCPer(C_M(S)\_v\ , k, R_1, R_2, c, f + 1)$, where the random parameters k, R_1, R_2 have the same values that the client used while executing Algorithm 6.26. Apparently $rnd\widetilde{C_M(S)}\_v$ and $rndC_M(S)\_v$ are the same. Let the simulator view for the client be $S_C(\lambda, O) = \big\{rnd\widetilde{C_M(S)}\_v\big\}$, thus $S_C(\lambda, O) \cong_c V_C$.

Server's view is $V_S\{\lambda, S\} = V_S = \{rndC_M(S)\_c\}$. For the server's simulator $S_s(\lambda, O)$ we construct a matrix of random queries $\tilde{S} = \{\widetilde{X^{(i)}}\}_{i=1}^{q}$ and use it as our input to proceed with lines 1-7 of Algorithm 6.26 to get $rnd\widetilde{C_M(S)}\_c$. Since the server cannot distinguish between$rndC_M(S)_c$ and $rnd\widetilde{C_M(S)}\_c$ due to the semantic security of the underlying RLWE scheme (he can't even distinguish the genuity between the decrypted and decoded $rndC_M(S)\_v$ and $rnd\widetilde{C_M(S)}\_v$), by having $S_s(\lambda, O) = \{rnd\widetilde{C_M(S)}\_c\}$ we proof $S_S(\lambda, O) \cong_c V_S$ ■

Theorem 6.10: *secC* (Algorithm 6.24) is secure under the semi-honest model.

Proof: EC's view is $V_{EC}^{secC}(\lambda, \bar{x}) = \{rndC_{TM}(q)\_v, h2\_v\}$ its private input $x_{EC}^{secC} = q\_v$ and output $O_{EC}^{secC}(\lambda, \bar{x}) = C_{TM}(q)$. For the EC's simulator we construct random $rnd\widetilde{C_{TM}}(q)\_v, \widetilde{h2\_v}$ s.t. their subtraction will give the same output when EC executes lines 14-18, thus $S_{EC}^{secC}\left(\lambda, x_{EC}^{secC}, O_{EC}^{secC}(\lambda, \bar{x})\right) = \{rnd\widetilde{C_{TM}(q)\_v}, \widetilde{h2\_v}\} \cong_c V_{EC}^{secC}(\lambda, \bar{x})$. The views of TEAS and E2DS are $V_{TEAS}^{secC}(\lambda, \bar{x}) = \{C_{TM}(q)\_c\}$ and is $V_{E2DS}^{secC}(\lambda, \bar{x}) = \{rndC_{TM}(q)\_c\}$, respectively. We construct random ciphertexts $C_{\widetilde{TM}(q)}\_c$ and $rnd\widetilde{C_{TM}}(q)\_c$ for their corresponding simulators. ■

Theorem 6.11: *MU-PPClassServCen* (Algorithm 6.28) is a secure multi-party protocol under the semi-honest model

Proof: *MU-PPClassServCen* will not change if we execute lines 13-17 after line 27 at TACS. Let $\rho_1{}'$ be a protocol that computes lines 1-10 of *MU-PPClassServCen*. By applying the same reasoning as in protocol ρ_1(Corollary 6.1) we deduce that it is a secure protocol under the semi-honest model. Lines 11-12 and line 18 are the *secSum* protocol which we proved to be secure (theorem 3). Let ρ_2' be the protocol that computes lines 19-27 and lines 13-17 (when putting them after line 27) which can be seen as a deterministic function. For the views and simulators of EDS in ρ_2' we have $V_{EDS}^{\rho_2'} = (sumResult\_c, r_{EDS}) = S_{EDS}^{\rho_2'}$. For the view of TACS we have $V_{TACS}^{\rho_2'} = (MU - TM\_c, MU - QueryVector\_c, k, R, r_{TACS}, outcome)$. For the simulator we have $S_{TACS}^{\rho_2'} = (MU - TM\_c, MU - QueryVector\_c, k, R, r_{TACS}, \widetilde{outcome})$, where we set $outcome = false$ if we abort in line

26, otherwise it's true. Line 28 executes protocol ρ_2 which was proven to be secure in Corollary 6.2.

If *MU-PPClassServCen* is substituted by sequential calls to protocols $\rho_1{}'$, *secSum*, ρ_2' and ρ_2, then by invoking the Modular Sequential Composition Theorem (Theorem 3.1) we proof that *MU-PPClassServCen* is secure under the semi-honest model. In the process we used the techniques (ideas) mentioned in Chapter 7 of [60] of forcing the correct behavior of malicious models (users) by using protocols under the semi-honest model. In other words, we apply protocols of the semi-honest model to detect cheatings (misbehaviors) of the malicious models, and if so we abort the protocol, otherwise we execute the protocol till the end ■

Theorem 6.12: *secMLClass-MU* (Algorithm 6.29) is a secure multi-party protocol under the semi-honest model

Proof: Let ρ denote the protocol that executes the first 4 lines of *secMLClass-MU.* Proving ρ is trivial, while in Theorem 6.3 we proved the security of *secMLClass*. Since *secMLClass-MU* is executed by sequentially calling ρ and *secMLClass,* we invoke Theorem 3.1 to proof *secMLClass-MU*'s security under the semi-honest model ■

Chapter 7

CONCLUSIONS

In this dissertation, initially, we provide a novel secure feature selection scheme of homomorphically evaluating features' information gains (a variant of information theoretic entropy) over distributed multi-label multi-output datasets in edge IoT environments. We proceed with secure training and classification of multi-label multi-output datasets over the selected features on the same environment settings (context). Since multi-label multi-output datasets in itself incorporate the special cases of single label multi-class and binary classes datasets, our schemes are valid for them as well. While doing so we take into consideration the heterogeneity (in terms of hardware and software platforms) and the restricted resources that are characteristic for edge IoT devices. We formally prove the security of all of our schemes (protocols, algorithms) under the semi-honest model. In the process, our participants interact with each other under strict security. privacy and efficiency requirements. To these ends, we provide confidentiality, integrity and authenticity to each interaction by signing their hashed contents with the corresponding participant's private key. We assure the consistency among interactions by introducing timestamps and linking them with the hashed content(s) of the preceding interaction(s). This makes our protocols a natural fit for blockchain technology. Our underlying cryptographic tools are proven to be resistant to quantum computer attacks, making our protocols applicable to the post quantum World. All of our protocols (secure feature selection, training and classification) are independent from each other, in terms that, according to the scenario and needs, each of them can be used solely or in combination with secure and private protocols from other research schemes. Our protocols show no loss of classification (prediction) accuracy due to applying ML algorithms in private and secure fashion. Also, they show high rate of fault tolerance (byzantine failures) and resistance to collusion attacks among dataset owners during the secure feature selection and secure training stages.

Our secure feature selection protocols satisfy several strict security and privacy goals by not only keeping private feature values and intermediate results while executing the protocols, rather they keep private the features (or words) themselves as well as the final output (which is the top *m* selected features). Extensive experimental evaluations show that our protocols outperform the state of the art in terms of computation and communication costs for dozen times. In the process the state-of-the art schemes operate under weaker privacy and security constraints. Compared to our protocols, they also suffer from a high level of interactions between the participants. In this sense, for textual datasets they need hundreds of thousands of interaction between the participants, compared to only a few ones needed in our protocols.

We transfer the security, privacy and efficiency properties of secure feature selection protocols to our secure training protocols as well. Namely, during our secure training protocols, besides the feature values, we also keep private the features themselves, the intermediate results while running the protocols as well as the final trained model. This makes our secure training protocols among the rare schemes to do so in literature. Our theoretical analysis and extensive experimental evaluations over benchmark datasets show that our schemes outperform the state-of-the-art in terms of computation and communication costs from several times to orders of magnitudes, not only when state-of-the-art schemes proceed to securely train their ML models without prior feature selection, rather it is the case when they also do it. In this sense, while it takes few minutes to our secure schemes to obtain the final ML trained model over raw datasets (secure feature selection proceeded by secure training), the state-of-the-art schemes do the same for several days or weeks. Besides, state-of-the-art schemes operate under weaker security and privacy requirements, while many of them suffer from high levels of interaction between participants

For the purposes of our secure classification protocol, we propose several novel secure building blocks for general purpose (which are commonly needed for secure ML classifications), as well as building blocks related to secure linear algebra. Our theoretical analysis and experimental evaluations show that our proposed blocks outperform the state-of-the-art in terms of computation and communication costs. Since our secure classification protocols are based on the proposed building blocks, our further theoretical analysis and extensive experimental evaluations over benchmark datasets show that our secure classification protocols outperform the state-of-the-art ones in terms of computation and communication costs, sometimes from several times to orders of magnitude. These results were observed for secure ML classifiers such as Deep Neural Networks, Naïve Bayes, Multinomial Naïve Bayes, Support Vector Machines, Logistic Regression, Decision Trees, Random Forests and K Nearest Neighbors. Similar to the security and privacy properties of the above mentioned secure protocols, during our secure classifications protocols the owner of the trained ML model learns nothing about the users queries, their final classifications or the intermediate results, while the users learn only their respective final classifications and nothing else. All of these goals are achieved by our protocols while operating in a non-interactive fashion (in a single round only). This makes our protocols among the rare ones to achieve those security, privacy and efficiency requirements under those circumstances, since the ones that do so usually suffer from high computation or communication cost. Furthermore, we extend the efficiency of our schemes to also deal with malicious users (which arbitrarily deviate from the protocol with the aim of illegally retrieving any information for the trained ML model or at least with the aim of sabotaging the protocol) during secure NB classifications as well as during multi-users (multi-query) scenarios. To the best of our knowledge, this makes our schemes among the rare (if not the only ones) to address malicious users during secure classifications.

We should note that the experimental evaluations of our secure comparison protocol based on arithmetic circuits showed that when it is used solely (isolated, as a single entity) it doesn't provide a perfect privacy for the difference of the two numbers that it compares. Since SCADS (secure comparison of all data slots) is based on it, SCADS by default inherits the privacy properties of the secure comparison protocol. However, when those two are used in combination with our secure classification protocols, we proof that in polynomial time they can be theoretically reduced to well established cryptographic problems assumed to be hard even for quantum computers, such as LWE. This is due to the fact that our secure classification protocols in their initial phase have the form of matrix-vector multiplication (the trained model multiplied by the user query), proceeded by the proposed secure comparison protocol which adds some noise (random value), which is also the case with the construction of LWE schemes.

We plan to extend these security, privacy and efficiency characteristics of our schemes to deal with other secure multi-label multi-output ML algorithms in distributed environments. E.g., knowing that decision trees use the features' information gains to choose the nodes for each tree levels, one such ML algorithm can be federated (distributed) tree learning, for which we can adjust our proposed protocol of securely evaluating the information gains in distributed environments (*secFS-S2*). Other such secure ML algorithms can be secure distributed training of SVM models or secure kNN over multiple edge IoT dataset owners. Also, they should deal with both horizontally and vertically partitioned datasets.

One of the less explored areas is PP graph theory, especially over multiple graphs, which we also plan to address in near future. It can be used for secure routing, for different companies that use graph theory to represent data in their businesses (such as internet service providers, cargo companies, etc.) to securely aggregate their data with other companies, for Google Maps-like applications to hide the user query to the server while showing the user only the best path (route) and hiding the other paths, etc.

BIBLIOGRAPHY

[1] Nordrum, Amy. "Popular internet of things forecast of 50 billion devices by 2020 is outdated." IEEE spectrum 18 (2016).

[2] Reinsel, David, John Gantz, and John Rydning. "Data age 2025: The evolution of data to life-critical." Don’t Focus on Big Data (2017).

[3] Shokri, Reza, and Vitaly Shmatikov. "Privacy-preserving deep learning." Proceedings of the 22nd ACM SIGSAC conference on computer and communications security. ACM, 2015.

[4] Lindell, Yehuda, and Benny Pinkas. "Privacy preserving data mining." Annual International Cryptology Conference. Springer, Berlin, Heidelberg, 2000.

[5] Agrawal, Rakesh, and Ramakrishnan Srikant. "Privacy-preserving data mining." Proceedings of the 2000 ACM SIGMOD international conference on Management of data. 2000.

[6] Verykios, V. S., Bertino, E., Fovino, I. N., Provenza, L. P., Saygin, Y., & Theodoridis, Y. (2004). State-of-the-art in privacy preserving data mining. ACM Sigmod Record, 33(1), 50-57.

[7] Aggarwal, Charu C., and S. Yu Philip. "A general survey of privacy-preserving data mining models and algorithms." Privacy-preserving data mining. Springer, Boston, MA, 2008. 11-52.

[8] Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K., Naehrig, M., & Wernsing, J. "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy." International Conference on Machine Learning. PMLR, 2016.

[9] Bost, Raphael, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser "Machine learning classification over encrypted data." Network & Distributed System Security Symposium. Vol. 4324. 2015.

[10] Gao, Chong-zhi, Qiong Cheng, Pei He, Willy Susilo, and Jin Li "Privacy-preserving Naive Bayes classifiers secure against the substitution-then-comparison attack." Information Sciences 444 (2018): 72-88.

[11] Khedr, Alhassan, Glenn Gulak, and Vinod Vaikuntanathan. "SHIELD: scalable homomorphic implementation of encrypted data-classifiers." IEEE Transactions on Computers 65.9 (2015): 2848-2858.

[12] Gupta, Trinabh, Henrique Fingler, Lorenzo Alvisi, and Michael Walfish. "Pretzel: Email encryption and provider-supplied functions are compatible." Proceedings of the Conference of the ACM Special Interest Group on Data Communication. 2017.

[13] Kjamilji, Artrim, Arben Idrizi, Shkurte Luma-Osmani, and Ferihane Zenuni-Kjamilji "Secure Naïve Bayes classification without loss of accuracy with application to breast cancer prediction." Proceeding International Conference on Science and Engineering. Vol. 3. 2020.

[14] Wu, David J., Tony Feng, Michael Naehrig, and Kristin Lauter "Privately evaluating decision trees and random forests." Proceedings on Privacy Enhancing Technologies 2016.4 (2016): 335-355.

[15] Liu, Ximeng, Rongxing Lu, Jianfeng Ma, Le Chen, and Baodong Qin "Privacy-preserving patient-centric clinical decision support system on naive Bayesian classification." IEEE journal of biomedical and health informatics 2012 (2015): 655-668.

[16] Liu, Ximeng, Robert Deng, Kim-Kwang Raymond Choo, and Yang Yang "Privacy-preserving outsourced clinical decision support system in the cloud." IEEE Transactions on Services Computing (2017).

[17] Al Badawi, Ahmad, Louie Hoang, Chan Fook Mun, Kim Laine, and Khin Mi Mi Aung "Privft: Private and fast text classification with homomorphic encryption." IEEE Access 8 (2020): 226544-226556.

[18] Kjamilji, Artrim, Erkay Savaş, and Albert Levi. "Efficient Secure Building Blocks With Application to Privacy Preserving Machine Learning Algorithms." IEEE Access 9: 8324-8353.

[19] Reich, Devin, Ariel Todoki, Rafael Dowsley, Martine De Cock, and Anderson CA Nascimento "Privacy-preserving classification of personal text messages with secure multi-party computation: An application to hate-speech detection." arXiv preprint arXiv:1906.02325 (2019).

[20] Resende, Amanda, Davis Railsback, Rafael Dowsley, Anderson CA Nascimento, and Diego F. Aranha "Fast privacy-preserving text classification based on secure multiparty computation." arXiv preprint arXiv:2101.07365 (2021).

[21] Costantino, Gianpiero, Antonio La Marra, Fabio Martinelli, Andrea Saracino, and Mina Sheikhalishahi. "Privacy-preserving text mining as a service." 2017 IEEE Symposium on Computers and Communications (ISCC). IEEE, 2017.

[22] Das, Kamalika, Kanishka Bhaduri, and Hillol Kargupta. "A local asynchronous distributed privacy preserving feature selection algorithm for large peer-to-peer networks." Knowledge and information systems 24.3 (2010): 341-367.

[23] Banerjee, Madhushri, and Sumit Chakravarty. "Privacy preserving feature selection for distributed data using virtual dimension." Proceedings of the 20th ACM international conference on Information and knowledge management. 2011.

[24] Jafer, Yasser, Stan Matwin, and Marina Sokolova. "Privacy-aware filter-based feature selection." 2014 IEEE International Conference on Big Data (Big Data). IEEE, 2014.

[25] Sheikhalishahi, Mina, and Fabio Martinelli. "Privacy-utility feature selection as a privacy mechanism in collaborative data classification." 2017 IEEE 26th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE). IEEE, 2017.

[26] Rahimipour Anaraki, Javad, and Saeed Samet. "Privacy-preserving feature selection: A survey and proposing a new set of protocols." arXiv e-prints (2020): arXiv-2008.

[27] Rao, Vanishree, Yunhui Long, Hoda Eldardiry, Shantanu Rane, Ryan Rossi, and Frank Torres "Secure Two-Party Feature Selection." arXiv preprint arXiv:1901.00832 (2019).

[28] Li, Xiling, Rafael Dowsley, and Martine De Cock. "Privacy-preserving feature selection with secure multiparty computation." arXiv preprint arXiv:2102.03517 (2021).

[29] Merkle, Ralph C. "A digital signature based on a conventional encryption function." Conference on the theory and application of cryptographic techniques. Springer, Berlin, Heidelberg, 1987.

[30] Bolt, Wilko. "Bitcoin and cryptocurrency technologies: A comprehensive introduction." (2017): 647-649.

[31] Li, Shancang, Li Da Xu, and Shanshan Zhao. "The internet of things: a survey." Information Systems Frontiers 17.2 (2015): 243-259.

[32] Shi, Weisong, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. "Edge computing: Vision and challenges." IEEE internet of things journal 3, no. 5 (2016): 637-646.

[33] Androutsopoulos, Ion, Georgios Paliouras, and Eirinaios Michelakis. Learning to filter unsolicited commercial e-mail. NCSR “Demokritos” Technical Report, No. 2004/2, March 2004

[34] Tang, Bo, Steven Kay, and Haibo He. "Toward optimal feature selection in naive Bayes for text categorization." IEEE Transactions on Knowledge and Data Engineering 28.9 (2016): 2508-2521.

[35] Ranaweera, Pasika, Anca Delia Jurcut, and Madhusanka Liyanage. "Survey on Multi-Access Edge Computing Security and Privacy." IEEE Communications Surveys & Tutorials (2021).

[36] Zhao, Wenbing, Congfeng Jiang, Honghao Gao, Shunkun Yang, and Xiong Luo. "Blockchain-Enabled Cyber-Physical Systems: A Review." IEEE Internet of Things Journal (2020).

[37] Metsis, Vangelis, Ion Androutsopoulos, and Georgios Paliouras. "Spam filtering with naive bayes-which naive bayes?." CEAS. Vol. 17. 2006.

[38] Buczak, Anna L., and Erhan Guven. "A survey of data mining and machine learning methods for cyber security intrusion detection." IEEE Communications Surveys & Tutorials 18.2 (2016): 1153-1176.

[39] Dua, Sumeet, and Xian Du. Data mining and machine learning in cybersecurity. CRC press, 2016.

[40] Quinlan, J. Ross. "Induction of decision trees." Machine learning 1.1 (1986): 81-106.

[41] Quinlan, J. Ross. "Bagging, boosting, and C4. 5." AAAI'96: Proceedings of the thirteenth national conference on Artificial intelligence - Volume 1 August 1996 Pages 725–730.

[42] Russell, Stuart, Peter Norvig, and Artificial Intelligence. "A modern approach." Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs 25 (1995): 27.

[43] Mitchell, Tom M. "Machine learning." (1997).

[44] Li, Tong, Zhengan Huang, Ping Li, Zheli Liu, and Chunfu Jia "Outsourced privacy-preserving classification service over encrypted data." Journal of Network and Computer Applications 106 (2018): 100-110.

[45] Mourao-Miranda, J., Reinders, A.A.T.S., Rocha-Rego, V., Lappin, J., Rondina, J., Morgan, C., Morgan, K.D., Fearon, P., Jones, P.B., Doody, G.A. and Murray, R.M

"Individualized prediction of illness course at the first psychotic episode: a support vector machine MRI study." Psychological medicine 42.5 (2012): 1037-1047.

[46] Schmidhuber, Jürgen. "Deep learning in neural networks: An overview." Neural networks 61 (2015): 85-117.

[47] Yao, Andrew C. "Protocols for secure computations." Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on. IEEE, 1982.

[48] ElGamal, Taher. "A public key cryptosystem and a signature scheme based on discrete logarithms." IEEE transactions on information theory 31.4 (1985): 469-472.

[49] Paillier, Pascal. "Public-key cryptosystems based on composite degree residuosity classes." International Conference on the Theory and Applications of Cryptographic Techniques. Springer, Berlin, Heidelberg, 1999.

[50] Shafi, Goldwasser, Silvio Micali."Probabilistic encryption."Journal of computer and system sciences 28.2 (1984):270-299.

[51] Henecka, Wilko, et al. "TASTY: tool for automating secure two-party computations." Proceedings of the 17th ACM conference on Computer and communications security. ACM, 2010.

[52] Ben-David, Assaf, Noam Nisan, and Benny Pinkas. "FairplayMP: a system for secure multi-party computation." In Proceedings of the 15th ACM conference on Computer and communications security, pp. 257-266. ACM, 2008.

[53] Bost, Raphael, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. "Machine learning classification over encrypted data." In Network & Distributed System Security Symposium, 2015.

[54] C. Gentry, "Fully homomorphic encryption using ideal lattices," in ACM Symposium on Theory of Computing, 2009, pp. 169–178.

[55] Brakerski, Zvika, Craig Gentry, and Vinod Vaikuntanathan. "(Leveled) fully homomorphic encryption without bootstrapping." ACM Transactions on Computation Theory (TOCT) 6, no. 3 (2014): 13.

[56] Fan, Junfeng, and Frederik Vercauteren. "Somewhat Practical Fully Homomorphic Encryption." IACR Cryptology ePrint Archive 2012 (2012): 144.

[57] Smart, Nigel P., and Frederik Vercauteren. "Fully homomorphic SIMD operations." Designs, codes and cryptography 71, no. 1 (2014): 57-81.

[58] Halevi, Shai, and Victor Shoup. "Algorithms in helib." In International Cryptology Conference, CRYPTO, pp. 554-571. Springer, Berlin, Heidelberg, 2014.

[59] Martins, Paulo, Leonel Sousa, and Artur Mariano. "A survey on fully homomorphic encryption: An engineering perspective." ACM Computing Surveys (CSUR) 50.6 (2017): 1-33.

[60] Goldreich, Oded. Foundations of cryptography: volume 2, basic applications. Cambridge university press, 2009.

[61] Kantarcıoglu, Murat, Jaideep Vaidya, and C. Clifton. "Privacy preserving naive bayes classifier for horizontally partitioned data." In IEEE ICDM workshop on privacy preserving data mining, pp. 3-9. 2003.

[62] Vaidya, Jaideep, Murat Kantarcıoğlu, and Chris Clifton. "Privacy-preserving naive bayes classification." The VLDB Journal 17, no. 4 (2008): 879-898.

[63] Lindell, Yehuda, and Benny Pinkas. "Privacy preserving data mining." Annual International Cryptology Conference, CRYPTO. Springer, Berlin, Heidelberg, 2000.

[64] Yang, Zhiqiang, Sheng Zhong, and Rebecca N. Wright. "Privacy-preserving classification of customer data without loss of accuracy." In Proceedings of the 2005 SIAM International Conference on Data Mining, pp. 92-102. Society for Industrial and Applied Mathematics, 2005

[65] Yi, Xun, and Yanchun Zhang. "Privacy-preserving naive Bayes classification on distributed data via semi-trusted mixers." Information systems 34, no. 3 (2009): 371-380.

[66] Liu, Ximeng, Rongxing Lu, Jianfeng Ma, Le Chen, and Baodong Qin. "Privacy-preserving patient-centric clinical decision support system on naive Bayesian classification." IEEE journal of biomedical and health informatics 20, no. 2 (2016): 655-668.

[67] Liu, Ximeng, Robert Deng, Kim-Kwang Raymond Choo, and Yang Yang. "Privacy-Preserving Outsourced Clinical Decision Support System in the Cloud." IEEE Transactions on Services Computing, Volume: 14, Issue: 1 (2021): 222 – 234

[68] Li, Ping, et al. "Privacy-preserving outsourced classification in cloud computing." Cluster Computing 21.1 (2018): 277-286.

[69] Park, Heejin, Pyung Kim, Heeyoul Kim, Ki-Woong Park, and Younho Lee. "Efficient machine learning over encrypted data with non-interactive communication." Computer Standards & Interfaces 58 (2018): 87-108.

[70] Li, Tong, Zhengan Huang, Ping Li, Zheli Liu, and Chunfu Jia "Outsourced privacy-preserving classification service over encrypted data." Journal of Network and Computer Applications 106 (2018): 100-110.

[71] Gao, Chong-zhi, Qiong Cheng, Pei He, Willy Susilo, and Jin Li "Privacy-preserving Naive Bayes classifiers secure against the substitution-then-comparison attack." Information Sciences 444 (2018): 72-88.

[72] Kjamilji, Artrim, Arben Idrizi, Shkurte Luma-Osmani, and Ferihane Zenuni-Kjamilji "Secure Naïve Bayes classification without loss of accuracy with application to breast cancer prediction." Proceeding International Conference on Science and Engineering. Vol. 3. 2020.

[73] Sun, Xiaoqiang, Peng Zhang, Joseph K. Liu, Jianping Yu, and Weixin Xie "Private machine learning classification based on fully homomorphic encryption." IEEE Transactions on Emerging Topics in Computing 8.2 (2018): 352-364.

[74] Khedr, Alhassan, Glenn Gulak, and Vinod Vaikuntanathan. "SHIELD: scalable homomorphic implementation of encrypted data-classifiers." IEEE Transactions on Computers 65.9 (2015): 2848-2858.

[75] Li, Tong, Zhengan Huang, Ping Li, Zheli Liu, and Chunfu Jia. "Outsourced privacy-preserving classification service over encrypted data." Journal of Network and Computer Applications 106 (2018): 100-110.

[76] Pereira, Hilder VL. "Efficient AGCD-based homomorphic encryption for matrix and vector arithmetic." IACR Cryptol. ePrint Arch. 2020 (2020): 491.

[77] Yasumura, Yoshiko, Yu Ishimaki, and Hayato Yamana. "Secure Naïve Bayes Classification Protocol over Encrypted Data Using Fully Homomorphic Encryption." Proceedings of the 21st International Conference on Information Integration and Web-based Applications & Services. 2019.

[78] Wood, Alexander, Vladimir Shpilrain, Kayvan Najarian, Ali Mostashari, and Delaram Kahrobaei "Private-Key Fully Homomorphic Encryption for Private Classification." International Congress on Mathematical Software. Springer, Cham, 2018.

[79] Wood, Alexander, Vladimir Shpilrain, Kayvan Najarian, and Delaram Kahrobaei", Private naive bayes classification of personal biomedical data: Application in cancer data analysis." Computers in biology and medicine 105 (2019): 144-150.

[80] Kim Laine, “Microsoft/SEAL”, last accessed on 01.07.2021 from https://github.com/Microsoft/SEAL

[81] Artrim Kjamilji, “artrimk/secMNB_Email”, last accessed on 01.07.2021 from https://github.com/artrimk/secMNB_Email

[82] Oleander Software, “OleanderSoftware”, last accessed on 01.07.2021 from https://github.com/OleanderSoftware/OleanderStemmingLibrary

[83] Joaquin Vanschoren, “OpenmML Speed Dating”, last accessed on 01.07.2021 from https://www.openml.org/d/40536

[84] Tiago A. Almeida and José María Gómez Hidalgo, “SMS spam collection”, last accessed on 01.07.2021 from http://www.dt.fee.unicamp.br/~tiago/smsspamcollection/

[85] Almeida, Tiago A., José María G. Hidalgo, and Akebo Yamakami. "Contributions to the study of SMS spam filtering: new collection and results." Proceedings of the 11th ACM symposium on Document engineering. ACM, 2011.

[86] Dr. WIlliam H. Wolberg, “Breast Cancer Wisconsin (Original) Data Set”, last accessed on 01.07.2021 from https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(original)

[87] Jacek Czerniak, “Acute Inflammations Data Set”, last accessed on 01.07.2021 from http://archive.ics.uci.edu/ml/datasets/acute+inflammations

[88] Nandakumar, Karthik, Nalini Ratha, Sharath Pankanti, and Shai Halevi "Towards deep neural network training on encrypted data." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops. 2019.

[89] Juvekar, Chiraag, Vinod Vaikuntanathan, and Anantha Chandrakasan. "GAZELLE: A low latency framework for secure neural network inference." 27th USENIX Security Symposium (USENIX Security 18). 2018.

[90] Jiang, Xiaoqian, Miran Kim, Kristin Lauter, and Yongsoo Song "Secure outsourced matrix computation and application to neural networks." Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. 2018.

[91] Kalchbrenner, Nal, Edward Grefenstette, and Phil Blunsom. "A convolutional neural network for modelling sentences." *arXiv preprint arXiv:1404.2188* (2014).

[92] Wu, David J., et al. " Wu, David J., Tony Feng, Michael Naehrig, and Kristin Lauter." Proceedings on Privacy Enhancing Technologies 2016.4 (2016): 335-355.

[93] University of Waikato, Craig Nevill-Manning, Mark Hall, "Weka", last accessed 01.07.2021 from http://old-www.cms.waikato.ac.nz/~ml/weka/

[94] Cyber Range Lab of UNSW Canberra, "The UNSW-NB15 Dataset", last accessed on 01.07.2021 from https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/

[95] Damgård, Ivan, Martin Geisler, and Mikkel Krøigaard. "Efficient and secure comparison for on-line auctions." Australasian Conference on Information Security and Privacy. Springer, Berlin, Heidelberg, 2007.

[96] Kim, Pyung, Younho Lee, and Hyunsoo Yoon. "Sorting method for fully homomorphic encrypted data using the cryptographic single-instruction multiple-data operation." IEICE Transactions on Communications 99.5 (2016): 1070-1086.

[97] De Cock, Martine, et al. "Efficient and private scoring of decision trees, support vector machines and logistic regression models based on pre-computation." IEEE Transactions on Dependable and Secure Computing 16.2 (2017): 217-230.

[98] Artrimk Kjamilji, "PhD Dissertation codes", last accessed on 01.07.2021 from https://github.com/artrimk.