TÉCNICO LISBOA

Perspectives on the Feasibility and Adoption of Fully Homomorphic Encryption

A Fully Homomorphic Encryption Application on Spam Detection with Machine Learning Models

Adriano Roberto Pinto

Thesis to obtain the Master of Science Degree in

Information Security and Cyberspace Law

Supervisor: Prof. Carlos Nuno da Cruz Ribeiro

Examination Committee

Chairperson: Prof. Carlos Manuel Costa Lourenço Caleiro
Supervisor: Prof. Carlos Nuno da Cruz Ribeiro
Member of the Committee: Prof. Fernando Mira da Silva

October 2024

Acknowledgments

In such an unequal world, generally only privileged people have the opportunity to thrive; this also applies to me. I had the privilege of a stable family, which was only possible thanks to my parents, Edson and Denise. Any achievement in my life, including this thesis, was only attained because of the much they renounced for me.

I shall also thank my supervisor, Prof. Carlos Ribeiro, for his guidance and availability, two close friends for sharing their academic experiences, and my manager for her flexibility and tolerance in my full-time job. I thank Prof. Rui Cruz for providing the template and tips on Texlive, and the Zama support team for the continuous improvements in the Concrete framework.

Agradecimentos

Em um mundo tão desigual, geralmente apenas as pessoas privilegiadas têm a oportunidade de prosperar, isso também se aplica a mim. Pois tive o privilégio de uma família estável, o que só foi possível graças aos meus pais, Edson e Denise. Qualquer conquista na minha vida, incluindo esta tese, só foi alcançada pelo muito que eles renunciaram por mim.

Agradeço também ao meu orientador, Prof. Carlos Ribeiro, pelo apoio e disponibilidade. Sou grato a dois amigos próximos que contribuiram com experiências de sua própria carreira académica, e à minha gestora pela flexibilidade e tolerância no meu trabalho profissional. Agradeço ao Prof. Rui Cruz por fornecer o modelo e dicas sobre Texlive, e à equipa de suporte da Zama pelas melhorias contínuas no framework Concrete.

Abstract

The general adoption of cloud computing resulted in frequent security incidents, including data breaches in both the public and private sectors. Cloud service providers operate without transparency, fail to comply with regulations, and exploit private data for targeted advertising and training artificial intelligence models. The major tech companies have also been found to collaborate with intelligence agencies to illegally surveil individuals and governments by sharing private data, such as email communications. This study analyses the feasibility of Fully Homomorphic Encryption (FHE) as a solution to these security and privacy concerns, focusing on spam detection and email providers as representative subjects in the universe of outsourced computations on users' private data. FHE maintains data privacy by enabling computations while the data is still encrypted, but it demands substantial computing and memory resources. The development of FHE-based applications is complex, requiring advanced knowledge of mathematical and cryptographic concepts. This work assesses the feasibility of FHE through experiments in spam detection by implementing Fully Homomorphic Encryption Spam Detector (FHE-SD), an application using the Concrete-ML libraries, which abstracts the complexity of FHE and simplifies its adoption. The experimental environment is a device with limited hardware resources, chosen to test if FHE can function without specialized hardware. For meaningful results, FHE-SD supports spam detection using machine learning algorithms, which are commonly used for spam detection. Four machine learning models are implemented in FHE-SD, in their FHE and on-clear versions, enabling various metrics and performance comparisons against traditional approaches.

Keywords

Fully Homomorphic Encryption, Machine Learning, Spam Detection, Performance, Privacy, Security

Resumo

A adoção da computação em nuvem resultou em incidentes de segurança, como vazamentos de dados nos setores público e privado. Provedores operam sem transparência, descumprem re-gulamentações e exploram dados privados para publicidade e treinamento de modelos de inteligência artificial. As empresas de tecnologia também colaboram com agências de inteligência para vigiar ilegalmente indivíduos e governos, compartilhando dados privados, como as mensagens de email. Este estudo analisa a viabilidade da Criptografia Completamente Homomórfica (CCH) como solução para essas questões de segurança, com foco na detecção de spam e provedores de email como objetos representativos no universo da computação feita por terceiros em dados privados. A CCH mantém a privacidade ao habilitar computações enquanto os dados estão criptografados, mas exige elevados recursos computacionais. O desenvolvimento de aplicativos baseados em CCH é complexo e requer aplicação de conceitos matemáticos e criptográficos avançados. Este trabalho avalia a viabilidade da CCH por meio de experimentos em detecção de spam implementando o Fully Homomorphic Encryption Spam Detector (FHE-SD), um aplicativo que utiliza a biblioteca Concrete-ML para abstrair a complexidade da CCH e simplificar sua adoção. O ambiente experimental comporta um dispositivo com hardware limitado, escolhido para verificar se a CCH pode funcionar sem hardware especializado. Para melhores resultados, o FHE-SD tem suporte à detecção de spam usando algoritmos de aprendizado de máquina, que são medidas anti-spam comuns. Quatro modelos de aprendizado de máquina foram implementados no FHE-SD, em suas versões CCH e em dados puros, permitindo métricas e comparações de desempenho em relação às abordagens tradicionais.

Palavras Chave

Criptografia Completamente Homomórfica, Aprendizado de Máquina, Detecção de Spam, Desempenho, Privacidade, Segurança

Contents

List of Figures

List of Tables

Acronyms

AVX512	Advanced Vector Extensions 512
AES	Advanced Encryption Standard
AI	Artificial Intelligence
API	Application Programming Interface
AUC	Area Under the ROC Curve
BFV	Brakerski/Fan-Vercauteren
Bagging	Boot-strap Aggregating
BGV	boot-strap aggregating
BV	Brakerski-Vaikuntanathan
CCH	Criptografia Completamente Homomórfica
CM	Client Machine
CML	Concrete-ML
CPU	Central Processing Unit
CKKS	Cheon-Kim-Kim-Song
DARPA	Defense Advanced Research Projects Agency
DM	Developer Machine
DNS	Domain Name Service
Email	Electronic Mail
ESMTP	Extended Simple Mail Transfer Protocol

FE Functional Encryption

FHE Fully Homomorphic Encryption

FHE-SD Fully Homomorphic Encryption-Spam Detector

FP False Positive

FPR False Positive Rate

FN False Negative

FNR False Negative Rate

GLWE General LWE

GPU Graphics Processing Unit

GSW Craig Gentry, Amit Sahai,and Brent Waters

HE Homomorphic Encryption

IP Internet Protocol

INTEL HEXL Intel Homomorphic Encryption Acceleration Library

HTTP Hypertext Transfer Protocol

IETF Internet Engineering Task Force

IMAP Internet Message Access Protocol

IND-CPA Indistinguishability Under Chosen Plaintext Attack

IEC International Electrotechnical Commission

IoT Internet of Things

IP Internet Protocol

ISO International Organization for Standardization

ISP Internet Service Provider

KNN k-Nearest Neighbors

LinearSVC Linear Support Vector Classification

LLM Large Language Model

LWE Learning With Errors

MAA Mail Accessing Agent

MD5 MD5 Message Digest Algorithm

MDA Mail Delivery Agent

MIME Multipurpose Internet Mail Extension

ML Machine Learning

SIMD Single Instruction Multiple Data

NP Nondeterministic Polynomial time

MTA Mail Transfer Agent

MUA Mail User Agent

NVT ASCII Network Virtual Terminal ASCII

NTRU N-th degree Truncated polynomial Ring Units

PHE Partially Homomorphic Encryption

PIR Private Information Retrieve

POP3 Post Office Protocol Version 3

PoS Part of Scheech

PK Public Key

QR Quick-Response

RF Random Forest

RFC Request for Comments

RLWE Ring Learning With Errors

ROC Curve Receiver Operating Characteristic Curve

RSA Rivest-Shamir-Adleman

SK Secret Key

SHA Secure Hash Algorithm

SGX	Software Guard Extensions
SHE	Somewhat Homomorphic Encryption
SM	Server Machine
SMC	Secure Multiparty Computation
SMTP	Simple Mail Transfer Protocol
SIMD	Single Instruction Multiple Data
SVM	Support Vector Machine
SVP	Shortest Vector Problem
TCP	Transmission Control Protocol
TCP/IP	Family or stack of protocols that uses TCP and IP
TFHE	Fully Homomorphic Encryption over the Torus
TLU	Table Lookups
TN	True Negative
TP	True Positive
UDP	User Datagram Protocol
URL	Uniform Resource Locator
VM	Virtual Machine
XGBoost	Extreme Gradient Boosting

1

Introduction

Contents

The general and often compulsory adoption of cloud computing has resulted in numerous security incidents, including data breaches in both public and private sectors, exposing the personal information of millions of individuals. Furthermore, service providers often operate without transparency, disregard regulations, and exploit private data for purposes such as targeted advertising and training Large Language Models (LLM) based services, which has become a highly profitable sector. Additionally, many Electronic Mail (email) providers have been found to cooperate with intelligence agencies for illegal surveillance of individuals and governments. In this work, we focus on email providers and spam detection as subjects to represent the services provided in the cloud or computing outsourced to third-party servers.

Before the internet and email technologies, people managed their physical mailboxes by sorting letters and discarding any unwanted mail. However, with electronic mail, most of the classification is done by email providers through spam detectors or anti-spam filters, which analyze the content of users' emails to detect spam.

The possibility of providers analyzing the content of private messages may seem extraordinary, but it is a fact. Email server administrators can access the content of all stored messages, which are often kept in plaintext files on servers' disks. When encrypted, the secret keys are generally maintained by system administrators themselves [18]. Suppose users want to guarantee their privacy, even in the eyes of email server administrators. In that case, they need to encrypt their messages in advance using tools such as OpenPGP [19], which is not trivial for most users.

Regulations, laws enforcement, and terms of use are necessary but insufficient to prevent unauthorized access and misuse of private data. Nevertheless, with spam detection and email providers as subjects, this work proposes Fully Homomorphic Encryption (FHE) as a solution to those branches of security and privacy concerns. FHE could increase confidence in the future of cloud security by ensuring persistent data encryption on third-party servers while enabling computations on the encrypted data without compromising privacy. However, FHE feasibility and adoption continue to be a subject of study also due to the high computational requirements and implementation complexity, which usually implies advanced mathematics and cryptography principles.

This work reinforces Fully Homomorphic Encryption (FHE) as a solution to security and privacy problems related to spam detection and email providers. FHE could enhance confidence in cloud security by ensuring persistent data encryption on third-party servers while enabling computations of encrypted data without compromising confidentiality.

1.1 Motivation

With the popularization of cloud computing, many applications are no longer executed locally, on-premise, and are delegated to remote servers managed by third-party providers. In the case of email applications,

tasks that were executed locally by Mail User Agents (MUA) can now be assigned to email providers through web interfaces. Popular email services such as Gmail, Yahoo, and Outlook tend to neglect user privacy. For example, they use anti-spam measures as a pretext to access the content of messages for data collection. Gmail's terms of use state that all emails are scanned to identify spam and malware [20]. Google even admits that until 2017, it scanned content to create advertisements, maintain features, and for other purposes [21].

Yahoo goes further than Google by directly declaring that email content can be used for advertising purposes. The following excerpt from Yahoo Mail's terms of use explicitly acknowledges this [22]:

> "... Our anti-spam team may review messages marked as spam or non-spam to improve our spam defenses ... This information may be used in personalized advertisements ..."

Microsoft's Privacy Statement vaguely describes data collection for its Artificial Intelligence (AI) chatbot *"Copilot"*, which is integrated with Microsoft 365 products such as Outlook [23].

The violation of email confidentiality is not limited to the providers alone, it also involves collaboration between technology companies and organizations focused on espionage. The former National Security Agency (NSA) contractor Edward Snowden released thousands of classified files revealing extensive global surveillance programs. The documents included conclusive evidences of cooperation between the NSA and major email providers. Big Tech corporations such as Google, Apple, Meta, Microsoft, and Yahoo collaborated by supplying data from millions of users, governments, organizations, and authorities, including email communications. The Snowden's files revealing this information were compiled and published in several prominent newspapers worldwide, such as The Guardian, The Washington Post, and The Intercept [24–26].

Figure 1.1 shows one of the documents released by Snowden where it is possible to verify the involvement of the Big Techs Microsoft, Yahoo, Google, Facebook, PalTalk, Apple, YouTube, Skype, AOL, and Apple in the NSA's PRISM espionage program. It details the notation utilized to send real-time notifications about private information that could interest the NSA. The first two characters refer to the Yahoo provider, and the third character refers to the type of content. The NSA also used the *Xkeyscore* system to collect in real-time almost all activities of millions of users on the Internet. In the case of Microsoft, the cooperation went even further, as the company implemented a backdoor in Outlook so that the FBI could access the content of emails before they were encrypted in Microsoft servers. In one of the documents provided by Snowden, the NSA praises Microsoft's cooperation with the following sentence [27]:

> "...collaborative teamwork was the key to the successful addition of another provider to the PRISM system."

This level of engagement was not exclusive to Microsoft, Yahoo also implemented software to scan all users' emails for specific keywords supplied by the FBI and the NSA [27].

Those pieces of evidence highlight the pressing need for enhanced security and privacy measures to protect the confidentiality of cloud data from criminals, organizations, and even national agencies. PRISM is not the only case, many others exist, such as the Five, Nine, and Fourteen Eyes alliances. These alliances involve collaborations among countries for surveillance and intelligence gathering, with broader and more complex implications, including industrial espionage [28–30].

Fully Homomorphic Encryption (FHE) features can address all those concerns by allowing third-party servers to perform computations on encrypted data while maintaining continuous secrecy. It also offers additional value through its quantum resistance, which results from the hardness[1] of the underlying problems upon which FHE algorithms are built. As a result, it will not be compromised when quantum computing becomes a reality.

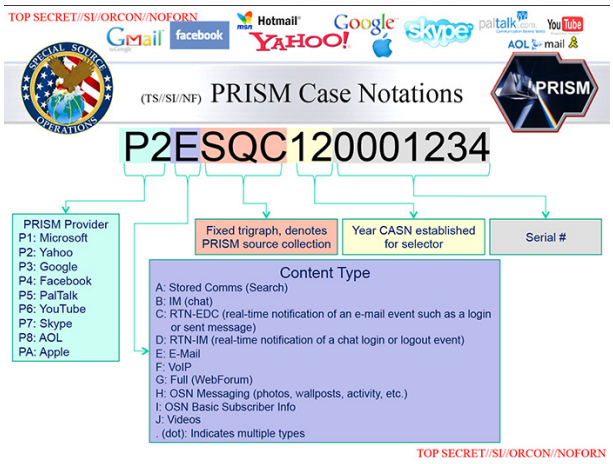


Figure 1.1: PRISM Case Notations

1.2 Contributions

Having Fully Homomorphic Encryption (FHE) as a solution for the given examples of security and privacy challenges, our purpose is to execute a set of experiments to evaluate the feasibility of FHE in spam detection applications. The feasibility is not limited to enabling computation on encrypted data but also includes reducing development efforts and making FHE go beyond the academic research scope to be effectively adopted by other sectors.

We evaluate and demonstrate the feasibility of FHE by developing an application named FHE-SD, a spam detector capable of identifying spam on both FHE encrypted data using Concrete-ML and on clear data using scikit-learn libraries. The FHE-SD supports several metrics based on ML models' prediction results and execution times at different stages, which help analyze and compare how FHE performs

[1] In Computational Complexity, hardness refers to the difficulty of solving a problem, typically characterized by the absence of known polynomial-time algorithms to solve it.

against on-clear counterparts. Four machine learning algorithms were integrated into FHE-SD to train both FHE and on-clear models' versions for practical spam classification. Machine learning is the most successful type of spam detection solution and is a technology that can be heavy on computational resources usage. Those aspects make it an ideal choice to consistently evaluate how FHE performs in demanding contexts and closer to real-world scenarios. FHE-SD implements the entire Machine Learning (ML) workflow, from preprocessing to training and prediction stages. The term *on-clear* will be often used in this thesis, referring to scenarios that do not apply encryption or when working on non-encrypted data or plaintext. FHE-SD utilizes Fully Homomorphic Encryption over the Torus (TFHE) as the underlying FHE scheme that has its complexity abstracted by the Concrete-ML (CML), making unnecessary working directly with the TFHE scheme implementation.

The objective of evaluating the feasibility of FHE can be divided into three sub-objectives.

- Demonstrate the FHE potential to replace traditional solutions by achieving results in machine learning performance metrics, such as accuracy, identical to their on-clear counterparts.
- Demonstrate that libraries and compilers can reduce the effort required to implement FHE-based applications, make the development process possible for non-cryptographers, and consequently facilitate broader adoption of FHE.
- Assess whether hardware acceleration or specialized hardware, such as a Graphics Processing Unit (GPU), is necessary for applications like spam detection.

Research Questions

The objective and sub-objectives can be summarized in the following research questions.

1. Is FHE feasible and possible to adopt in real-world applications such as spam detection?
   - **(a)** Can FHE spam detection solutions achieve identical inference and ML performance results as their on-clear counterparts?
   - **(b)** Can the development process of FHE-based applications be facilitated and made attainable to non-cryptographers?
   - **(c)** Do applications like spam detection require hardware acceleration or specialized hardware? To address this question, we establish that specialized hardware is unnecessary if a single spam email can be classified as spam or ham within 10 seconds. The reason for this threshold is explained in Chapter 5.

1.3 Thesis Outline

This study is structured as follows: Chapter 2 provides a brief overview of the fundamental technologies and background knowledge required to comprehend this work, including networks, machine learn-

ing, spam detection, and cryptography. Chapter 3 presents a brief literature review and related works, comparing their contributions with the contributions of this thesis. Chapter 4 focuses on the proposed solution, describing the development of the FHE-SD application and the experimental methods. Chapter 5 evaluates and discusses the experimental results. Chapter 6 contains the conclusion, outlining the achievements and proposing future directions for research.

2

Background

Contents

This chapter covers a range of disciplines necessary to understand the research methodologies, results, and conclusions within the context of this thesis. Section 2.1 presents the basics of the Internet Protocol Suite (TCP/IP) and email protocols. Section 2.2 and Section 2.3 outline various categories of spam and methods for detection and mitigation. Section 2.4 and Section 2.5 cover the necessary theory to explain the machine learning workflow and techniques used to implement FHE-SD. Section 2.6 and Section 2.7 provide an overview of cryptography and Homomorphic Encryption (HE) fundamentals.

2.1 Internet Protocol Suite

The Internet Protocol Suite, also known as TCP/IP stack, is the set of communication protocols used on the Internet. That collection of protocols enables network communication between computers and other devices on the Internet. The TCP/IP architecture is structured in four layers to better divide and organize functions [31]. In general, the higher layers obtain services from lower layers by transferring information to each other through the interfaces between adjacent layers. Protocol layering has conceptual and structural advantages, allowing a structured way to discuss and update system components [2]. Another widely accepted model is the 7-layered OSI; the comparison can be seen in Figure 2.1 [32]. Each model has its advantages. Since several manufacturers and technologies exist, the protocols define the necessary standards for the network connection between different systems.

This chapter presents just a brief overview of the protocols and network technologies related to email communications. In-depth descriptions can be found in the Request for Comments (RFC) referenced along the text. The RFC are technical documents where protocols and other internet standards are specified. They are authored by volunteers, engineers, and computer scientists and published by the Internet Engineering Task Force (IETF), the main internet standards organization [33].

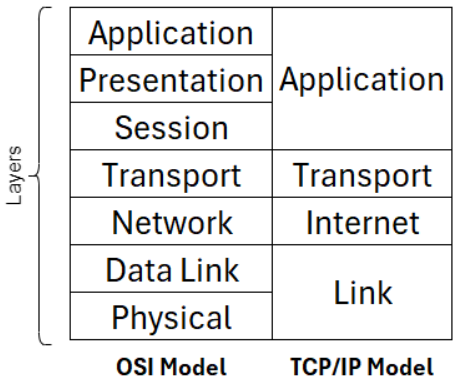


Figure 2.1: OSI and TCP/IP models compared

Email Protocols

The email protocols serve the purpose of transmitting text messages between network computers with the possibility of also sending attached digital files. More rudimentary and limited email versions were first used in the 1960s, but essentially for communication at universities and restricted to the same mainframes. In the 1970s the programmer Ray Tomlinson, at the time linked to the Defense Advanced Research Projects Agency (DARPA) and the ARPANET project, introduced the use of the symbol @ in email addresses in order to separate the user name from the computer domain name, which makes it possible to identify the recipient's host on the network. Sometime later, email communications began to be widely used on local networks and on the internet [34].

To send a message using protocols based on a client-server model, the user connects to an email server through a Mail User Agent (MUA) application client, such as Thunderbird and Outlook. On the email server, the user has their own mailbox, which can be a file directory where messages exchanged by the user are stored and accessed from the MUA [35].

The type of program that archives messages received in users' email boxes is a Mail Delivery Agent (MDA), which is activated by a Mail Transfer Agent (MTA) - the Linux's program Procmail is an example of an MDA. The MTA is responsible for transporting emails between servers via SMTP [36]. Figure 2.2 summarizes this dynamic, and in the following sections we briefly explain how some of the main email protocols in the TCP/IP protocol stack function: SMTP, POP, IMAP, and MIME [1].

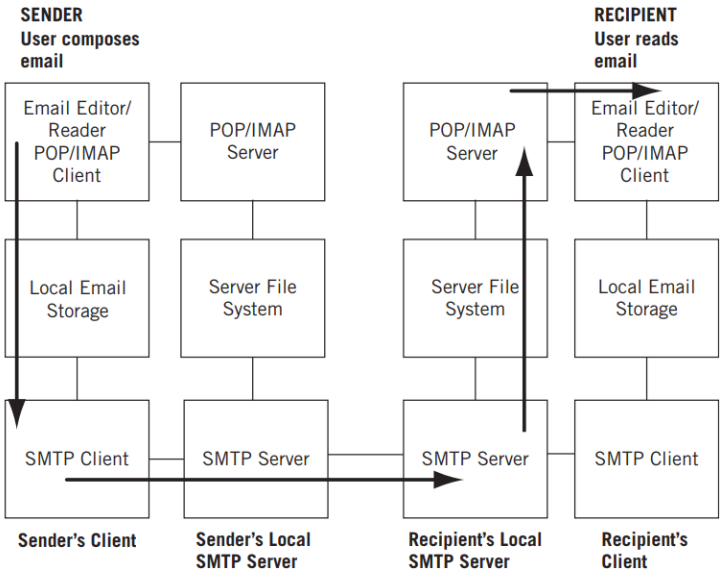


Figure 2.2: Email Protocols [1]

SMTP Protocol

Simple Mail Transfer Protocol (SMTP) [36] is an application layer internet protocol in the OSI model, see Figure 2.5, and the primary protocol for sending and receiving emails in the TCP/IP protocol stack. Therefore it depends on the Transfer Control Protocol (TCP) to make the connections that are made by default through port 25. The SMTP transports a mail object, which is composed of the envelope and the content. The envelope is usually generated based on the elements ”from”, ”to”, ”cc”, and ”bcc” inside the email object, and contains the necessary information to accomplish transmission and delivery. The mail object content is described in Section 2.1.

The original version of SMTP was used only for host-to-host messages through Mail Transfer Agent (MTA) type programs, such as Mail, Sendmail, and Postfix, which are found on Linux systems. However, to send the message, both the sender and the recipient need to be online. This limitation was overcome through dedicated servers, which receive emails and redirect them to recipients when they are online. MTAs continue to be used in the client-server model; they are even responsible for assembling message envelopes through the ”MAIL FROM” and ”RCPT TO” commands in the SMTP protocol [36]. The Figure 2.3 illustrates an example of communication via SMTP [1].

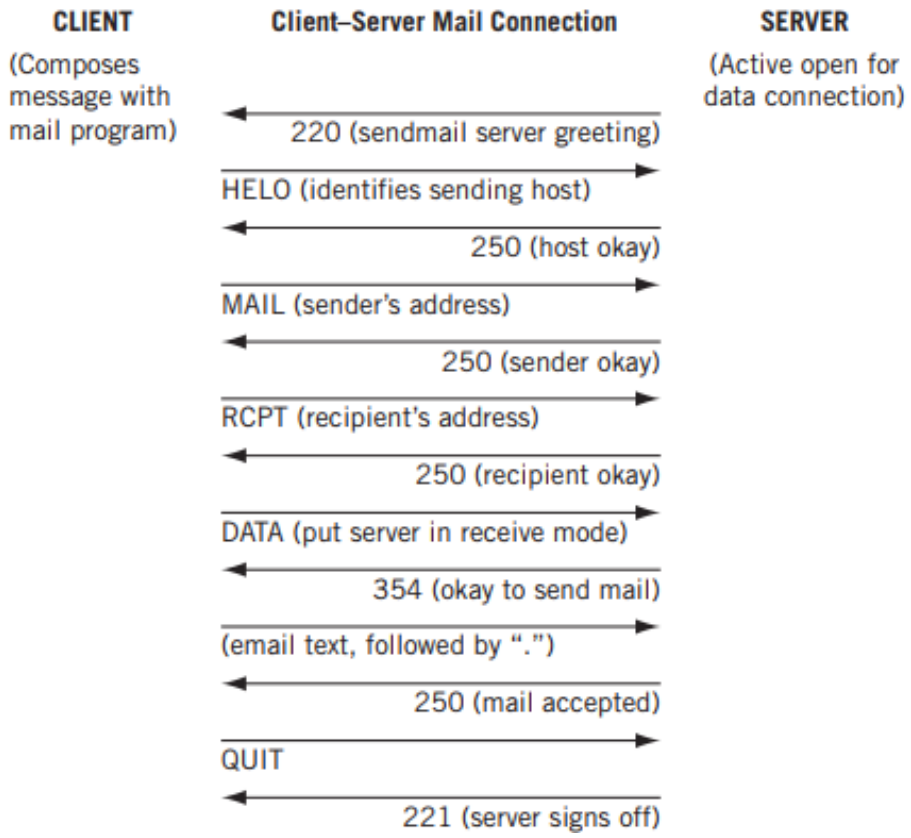


Figure 2.3: SMTP [1]

The following steps and Figure 2.4 exemplify the SMTP operation, where the sender (Alice) wants to send a message to the recipient (Bob) [2]:

1. Alice executes her MUA, enters Bob's email address, composes a message, and instructs the MUA to send the message.
2. Alice's MUA forwards the message to Alice's email server, or email provider, where it is placed in a message queue.
3. Alice's email server, which has an SMTP service configured, selects the message in the queue and opens a TCP connection with Bob's email server.
4. After the SMTP connection is established, the MTA of Alice's SMTP server sends the message via TCP connection.
5. The message arrives at the MTA from Bob's SMTP server and triggers the MDA, which places the message in the email box belonging to Bob.
6. Bob executes his MUA to read the received message. If Bob's email server is temporarily down, the message waits on Alice's email server for a future attempt.

In step (2), the MUA can use both SMTP and HTTP protocols for connection. In (4), communication is done exclusively through SMTP, while in (6) HTTP, IMAP or POP3 can be used. POP3 and IMAP are described in the following section, however HTTP is not discussed because it applies only to the use of webmail clients, usually accessed via internet browser.

It is important to note that the SMTP protocol has limitations that are mitigated through other standards and protocols, such as MIME, discussed in Section 2.1, and Extended Simple Mail Transfer Protocol (ESMTP), which provides the structure for extending and enhancing the original SMTP [37].

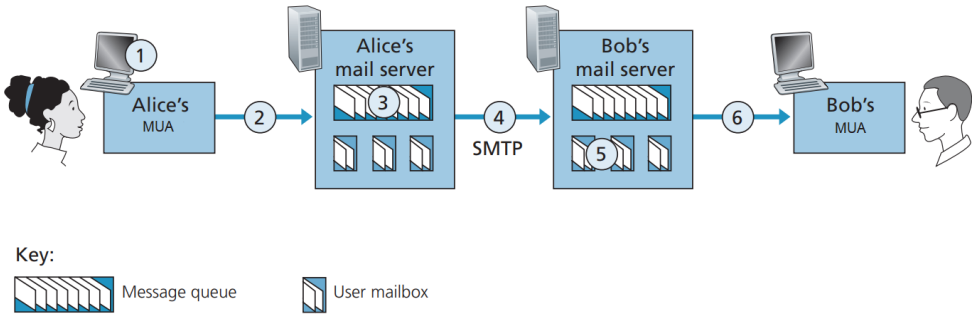


Figure 2.4: Example of sending an email [2]

IMAP and POP3 Protocols

Internet Message Access Protocol (IMAP) [38] and Post Office Protocol Version 3 (POP3) [39] are application layer protocols in the OSI model (Figure 2.5). These types of protocols are also called Message Accessing Agent (MAA), as they serve for message recovery in step (6) of Figure 2.4, maintaining mes-

sage states, searching for messages, and some other basic operations the MUA can make on the emails received.

These protocols are competing, with IMAP being the most used because it facilitates access to emails from different devices, has more advanced functionalities and better synchronization, and does not require space on the local disk. While POP3 is simpler and faster, it depends on downloading all messages to the local device and, by default, deletes the email from the server after downloading. In general, POP3 is useful when the user has their emails stored locally or accesses them offline [2, 35, 40]. Regarding email attachments, adding different types of files to the emails requires specific encoding, which is enabled by the MIME protocol.

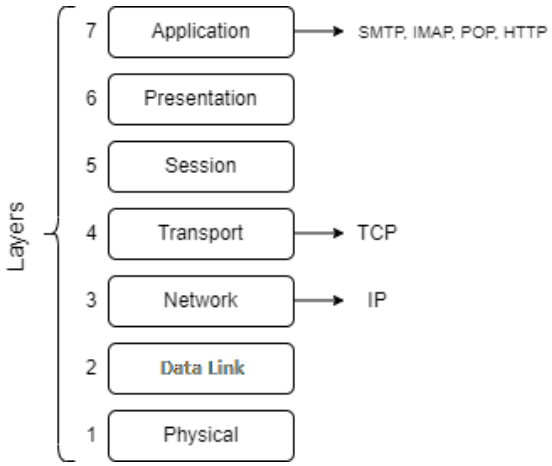


Figure 2.5: ISO/OSI Model

MIME Protocol

The original SMTP protocol only supports ASCII character encoding, so it cannot handle accented characters and other special symbols. This limitation also affects binary files, which would prevent attachments from being sent in emails.

The Multipurpose Internet Mail Extension (MIME) protocol is an internet standard defined in RFC-2045 [41]. It was created to overcome the aforementioned limitations, and its integration with SMTP is described in RFC-1869 [37]. Therefore, MIME is an extension of the SMTP protocol that allows the sending of different types of files attached to emails. The MIME header describes the types of attached files or data, which when sent are converted to 7-bit Network Virtual Terminal ASCII (NVT ASCII) encoding. This encoding is supported by SMTP, transfering the email as described previously in Section 2.1. When it reaches the recipient, it is converted back to the original encoding according to the content types defined in the MIME header.

In summary, MIME is a protocol with rules for transforming data between ASCII and non-ASCII en-

coding, allowing sending attachments and texts in non-ASCII languages, such as Portuguese, French, and Chinese. In Figure 2.6, the MIME header is exemplified in the structure of an email.

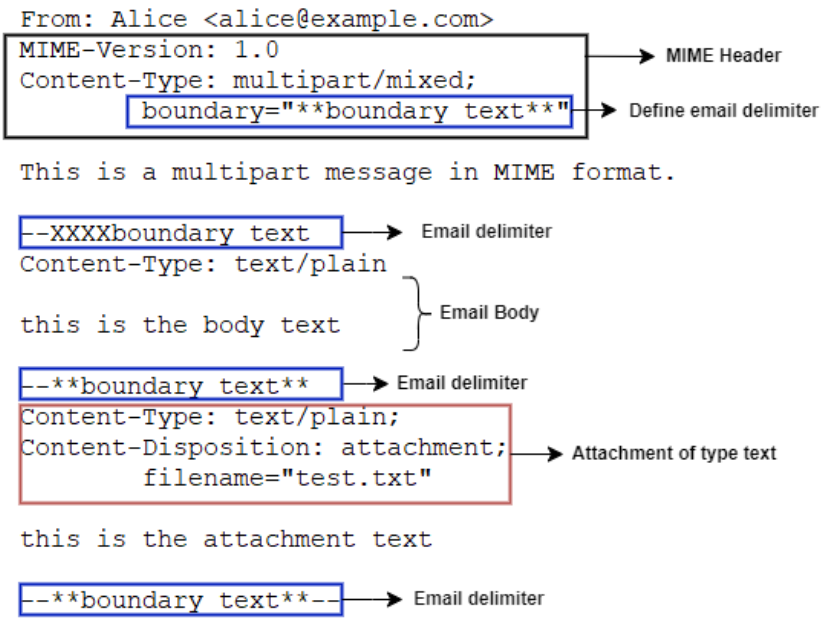


Figure 2.6: MIME Header

Internet Message Format

The message in an email object has a plain text format, using ASCII encoding, and is composed of three parts: an envelope, a header, and a body. The envelope, also discussed in Section 2.1, contains the information necessary for the MTA to process the messages received. The header has one keyword per line, and a colon follows each keyword. Some keywords are optional and can also be utilized by the MTA and MUA. In Figure 2.7, we have an example of an email header [18], followed by the CRLF separator, and then the email body. The keywords ”From:” and ”To:” are mandatory fields corresponding to the sender and recipient respectively [2]. The format of the email message may vary depending on the optional fields displayed and if used in conjunction with other protocols, such as MIME protocol.

2.2 Spam

Spam are unwanted messages sent in electronic format, most commonly email, but can also occur in other media such as mobile text messaging and social networks. The act of sending spam messages is known as *”spamming”*, generally sent on bulks of messages that can be abusive and deceptive, and those that send spam are referred as *”spammers”*. A possible analogy for spam are the pamphlets and

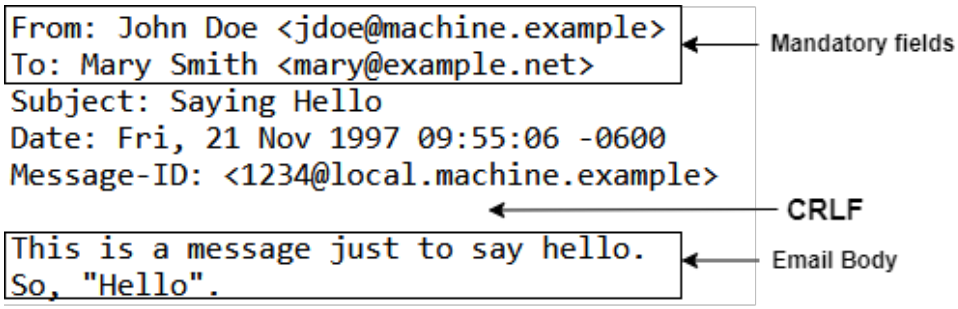


Figure 2.7: Email Header and Body

unaddressed advertising, which we still receive without consent at our homes today. In summary, spam is the propagation of unsolicited messages on various electronic communication platforms.

The term spam was inspired by a product from the company Hormel Food Corporation, a canned meat that appeared in a Monty Python sketch where a restaurant menu has spam in almost every dish, and at some point, a group of vikings starts to repeat the word *"spam"*. In some publications, to refer to the opposite of spam, authors use the term *"ham"*, which is the same as non-spam [42]. The term spam was used for the first time via email, supposedly in 1994, in a *USENET* discussion group [43].

Initially, spam served mainly for marketing and pay-per-click advertising but soon started to be utilized for fraud. Among the very first was the *"419 Fraud"*, where someone presenting themselves as from a foreign country asks for a certain amount of money in order to falsely give the victim access to a more significant sum of money that under certain conditions would be shared for both sides profit [44].

Nowadays, fraudulent techniques using spam continue to evolve. They are used to steal banking and personal data, install viruses, steal passwords, and engage in several other criminal activities. Some of these techniques include phishing, quishing, and scams that utilize artificial intelligence (AI). Phishing is the most common method, where the spam message is disguised as a legitimate email to trick the victim into providing information or clicking on a link that inadvertently installs malware. Quishing is a form of phishing where the user is manipulated into scanning a Quick-Response (QR) code that can compromise the device.

The methods of spreading spam are constantly evolving and becoming more sophisticated. In December 2022, around 45% of email traffic on the internet was spam [45]. In 2022, the introduction of Chat GPT led to increased use of AI in social-engineering attacks, including spam. From the fourth quarter of 2022 to the third quarter of 2023, there was a 1265% increase in malicious phishing messages [46].

The following is a selection of different forms of spam:

- Unauthorized advertising.
- Illegal Commerce.
- Pornography.
- Dissemination of malicious codes.
- Scams, fraud.
- Phishing, quishing.
- Steganography.
- AI based social-engineering.

Among the strategies to combat spam are educational campaigns, anti-virus software, managing network port 25 used by SMTP, and anti-spam filters [47].

2.3 Anti-spam Filters or Spam Detectors

Anti-spam filters are tools to detect and block spam. There are several types and implementations of anti-spam filters. In general, they use algorithms that consider the probability that parts of an email may have spam characteristics. For each identified part is assigned values, if those values exceed a certain threshold, it is marked as spam. Among the main attributes taken into account for spam detection are IP address, domain, content, and related reputation. The following subsections describe some of the common techniques for detecting spam. These methods are often combined to overcome individual deficiencies [48].

2.3.1 Origin-Based Filters

Can be considered the least invasive type of anti-spam filter regarding user privacy. Origin-Based filter detect spam using the email header and network information. Under certain criteria, it scans the header fields, if the sender is on a blocklist, for example, then the email is discarded as spam. The blocklists are types of origin-based filters.

2.3.2 Blocklists

Blocklists are databases widely used as an anti-spam measure. They allow the filtering of SMTP traffic from senders using IP addresses with bad reputations. Blocklist filtering is a very efficient mechanism that does not require inspecting the email content and preserves user privacy. Standard Domain Name Service (DNS) lookups are used to query these databases during an SMTP connection or when email is received.

There are three types of blocklists: blacklist, whitelist, and greylist. Implementing these lists is relatively simple and generally involves techniques that use DNS to identify and link IP addresses and domains. The requirements for each system can vary according to the policies defined by the system administrators. For example, some blocklists are more aggressive and focus on blocking the highest amount of spam, while others are more permissive to avoid false positives [2, 35, 49].

The following itens describe the three types of spam blocklists:

- **Blacklist:** Databases or lists that contain IP addresses and domains known for abuse in sending spam. They are maintained by administrators, providers, and operators, some of which are publicly accessible. Among the situations that can lead a domain to a blacklist is when the IP

address of the sender using the domain changes very frequently and sends a large volume of emails to several recipients. Blacklists require low CPU usage and allow spam to be blocked at SMTP level.

- **Whitelist:** As opposed to blacklist, senders present in this type of database are always considered reliable. Administrators and users maintain these lists, defining the email addresses that are trusted. The contact list in a user's email client can also be considered a form of whitelist, as contact addresses bypass spam detection.
- **Greylist:** This approach evaluates new senders and identifies and blocks spambots, that are programs designed to send spam automatically. In general, legitimate servers make more than one attempt to send an email; if they do not, the email is marked as spam. Greylisting can be inconvenient, as it slows down the sending and receiving process given the additional interactions required [50]. Greylists are also known as challenge-response systems, as some implementations challenge the senders with an automatic reply, requiring them to prove if they are legitimate and not an automated mailer or spam bot [35, 51, 52].

An example of greylisting is presented in Figure 2.8, where [3]:

- *t0* represents the moment of sending the first message from the new sender, its inclusion in the *greylist*, and the temporary blocking of the message.
- *t1* is the time interval for the second sending attempt, if exceeded, the address is removed from the list and future messages restart the process as if they were the first messages and are temporarily blocked again.
- *t2* is a second attempt made within the *t1* interval, the sender's address is added to the list and is blocked during time interval *t2*. After *t2* the address is removed from the list and the next email is considered first message.

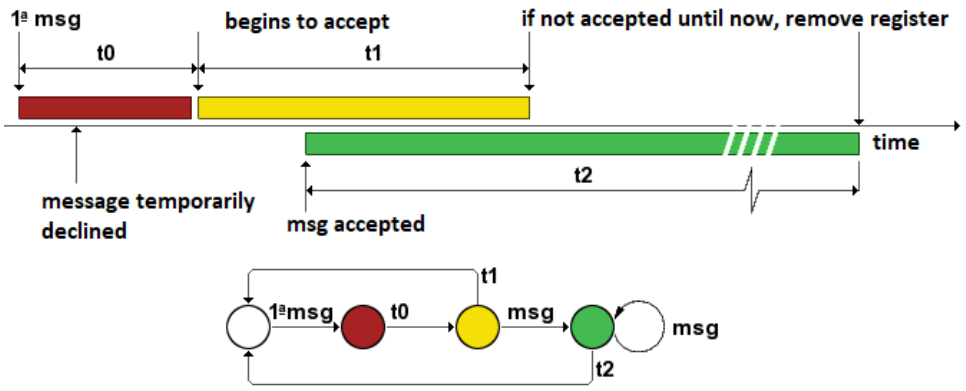


Figure 2.8: Greylisting, adapted from [3]

2.3.3 Heuristic Filters

Heuristic filters are implemented by assigning rules based on the administrator's or user's experience. They can be simple validations, such as the existence of a specific word in the email subject, content, or parts of the sender's address. An example is the *"Sieve Filter"* available in email providers such as Protonmail [53].

2.3.4 Filters by Content

While blocklist filters examine email headers and other network data, content filters operate by scanning the content of all emails, searching for patterns that indicate spam. Various algorithms, such as heuristics, and machine learning-based filters, can be employed to implement these filters. Content filters are widely used and effective, but they can be highly invasive, potentially compromising the users' privacy.

2.3.5 Filters Based on Machine Learning

The increasing computational resource demand for spam detection is a cause for concern, implying on high memory usage on servers, significant network traffic, and substantial CPU time for email processing. Another reason for concern is the constant adaptation and improvement of spamming techniques. Several machine learning methods are used nowdays to address those issues. Machine learning is a sub-area of AI in which systems use algorithms that learn and improve independently based on data and previous experiences. Implementations of these types of anti-spam filters are also called spam classifier models. Section 2.5 describes in more detail the machine learning algorithms applied in the experiments and development of the Fully Homomorphic Encryption Spam Detector application (FHE-SD).

2.4 Artificial Intelligence

Artificial Intelligence (AI) is a field of research in computer science focused on implementing computational systems that attempt to imitate human cognitive functions, such as learning and reasoning. These systems are designed to acquire data, learn, and adapt for specific purposes. However, the definition of AI is still a subject of debate, involving not only computer science but also neuroscience, psychology, philosophy, and other areas.

Russel and Norvig describe four notable AI definitions adopted by researchers [54]:

- **Thinking Rationally:** The agent is intelligent if it can think logically to correctly understand the real world, such as in Aristotelian syllogisms. However, it does not generate intelligent behavior, which implies the *Acting Rationally* definition.

- **Thinking Humanly:** The agent is intelligent if it can think like humans. It involves psychology, cognitive sciences, and a complete understanding of human cognition.
- **Acting Rationally:** The agent is intelligent if it is capable of autonomous operation while adapting to the environment and pursuing the best outcomes.
- **Acting Humanly:** The agent is intelligent if it can pass the Turin test and act like humans.

Walter Pitts published the very first AI work in 1943, proposing a model of artificial neurons. In 1949, Donald Hebb created the Hebbian learning model, which describes rules for updating connection strengths between neurons. After that, several other works were considered to be AI related, but the most influential was the article by Alan Turing, published in 1950. It describes the Turing test, which involves three participants: a human judge, a human enquirer, and a computer attempting to convince the judge that it is human [55].

The following is a summary of the capabilities required to pass a Turin test [54]:

- **Natural Language Processing**: Communicate successfully in a human language.
- **Knowledge Representation**: Store what knows and hears.
- **Automated Reasoning**: Answer questions and draw new conclusions.
- **Machine Learning**: Adapt to new circumstances.

The AI field of research was formally established in 1956 with a workshop at Dartmouth College in Hanover. Later, in the 1980s, the industry saw a significant influx of investment, only to experience a subsequent decline. Much of the AI foundations used today were set in those decates, particularly the underlying statistical theories.

AI has become popular again in recent years due to the vast amount of data available through the Big Data phenomenon. ML algorithms require large training datasets to achieve acceptable accuracy, and big data has taken it to unprecedented levels. In 2023, the global amount of data reached 120 zettabytes [56] against 33 zettabytes in 2018 [57]. Another factor is the powerful hardware available; some learning algorithms running on specialized hardware, such as highly parallelized GPU architectures, can consume over 10^{14} operations per second.

It is critical to note that maintaining data centers to store and process such large volume of data has significant energy requirements and environmental impact. A recent peer-reviewed study indicates that AI technologies can consume approximately 29,3 terawatt-hours annually. This level of energy usage is comparable to the total energy consumption of entire countries [58].

2.5 Machine Learning

Machine Learning (ML) is the process of training computing systems with data, enabling them to learn and improve over time to predict outcomes based on past experiences; the prediction process is also

known as inference.

Machine Learning is a subset of Artificial Intelligence, illustrated in Figure 2.9, where lies some of the technologies used to implement the FHE-SD. The most critical stage in ML is the training, which requires extracting knowledge from a large volume of high-quality data [54].

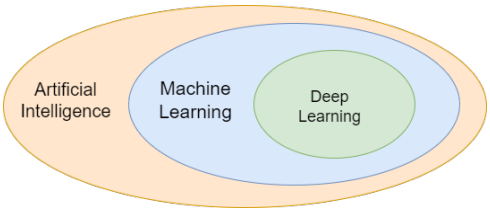


Figure 2.9: AI Subsets

Anti-spam filters are designed to detect spam emails and discard or move them to a spam folder. To implement that process, a developer could use standard programming and algorithms to compare the words in emails to a list of prohibited words. An email containing a certain number of these prohibited words is flagged as spam. While this method may sometimes work, it does not account for an unlimited number of unforeseen conditions and exceptions. Whenever a requirement changes, such as a new condition, the code or algorithm must be reworked, retested, and redeployed. These issues can be addressed by using machine learning algorithms to train ML models.

ML model training require large email datasets, which are collections of spam and non-spam email examples used by ML algorithms to learn the features necessary for spam classification. The ML algorithm trains an ML model by receiving as input samples of emails labeled as spam or ham. Once trained, the model can predict whether new emails are spam, even if they were not part of the training dataset. This process is known as inductive learning from examples, where machine learning algorithms make decisions based on patterns learned from known examples without human intervention or coding changes in unforeseen situations. As described below, inductive learning from examples is divided into supervised, unsupervised, and reinforcement learning [54, 59].

- **Supervised Learning**: The algorithms learn through examples, and the data is pre-labeled with the expected results or outputs. The resulting models are trained by taking data samples (input) and expected outputs (labels). Once trained, the models can predict the correct output for new unknown data. In the case of a model trained for spam detection, a large dataset of emails is the input training data, and each email in the dataset is labeled as spam or ham, which identifies the expected output. Inside supervised learning are two branches of algorithms: classification and regression. For classification algorithms, the prediction tasks have a target of discrete type, serving the purpose of predicting the category or class. The regression algorithms work over continuous variables, for example, predicting values such as prices and age in a population.
- **Unsupervised Learning**: The algorithms learn through examples, but the training datasets are

not pre-labeled, so the resulting models learn from the data without supervision, determining the patterns by themselves. Although there are many uncertainties in the results of these models, we can obtain useful information, such as unexpected categories. An algorithm example for unsupervised learning is the K-means clustering, which can be used to find similarities and identify classes, or categories, in large datasets [60].

- **Reinforcement Learning**: The algorithms learn through trial and error by receiving many unlabeled examples, like in unsupervised learning. However, the learning process is reinforced by positive and negative feedback, reward, or punishment, according to the solutions provided. The reinforcements adapt the models to achieve results that may provide more rewards.

Implementing machine learning solutions involves multiple stages. A typical workflow is represented in Figure 2.10 and is based on the approaches applied in this thesis. The concepts and techniques shown in the workflow are described in the following sections.

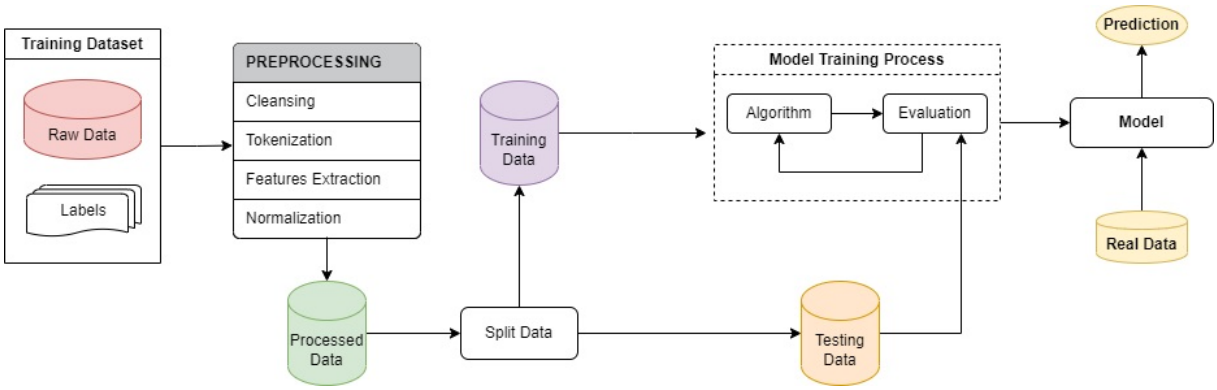


Figure 2.10: Model Training Workflow

2.5.1 Model Training

Model training is a fundamental stage in the machine learning workflow. It involves applying an ML algorithm to datasets, enabling the algorithm to learn from experience and examples to achieve specific goals. Each algorithm has distinct strengths and weaknesses, making some more suitable for certain types of problems and data than others. This thesis explores four algorithms: Linear Support Vector Classification (LinearSVC), Logistic Regression, Decision Trees, and Extreme Gradient Boosting (XGBoost).

The final result of the training process is a model, which is a program or computer system capable of making decisions and predictions from unseen input data. The performance of a model is directly influenced by the quality and volume of data used during the training process. Larger datasets provide enough examples to assist the model in achieving generalization. Generalization is the main aspect of

a successfully trained model, giving it the ability to adapt to new and unseen data. Models that fail to achieve generalization may experience issues related to either overfitting or underfitting [61].

- **Overfitting:** The machine learning model performs well on the training dataset but performs poorly on unseen datasets. It means that the model's accurate predictions mostly correspond to data in the training dataset, failing to generalize to other datasets [61].
- **Underfitting:** The training data may be of low quality or insufficient volume, failing to accurately represent the underlying pattern in the training dataset. It can result in high bias and poor performance on both training and testing datasets [61].

There are several statistical methods to evaluate model performance and detect issues, most of them utilize the following notation: True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN).

Accuracy is the most utilized ML performance metric. It is defined as the proportion of actual instances retrieved, positive and negative, among all instances retrieved, showing how close a set of measurements are to the correct values, see Equation (2.1). This metric is appropriate when the classes are well balanced, in other words, when the number of positives and negatives in the dataset is proportional [61].

$$\textit{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.1}$$

Precision is a metric suited for class-imbalanced datasets that measures the number of correct instances retrieved divided by all retrieved instances; see Equation (2.2). The precision of a measured system refers to the proximity between the results from repeated measurements under the same conditions, or how often true positive predictions are correct. This metric is particularly useful when it is important to identify the rate at which true positives are identified, as the system requires confidence in identifying true positives [61].

$$\textit{Precision} = \frac{TP}{TP + FP} \tag{2.2}$$

Recall or Sensitivity is better suited for class-imbalanced datasets and functions by measuring the number of correct instances retrieved divided by all correct instances; see Equation (2.3). In this case, the objective is to measure if the system can identify all true positives, even at the cost of incorrectly classifying negatives as positives [61].

$$\textit{Recall} = \frac{TP}{TP + FN} \tag{2.3}$$

F-Score, F-Measure or F1 is another metric for class-imbalanced datasets. F1 is defined as the harmonic average of precision and recall, making it useful when both metrics are relevant in the evaluation, see Equation (2.4). The F1 is popular because it combines precision and recall, and it is commonly used in Large Language Model [62]. However, some studies disagree on its effectiveness, particularly when precision and recall are not equally relevant in the dataset [63].

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \tag{2.4}$$

Confusion Matrix visually represents all prediction results from a model for a given labeled feature array. The Figure 2.11(a) describes the general notation, where green quandrants contain correct classifications, and orange quadrants the incorrect classifications. Figure 2.11(b) is an applied example, where the numbers in each quadrant represent the samples classified as TP, TN, FP, or FN. The dataset of 746 samples was classified by a model as follows: 350 true positives, 50 false positives, 19 false negatives, and 327 true negatives.

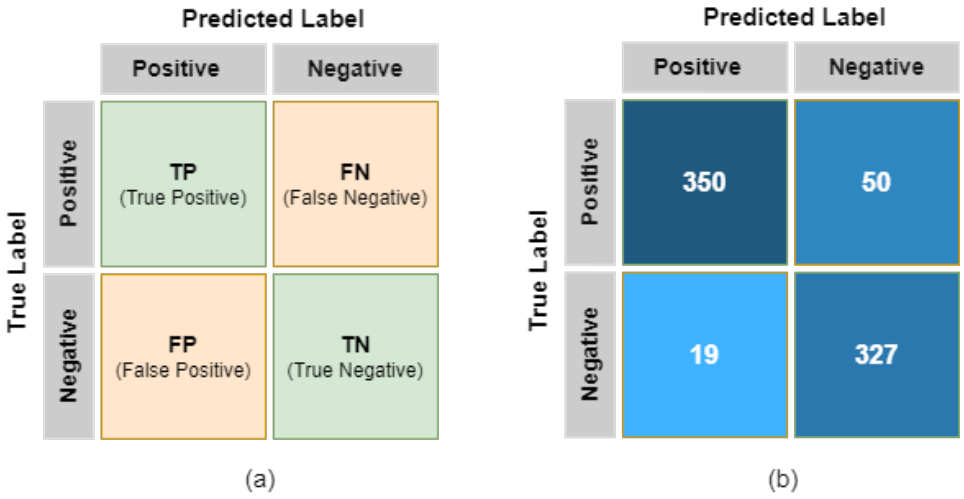


Figure 2.11: Confusion Matrix

Receiver Operating Characteristic Curve (ROC Curve) portrays the performance of a classification model at all classification thresholds by showing the true positive rate versus the false positive rate for binary classification systems. The axis assumes values between 0 and 1, and the points on the curve represent how well a classifier performs for a specific dataset. This visual representation helps to understand the trade-off between the benefits (true positives) and costs (false positives) in a classification system [64].

The plot utilizes the True Positive Rate (TRP) and False Positive Rate (FPR) as parameters, described in Equation (2.5) and Equation (2.6).

$$TRP = \frac{TP}{TP + FN} \quad (2.5) \qquad FRP = \frac{FP}{FP + TN} \quad (2.6)$$

In Figure 2.12, the graph displays a green diagonal line where the axes characterize the TRP and FRP at the same rates. For adequate results on the classifier, we expect to have the ROC curve in the upper left section drawn by the green reference line, also known as the *"random classifier"*. Points above the random classifier line represent good results, while points below represent poor results.

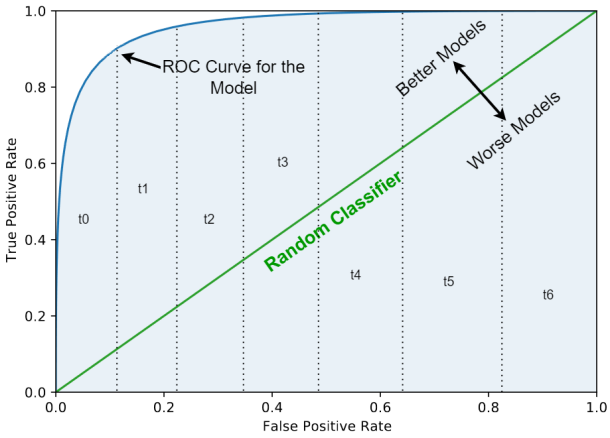


Figure 2.12: Example of ROC Curve and AUC

The Area Under the ROC Curve (AUC) denotes the overall performance of a binary classification model. As the TPR and FPR axes ranges from 0 to 1, the AUC score always lies between those two values, and the worst case is $AUC = 0,5$, meaning random guessing or no discrimination, and $AUC = 1,0$ for perfect prediction. Therefore, a higher AUC score reflects better model performance. The objective is to maximize the AUC to achieve the highest TPR and lowest FPR at the given threshold.

There are a few methods to calculate the AUC, such as dividing the area under the curve in multiple trapezoids, calculating the area of each trapezoid, and then summing them up. Another approach would be using integral calculus [65, 66]. In Figure 2.12, the blue area defines the AUC, the doted trapezoids the possible division that can be used to calculate the AUC score, and the dark blue line is the actual ROC curve for a given ML model.

Calibration Curve compares the reliability of binary classifiers' predictions and how they could be better calibrated. It involves plotting the frequency of the positive label on the y-axis against the predicted probability of a model on the x-axis. The challenge lies in obtaining accurate values for the y-axis, which can significantly impact the interpretation of a model's calibration. Intermediary probability values between 0 and 1 are not necessarily well-calibrated and can result in poor estimations.

The calibration curve is also useful for comparing the different models and finding which is better

calibrated. It helps identifying models that may be suffering from under or overconfidence. Figure 2.13 shows an example of a calibration curve, where the dotted diagonal line symbolizes what is a perfectly calibrated model, and the blue and red lines represent two examples of models' calibrations [67]. The dots above that line express underconfident predictions, and below the line, they are overconfident. In this case, model B is better calibrated, and the model A is underconfident.

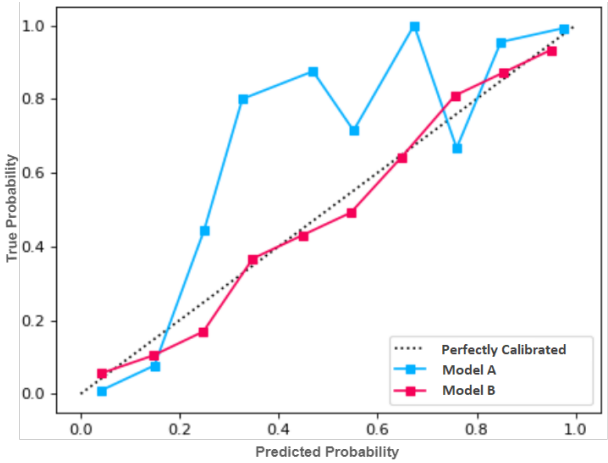


Figure 2.13: Example of Calibration Curve (Source:Unknown, possibly from Kaggle)

2.5.2 Data Preprocessing

Raw data comes in different formats and may contain inconsistencies, irrelevant information, and errors. For these reasons, they require standardization before being utilized in ML algorithms. It is achieved through preprocessing, which involves techniques to clean, transform, and make the data suited for statistical analysis and machine learning. This process helps remove what is unnecessary, enhances the data quality, and improves ML models' performance [68].

Some of the most important preprocessing techniques include cleaning, tokenization, lemmatization, transformation, and feature engineering [68, 69].

Data-cleaning is a set of techniques required to ensure the quality of a dataset by eliminating noise and inconsistencies. These techniques include removing stopwords, meaningless symbols, and replacing missing values.

Tokenization divides texts into minimal individual parts, known as tokens. For example, a sentence or phrase can be split into component words. During tokenization, the grammatical class of each token is identified using Part of Speech (PoS). This stage involves adding tags or labels to the words to define their function in the language of the text. Tokenization can be a complex process, especially in languages that do not have a clear separation between words, such as Japanese and Chinese.

Lemmatization reduces a given word to its root form while preserving its meaning and grammatical class. For example, plural words are reduced to the singular, and conjugated verbs are converted to their infinitive form. A similar process that also aims to reduce words to their root form is the *"stemming"*. However, it does not consider the linguistic context.

Transformation converts the data into a form appropriate for analysis and modeling. It involves several techniques that can help improve the algorithms' performance, normalize the data's scale, reduce its dimensionality, and make it usable in a computer system.

Feature Engineering further converts the transformed data into feature arrays that are used as input for the machine learning algorithms and models. Features are characteristics in datasets that allow the models to distinguish different categories or classes. In the scope of spam detection, the dataset is a collection of emails, the features are words extracted from the emails and converted into tokens, and the classes are labels such as spam or ham. Feature engineering can be divided into feature extraction and feature selection. Feature Extraction is the process of identifying the most relevant features from the transformed raw data. Feature selection extracts a subset of the most representative features to integrate into a machine-learning model. Some sources refer to the entire feature engineering process as feature extraction, we use the same convention in this thesis [70].

The preprocessing converts raw datasets into feature arrays, whose serve as inputs for machine learning algorithms and models. Typically, the resulting feature array is divided into two subsets: training dataset and testing dataset. The training dataset is input to train the ML models, while the testing dataset is an input applied to assess the performance of the trained models.

2.5.3 Machine Learning Algorithms

A machine learning algorithm has the necessary set of rules and tasks to train and create ML models. They can involve discovering new data insights and patterns or predicting output values from a given set of input variables. This section describes the four algorithms used to implement the FHE-SD.

Support Vector Machine

Support Vector Machine (SVM) is a Machine Learning algorithm used for binary classification, presenting good generalization capacity and robustness in large data dimensions. It performs better for data perfectly split by a vector line in a plane, such as in Figure 2.14(a). For non-linearly separable data like in Figure 2.14(b), it is necessary to make use of kernels, which are mathematical functions used to map the data into higher dimensional spaces, as illustrated in Figure 2.14(c), making possible the separation of the data by a hyperplane [71, 72]. This work uses a scikit-learn implementation of SVM, the Linear Support Vector Classification (LinearSVC) [73].

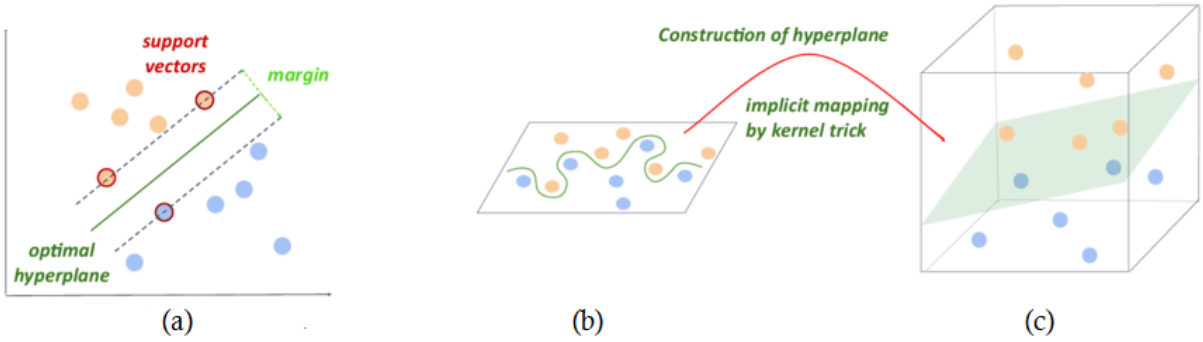


Figure 2.14: Support Vector Machine, adapted from [4]

Logistic Regression

Logistic regression predicts outcomes by estimating the probability of an event occurring or an instance belonging to a particular class. Like SVM and LinearSVC, logistic regression is very popular for binary classification and is also a linear algorithm. In general, SVM and logistic regression have similar performance [74, 75].

Decision Tree

Algorithms based on decision trees are structured in ways that remind tree structures. A decision tree classifier divides a problem into subsets of problems until it contains only one class or when a particular class is predominant, resulting in a leaf node, meaning no more divisions. The root node can be considered the training dataset, internal nodes evaluates features to make decisions, and the descending branches are the potential values from a node evaluation. The leaf nodes represent the resulting classes from the samples, with each leaf containing a label for the prediction result [76]. Figure 2.15 describes an example of a decision tree, with the root node in red, internal nodes in blue, leaves in orange, and the branches represented by the black arrows.

Extreme Gradient Boosting

Extreme Gradient Boosting (XGBoost) is an open-source library that implements an enhanced version of gradient boosting algorithm, becoming popular for its performance, scalability, portability, and winning several machine learning competitions. Like the Random Forest algorithm, XGBoost also uses ensemble learning, a process in which decisions from multiple tree-based ML models are combined to reduce errors and improve predictions when compared to a single ML model. Among the differences between Random Forest and XGBoost is that Random Forest is based on the concept of bootstrap aggregating (bagging) and trains each tree independently to combine their predictions. XGBoost uses an additive approach (boosting), where weak learners are sequentially trained to correct previous mistakes. It aims

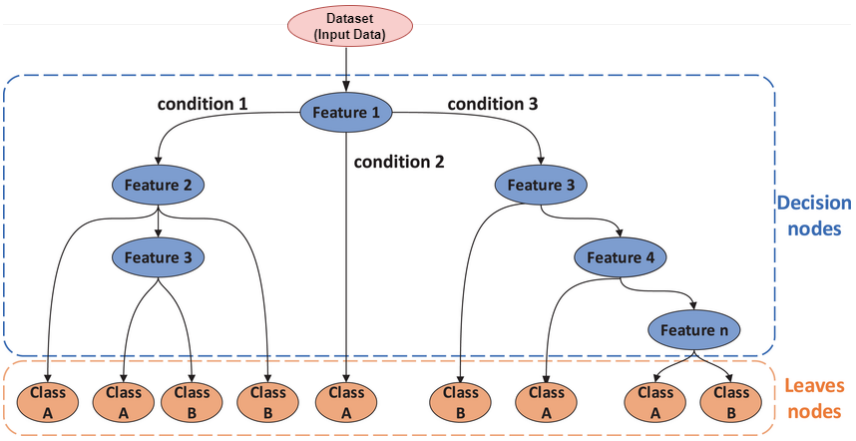


Figure 2.15: Decision Tree Example, adapted from [5]

to improve overall predictive performance by adjusting the model's weights based on previous iteration errors, gradually reducing prediction errors, and improving the model's accuracy [77].

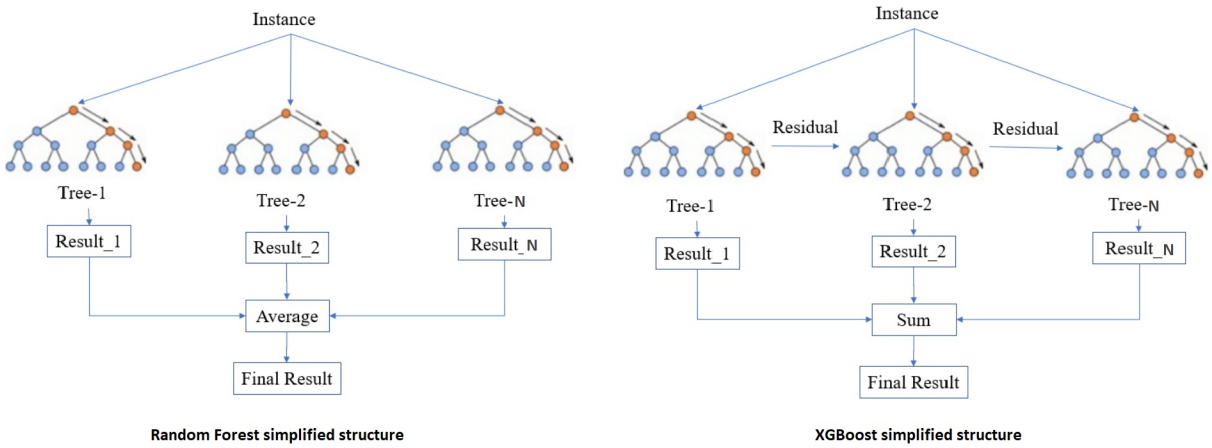


Figure 2.16: Random Forest and Extreme Gradient Boosting (XGBoost) simplified structures, adapted from [6]

Another feature in XGBoost is the use of tree pruning, which is an ML technique that reduces the size of regression trees by replacing nodes that do not contribute to enhancing the classification in the leaves, also preventing overfitting from the training data [78]. The Figure 2.16 compares Random Forest (RF) and XGBoost using a simplified version of their structures.

2.6 Cryptology and Cryptography

Cryptology is the science of secure communications and information security, which is divided into cryptography and cryptoanalysis. Cryptography studies encryption techniques, while cryptoanalysis studies vulnerabilities and ways to break encryption. Cryptographers have invested in research and technologi-

cal advances throughout cryptology history to maintain and preserve information security [79].

In the 20th century, Claude Shannon formalized several ideas regarding security and proposed the concept of perfect secrecy. A cryptosystem has perfect secrecy when knowledge of the ciphertext does not give the adversary any information about the message. Even if the adversary knows the ciphertext, the probability of breaking it does not change. Ciphertext is the result of applying an encryption algorithm to a plaintext, rendering it illegible without the decryption key. [80].

Cryptography studies techniques that allow secure communications over unsecured channels by disguising the message content in ways that only the recipient can access the original message. This process is called encryption, which applies cryptographic algorithms to protect secrecy and data integrity during transmission. The underlying plaintext can only be read if decrypted by those possessing the correct secret key. The set of algorithms utilized for encryption and decryption includes the *cipher* and the *encryption scheme*, also known as *cryptosystem*. Several cryptosystems exist, such as the AES, RSA, MD5, and SHA. The AES is a private key type of cryptosystem, while RSA is an example of a public-key cryptosystem [81].

A private key cryptosystem consists of three algorithms [82]:

1. Key-generation: A key-generation algorithm *Gen* is a probabilistic algorithm that outputs a key *k* chosen according to some distribution that is determined by the encryption scheme.

2. Encryption (*Enc*): Convert the original plaintext to an unreadable ciphertext. The encryption algorithm *Enc* takes as input a key *k* and a plaintext *m* and outputs a ciphertext *c*, see Equation (2.7).

3. Decryption (*Dec*): Transforms the encrypted text into readable text. The decryption algorithm *Dec* takes as input a key *k* and a ciphertext *c* and outputs a plaintext *m*, see Equation (2.8).

$$Enc_k = (m) \tag{2.7}$$

$$Dec_k = (c) \tag{2.8}$$

A complete definition of an encryption scheme requires the three algorithms *(Gen, Enc, Dec)*, as in Equation (2.9).

$$Dec_k = (Enc_k(m)) = m \tag{2.9}$$

A cryptosystem, or scheme, is considered secure if it is computationally infeasible for an attacker to obtain or deduce any part of the original text from its corresponding ciphertext. A scheme must be difficult to solve, but it does not imply that it is impossible. Cryptoanalysts eventually find vulnerabilities, such as in the SHA-1 and the MD5 cryptosystems, that are no longer recommended [83].

In 1883, Auguste Kerckhoffs stated his theorems, which became a worldwide cryptography paradigm known as the Kerckhoffs principle. Kerckhoffs says that the security of a cryptosystem must solely rely

on keeping the keys secret, and the cryptosystem itself should be open, available, and not maintained in secret. Therefore, the cryptographic scheme must be designed in such a way to be secure even if an adversary knows details of all the algorithms that conceive the cryptosystem [84].

Public-key Cryptography

Public-key cryptography, or asymmetric cryptography, is a type of cryptosystem that uses key pairs, one secret and the other public, where encryption and decryption are performed through both keys. One key encrypts the message, and the other key gives access to the hidden message exchanged between two parties. This scheme transforms plaintext into ciphertext using a public key *(Pk)* and an encryption algorithm, then a secret key *(Sk)* associated with a decryption algorithm is used in the decryption process [82].

Secret Key: *(Sk)* is the key that must remain secret and is known only to the person who generates it; the security of this cryptosystem is based on the secrecy of *Sk*.

Public Key: *(Pk)* is the key that can be publicly distributed without compromising security.

The Equation (2.9) can be modified for use in public-key cryptography as in Equation (2.10):

$$Dec_{Sk} = (Enc_{Pk}(m)) = m \tag{2.10}$$

Where:

- *Dec* is the decryption algorithm.
- *Enc* is the encryption algorithm.
- *m* is the original message.
- *Pk* is the public key.
- *Sk* is the secret key.

Before asymmetric cryptography, secure communications were conducted using symmetric cryptography, or private-key cryptography. In this type of scheme, both sender and receiver use the same key to encrypt and decrypt data. The main challenge is the secret key distribution because the recipient should have the key in advance to read the encrypted message. However, in many cases, it is impossible to send the key through a secure channel or meet physically to provide it safely. If a key is sent through a compromised channel or is stolen by an adversary, the sender must generate a new key and share it again with the recipients.

In 1976, Whitfield Diffie and Martin Hellman developed public-key cryptography to exchange cryptographic keys over a public channel securely. This method allows two parties without prior knowledge of each other to jointly establish a shared secret key over an insecure channel. This key can then encrypt subsequent communications using an asymmetric key cipher [85].

The following example demonstrates a communication using public-key where Alice is the sender, and Bob is the receiver:

1. Alice generates a pair of keys, *Pk* and *Sk*.
2. Alice sends *Pk* to Bob.
3. Bob uses *Pk* to encrypt his message and sends it to Alice.
4. Alice receives Bob´s message and decrypts it using *Sk*.

Public-key encryption algorithms are based on one-way functions, which are easy to compute for any given input but difficult to calculate their inverse. Therefore, the security of these algorithms is guaranteed by the fact that the underlying mathematical problems are nearly impossible to solve in polynomial time [86].

2.7 Homomorphic Encryption

Homomorphic Encryption (HE) is an enhanced form of public-key cryptography that supports arbitrary computations directly on encrypted data. The main aspect is that the decrypted ciphertext result obtained after a specific computation on encrypted data must be identical to the result obtained by applying the computation over data on-clear [87].

The Figure 2.17 illustrates the concept of homomorphic encryption and its application, where a user encrypts data and sends it to a third-party cloud server for processing. The third-party server can perform computations on this encrypted data and send the encrypted results back to the user. Only the user who owns the secret key used for encryption can decrypt the result. In this example, the data remains encrypted during the entire process, and not even a system administrator could have access to the user's private data without the secret key [7].

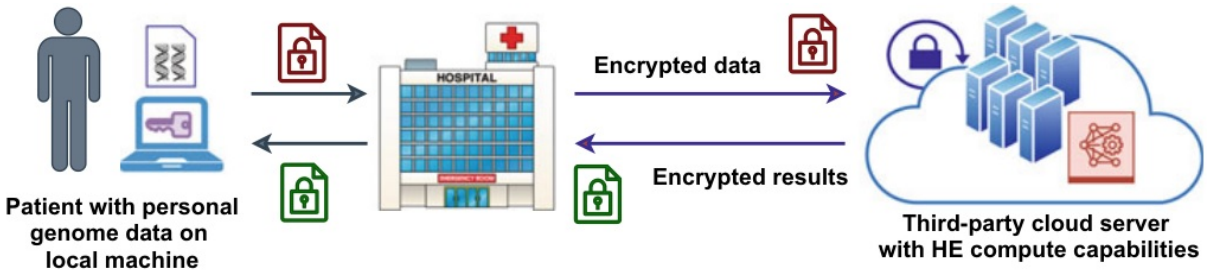


Figure 2.17: Secure computations over encrypted data using HE [7]

The idea of homomorphic encryption was first introduced in 1978 by Ronald Rivest, Leonard Adleman, and Michael Dertouzos in the paper *"On data banks and privacy homomorphisms"*, through the concept of *"privacy homomorphism"* [88]. They describe privacy homomorphism as hypothetical encryption functions that enable encrypted data to be processed without preliminary decryption. However, the first homomorphic

cryptosystem was already proposed in 1977 - the Rivest-Shamir-Adleman (RSA) cryptosystem [89].

The Rivest-Shamir-Adleman (RSA) is a public-key cryptosystem that is multiplicatively homomorphic. Therefore, the product of two ciphertexts is equivalent to raising the product of their plaintexts to the power of the secret key.

Given two ciphertexts $c_1 = m_1^e \pmod q$ and $c_2 = m_2^e \pmod q$, where:

- m_1 and m_2 are plaintexts.
- q is the product of two large prime numbers.
- e is such that $gcd(e, \Phi(q)) = 1$.
- $\Phi(q)$ in $gdc(e, \Phi(q))$ is the $Euler's Phi Function$ [90] for q.
- $c_3 = c_1 * c_2$.

The RSA is multiplicatively homomorphic if:

$$c_3 = c_1 * c_2 = m_1^e * m_2^e \pmod q = (m_1 * m_2)^e \pmod q \tag{2.11}$$

Therefore, the decryption of $c3$ equals $m1 * m2$ [91]. The algorithm supports arbitrarily unlimited multiply operations over encrypted data but does not support addition operations, for that reason RSA is considered a Partially Homomorphic Encryption (PHE) scheme. When discussing homomorphic multiplication and addition, we refer to boolean circuits executing the operations on binary data using *AND* logical gates for multiplications and *OR* logical gates for additions. The logical gates support only binary addition and multiplication, so subtraction is possible by inverting the addition operation and division by inverting the multiplication operation. The logical gates are devices that execute the logical operations in circuits.

2.7.1 Boolean Algebra and Boolean Circuits

The english mathematician George Boole proposed the boolean algebra as a variant of Aristotle's propositional logic, using binary numbers to execute operations based on the binary system. Boolean algebra concerns binary variables and logic operations, allowing the resolution of complex equations that result in a boolean value such as true (1) or false (0). Table 2.1 shows the most basic operators in boolean algebra and Table 2.2 the boolean algebra properties or laws. These properties simplify complex boolean equations, helping reduce the number of logic gates necessary to perform a particular logic operation [92, 93].

As this work does not have the objective of discussing every concept and theory in such broad fields, we left out in Table 2.2 the *De Morgan's theorem*, which basically explains that a *NAND* gate is equivalent to an *OR* gate with inverted inputs. Similarly, a *NOR* gate is equivalent to an *AND* gate with inverted inputs.

Table 2.1: Basic Boolean Operators

Logical Operation	Operator	Notation	Result	Usage
Conjunction	AND	$\wedge$	Logical Multiplications	$A.B$
Disjunction	OR	$\vee$	Logical Additions	$A + B$
Negation	NOT	$\neg$	Logical Complement	$\neg A$
Exclusive-OR	XOR	$\oplus$	[94]	$A \oplus B$

Table 2.2: Boolean Algebra Properties

$Identity$	$A + 0 = 0$ and $A \cdot 1 = A$
$Zero\ and\ One$	$A + 1 = 1$ and $A \cdot 0 = 0$
$Inverse$	$A + \neg A = 1$ and $A \cdot \neg A = 0$
$Commutative$	$A + B = B + A$ and $A \cdot B = B \cdot A$
$Associative$	$A + (B + C) = (A + B) + C$ and $A \cdot (B \cdot C) = (A \cdot B) \cdot C$
$Distributive$	$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$ and $A + (B \cdot C) = (A + B) \cdot (A + C)$

Figure 2.18 shows gates and boolean operations, and it is common to use *"Truth Tables"* to represent all possible inputs and outputs of a given circuit.

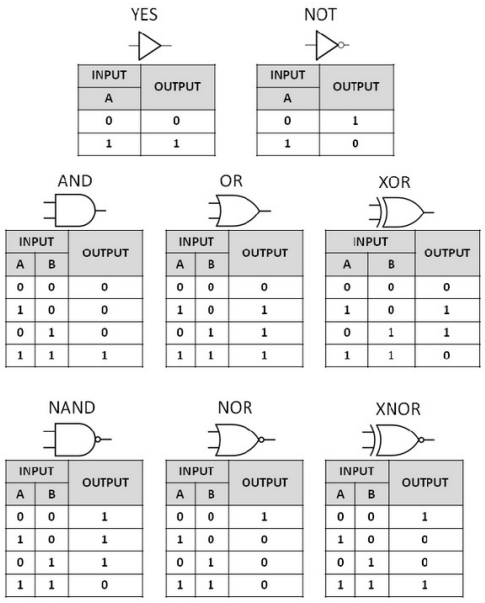


Figure 2.18: Summary of the common Boolean logic gates with symbols and respective truth tables [8]

Boolean Circuits

A Boolean circuit is a set of logic gates connected by wires that are used to describe boolean equations and can be modeled as a directed acyclic graph. The circuit receives a set of Boolean variables as input, and the logic gates process and generate a set of Boolean variables as output. The circuit depth is the distance between the circuit's inputs and outputs, and the size of the circuit is the number of

edges in the graph. A model of computation based on boolean circuits is equivalent to a Turing machine capable of computing arbitrary algorithms if the circuit's size is limited. Likewise, algebraic circuits also correspond to a complete computational model. Therefore, obtaining a function that is simultaneously a homomorphism and can be used for encryption means that adding and multiplying encrypted texts is possible. Consequently, it is also possible to compute algebraic circuits in a homomorphic way. The Figure 2.19 illustrates a simple circuit for the boolean equation $Y = \neg B \neg C + A \neg B$ [9].

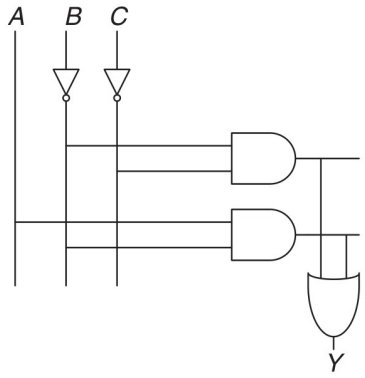


Figure 2.19: Boolean Circuit Example [9]

2.7.2 Types of Homomorphic Encryption Schemes

Homomorphic encryption is necessary because it allows data to remain encrypted while being processed, thus maintaining user data privacy. Based on their functional capabilities and limitations, homomorphic encryption schemes can be classified into three types: Partially Homomorphic Encryption (PHE), Somewhat Homomorphic Encryption (SHE), and Fully Homomorphic Encryption (FHE). Their primary difference is the type of logical operations supported and how many times they can be executed in the circuit, which reflects the circuit depth [87].

Partially Homomorphic Encryption

Partially Homomorphic Encryption (PHE) represents the earliest type of homomorphic cryptosystem; these schemes evaluate circuits consisting of only one type of logical gate or operation. While that limits the applications for PHE, they are relatively easy to design. An example of a homomorphic additive PHE scheme is the Paillier cryptosystem, which is a type of public-key cryptosystem that supports only addition operations in the circuit [95, 96]. Another example of the PHE scheme is the RSA, but in this case, it is homomorphic multiplication, supporting infinite multiplications but no addition; its homomorphism was described in Equation (2.11).

Somewhat Homomorphic Encryption

Somewhat Homomorphic Encryption (SHE) refers to homomorphic schemes that can evaluate two types of gates, addition and multiplication, but only for a subset of circuits. Each SHE scheme is designed to execute logical operations until a certain depth. For each operation, the ciphertext generates noise in the data; if the noise reaches a specific threshold, the ciphertext can no longer be decrypted. The noise component is essential for security, but it grows with each operation [87]. Despite these limitations, SHE schemes are considered an advance from PHE towards FHE and are still valuable in scenarios where the computational requirements are known and delimited. They can securely evaluate functions of a known depth and provide enhanced efficiency compared to Fully Homomorphic Encryption schemes [97]. Examples of SHE schemes include Yao, SYY, IP and BGN [98, 99].

The Figure 2.20 shows the homomorphic encryption timeline before the introduction of Fully Homomorphic Encryption.

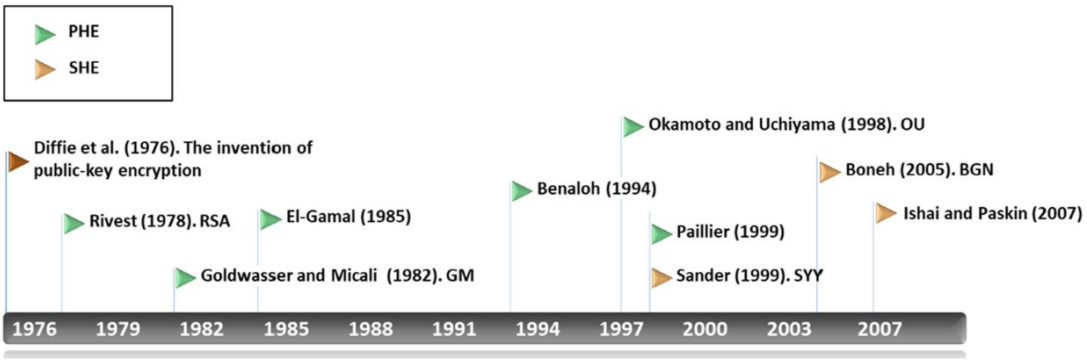


Figure 2.20: Homomorphic Encryption timeline before FHE introduction [10]

Fully Homomorphic Encryption

In 2009, Craig Gentry introduced the first viable Fully Homomorphic Encryption (FHE) scheme. Due to the inherent noise growth, unlimited multiplication and addition operations on encrypted data were impossible. Gentry's breakthrough came with the bootstrapping technique, which enabled noise reduction without the need for data decryption. In Section 3.2, we tackle more details about the FHE advances and related works.

The following notation demonstrates FHE with respect to the addition and multiplication operations when a function f is composed of finitely many additions and multiplications in the ring [100]:

$$Dec(f(c_1, ..., c_t)) = f(m_1, ..., m_t) \tag{2.12}$$

A simplified form to express the FHE addition and multiplication property in rings can be described

as two ciphertexts, C_1 and C_2, encrypted from two plaintexts, m_1 and m_2, as demonstrated in Equation (2.13) for addition and in Equation (2.14) for multiplication [100].

$$Dec(c1 + c2) = m1 + m2 \quad (2.13) \qquad Dec(c_1 \cdot c_2) = m1 \cdot m2 \quad (2.14)$$

2.7.3 Lattice, LWE and RLWE

The security of cryptographic schemes depends on the hardness of the mathematical problems upon which they are built. Most FHE schemes are based on lattices. Lattice is a discrete and additive subgroup of Euclidean space that can be represented as an arrangement of regularly spaced dots forming a grid stretching out to infinity. Lattice is the foundation for many cryptosystems because it is believed to be hard to solve even for quantum computers, making them candidates for post-quantum cryptography [101]. Examples of lattice problems include Learning With Errors (LWE), Ring Learning With Errors (RLWE), Shortest Vector Problem (SVP), and the Closest Vector Problem (CVP). These problems are favored for building secure cryptographic systems because they provide security based on deeply explored problems in lattice-based cryptography [87].

The RLWE is a variation of the LWE problem for polynomial rings over finite fields. It has been frequently applied in cryptography because a hypothetical solution to the RLWE problem could also solve the NP-hard [1] Shortest Vector Problem (SVP) in a lattice. Such a fact implies that RLWE has NP-hardness similar to SVP, which is a well-studied and scrutinized problem. There are also other variations of the LWE, such as General LWE (GLWE), General GSW (GGSW) [103], and Glev [104].

A simple example of the SVP asks for the shortest non-zero vector in a given lattice, where [2]:

- γ-approximate SVP (SVP$_\gamma$): it requires finding the "almost" shortest vector: for $\gamma \geq 1$, find the vector $v \in \mathcal{L}$ that satisfies the condition $\|v\| \leq \gamma \cdot \lambda_1(\mathcal{L})$. Figure 2.21 illustrates this case.
- γ-unique SVP (uSVP$_\gamma$): it requires finding the shortest vector that is γ times smaller than $\lambda_2(\mathcal{L})$: for $\gamma \geq 1$, find the vector $v \in \mathcal{L}$ for which $\gamma\lambda_1(\mathcal{L}) < \lambda_\in(\mathcal{L})$.
- Decisional SVP (DSVP or GapSVP$_\gamma$): it requires deciding which given bound for the shortest vector is the right one: for $\gamma \geq 1, r > 0$, decide whether $\lambda_1(\mathcal{L}) \leq \nabla$ or $\lambda_1(\mathcal{L}) \geq \gamma \cdot \nabla$.

2.7.4 Noise Growth

Noise is generated by the underlying RLWE and lattice-based problems utilized to implement HE schemes, which make homomorphic ciphertexts *"noisy"*. Noise is necessary to enhance the security of ciphertexts. However, if it grows beyond a certain threshold, the ciphertexts become impossible to decrypt. Each homomorphic operation increases the noise in the ciphertexts; for example, addition doubles the noise, and

[1] From computational complexity theory, NP-hard is a complexity class that consists of decision problems that are at least as hard as the hardest problems in nondeterministic polynomial time (NP) [102].

[2] This work does not cover every the mathematical details (SVP demonstration extracted from [87]).

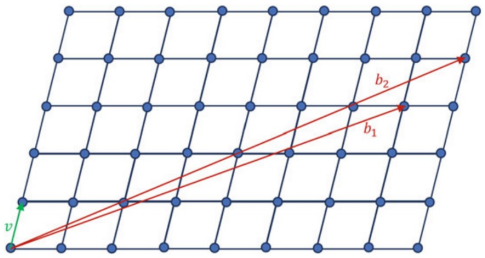


Figure 2.21: Lattice Shortest Vector Problem (SVP) [11]

multiplication squares the noise in schemes such as Cheon-Kim-Kim-Song (CKKS), making the decryption fail if the noise gets too high. Multiple approaches to deal with noise growth are possible depending on the scheme and the depth of the circuit evaluated. Such as choosing scheme parameters, N and $logQ$ value, that are large enough so that the coefficients in the ciphertext polynomial can accommodate the growing noise without destroying the underlying message. For applications like machine learning training, the depth of the circuit d can be as large as 150, leading to the parameter log Q to be larger than $(d * log\Delta) + \eta = (150 * 54) + 40 = 745,240$, which is practically impossible to work with due to the computational overhead, demonstrating that there is a limit in practice on how large the scheme parameters can be for an application [7]. Homomorphic schemes address noise growth through two main approaches: bootstrapping and Less Noisy Operations.

2.7.5 Bootstrapping

As mentioned in Section 2.7.2, the notion of bootstrapping was introduced by Gentry in 2009, and is used in cryptosystems such as GSW and TFHE. Most FHE schemes are based on hard lattice problems, so the generated ciphertexts contain a certain quantity of noise to ensure encryption security. Such noise increases for every operation computed homomorphically on the ciphertext. However, to decrypt the ciphertext, the noise level must be below a specific threshold, or the noise can overflow the data, making the decryption impossible, as illustrated in Figure 2.22.

Each scheme has a limited number of operations before the ciphertext accumulates too much noise. Once the limit is reached, it has to be bootstrapped to reduce the noise. Generally, the process involves applying a cryptographic key and algorithm to the exhausted ciphertext, which results in a refreshed ciphertext that is different from the original but with reduced noise and without exposing the data's secrecy. In summary, bootstrapping refers to a process in which a ciphertext can be transformed into a new ciphertext with the same underlying plaintext but with noise minimized. It allows additional homomorphic operations on the ciphertext without the noise or errors accumulating to the point where the decryption becomes impossible. Besides the performance improvements since bootstrapping was introduced, it is generally considered an expensive operation and should be avoided when possible [7].

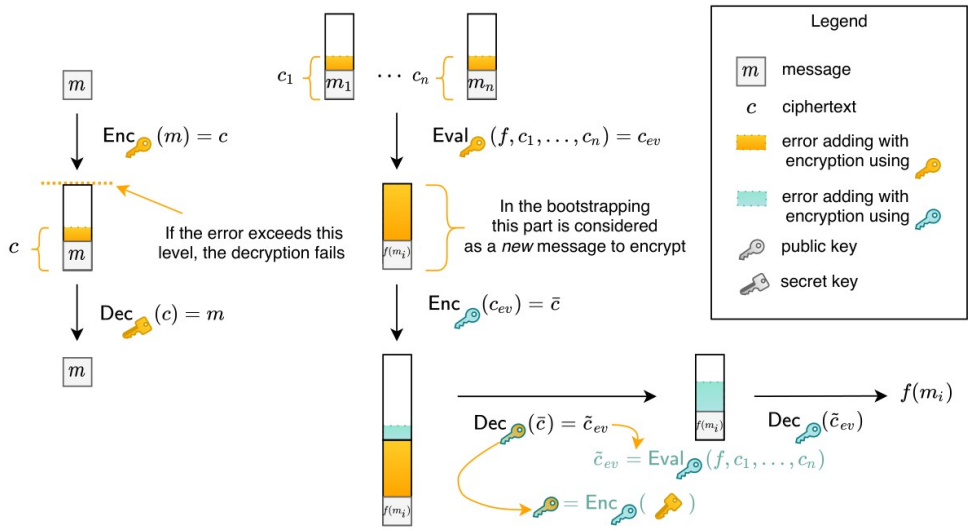


Figure 2.22: Bootstrapping [12]

2.7.6 Less Noisy Operations

The Leveled FHE schemes do not use bootstrapping to reduce ciphertext noise. Instead, they apply leveled techniques, such as less noisy operations, that implement cryptographic parameters to minimize the amount of noise generated in homomorphic operations. Particular attention is paid to multiplications, as they introduce the most noise in ciphertexts. The parameters' purposes include determining an upper-bound noise or noise budget and setting the maximum number of operation levels. Those levels directly reflect the circuits' depth and the longest sequence of consecutive operations possible before corrupting the ciphertext and making decryption impossible [105].

Defining a large circuit depth increases the computational cost, encryption and decryption complexity, and the size of keys and ciphertexts. Therefore, leveled schemes are recommended for small circuits with well-defined logical operations and functions. Examples of Leveled FHE schemes include BFV, BGV, and CKKS. There are also schemes that combine leveling and bootstrapping to control noise growth by applying the method that provides more benefits for ongoing operations. That is the case of the TFHE scheme, mainly based on bootstrapping but recently had added support for leveled operations [87].

2.7.7 Fully Homomorphic Encryption over the Torus

The Fully Homomorphic Encryption over the Torus (TFHE) is the underlying FHE scheme applied in this thesis. FHE-SD is implemented using a TFHE compiler introduced by Zama [106] and the machine learning framework Concrete-ML (CML), which implements an FHE variation of the machine learning library scikit-learn [107]. TFHE is based on the RLWE, LWE, and RGSW problems, which were discussed in Section 2.7.3. It was proposed as an improvement over the FHEW schema by enhancing its efficiency

[108] through the implementation of a ring variant of the bootstrapping [109] procedure while using a similar approach to that of FHEW. However, TFHE can also make use of the leveled approach in certain cases [110].

The TFHE scheme functions within a mathematical structure known as a torus, defined as the set of real numbers modulo *one*. This feature provides TFHE with greater efficiency for non-polynomial operations. One key distinction between a torus and a ring lies in the multiplication of elements. Unlike a ring, the multiplication of two elements within the torus does not yield a value within the torus. However, if an element represented on the torus is multiplied by an integer, the result is defined within the torus. TFHE supports addition and multiplication over integers, other operations are converted into a Table Lookup (TLU) - discussed in Section 2.7.8.

The following is an excerpt from Ilaria Chillotti's work where she mathematically defines the real torus [111]:

> "The letter "T" in TFHE refers to the real torus $\mathbb{T} = \mathbb{R}/\mathbb{Z}$, that is, the set of real numbers modulo one. Any two elements of $\mathbb{T}$ can be added modulo one: $(\mathbb{T}, +)$ forms an abelian group. But $\mathbb{T}$ is not a ring as the internal product of torus elements is not defined. The external product between integers and torus elements is however well defined. Given $k \in \mathbb{Z}$ and $t \in \mathbb{T}$, the element $k \cdot t \in \mathbb{T}$ is defined as: $k \cdot t = \underbrace{t + \cdots + t}_{k \text{ times}}$ if $k \geq 0$ and $k \cdot t = (-k) \cdot (-t)$ if $k < 0$. Mathematically, $\mathbb{T}$ is endowed with a $\mathbb{Z}$-module structure. Polynomials can as well be defined over the torus."

Therefore, the mathematical properties of the real torus $\mathbb{T} = \mathbb{R}/\mathbb{Z}$ allow operations on encrypted data, such as additive modulo *one* and multiplicative, as the external product between integers and torus elements is well defined [110].

Figure 2.23 illustrates a torus structure used to visualize the encoding as an alternative to the bit representation. In orange is an example of encoding of the Most Significant Bit (MSB).

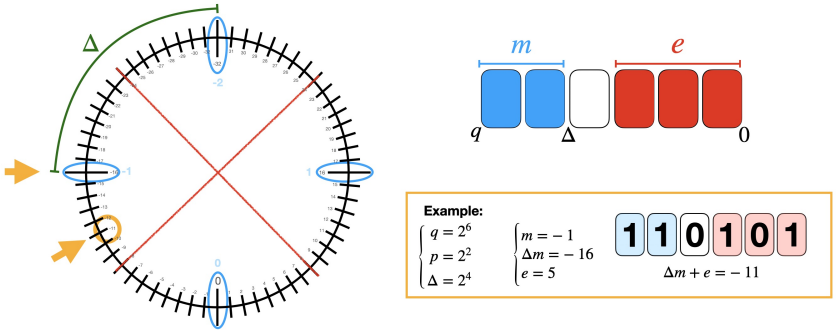


Figure 2.23: Torus Visualization [13]

In April 2021, the Joint Technical Committee of the International Organization for Standardization

(ISO) and the International Electrotechnical Commission (IEC) introduced a new proposal for expanding the current set of encryption standards known as ISO/IEC 18033. This expansion would involve the addition of Fully Homomorphic Encryption as an 8^{th} part with the committee recognition of TFHE as one of the encryption schemes that meet the criteria for standardization under ISO/IEC. Currently, the project is marked as deleted without indicating if it was canceled or if the document was moved [87, 112].

2.7.8 Table Lookup and Quantization

Table Lookups (TLUs) are arrays that replace runtime computation by a simpler array indexing operation. It provides faster data retrieval and reduces the need for time-consuming calculations. The TFHE scheme, which is discussed in Section 2.7.7, supports three primitive operations on integers: addition, subtraction, and multiplication. Other operations, such as tensor manipulation and float point operations, are converted to TLUs. Table Lookups are flexible and commonly used, but they can be resource-intensive, and it is advisable to avoid them when possible, prioritizing the primitive operations of addition and multiplication.

Minimizing the TLUs' bitwidth by diminishing the size of the inputs is an important aspect of enhancing TLUs' performance. However, a smaller bitwidth affects the operations' accuracy due to the reduced precision of the values represented. The conversion of values from one bitwidth to another is achieved through quantization, a process that involves turning continuous and large sets of values into discrete representations. Quantization operations such as rounding, truncating, and floating point are converted to integer equivalents and then into TLUs.

Summary

This chapter provided an overview of the theories, concepts, and technologies applied in this thesis. We covered internet protocols, spam detection techniques, machine learning, and cryptography, focusing mainly on public-key cryptography and Fully Homomorphic Encryption (FHE).

3

Related Work

Contents

Based on the objectives defined in Chapter 1, this chapter is structured as a brief literature review to assess the research progress in FHE since its inception and to help determine whether it remains a promising technology confined to academic circles or a feasible solution to be integrated into real-world applications. Section 3.1 contains selected works about applications for FHE that demonstrate the distinct contexts where it can be incorporated. Section 3.2 recollects the evolution of prominent FHE schemes. Section 3.3 addresses recent usability advancements, such as the introduction of libraries and compilers that abstract the complexity and facilitate the development of FHE applications. Section 3.4 examines the standardization efforts from various sectors and entities, emphasizing the current relevance of FHE. Section 3.5 reviews works about spam, recent anti-spam measures, and threats, which influenced our decision on which spam detection functions to incorporate into FHE-SD. Section 3.6 discusses and compares the contributions of five papers that effectively investigate homomorphic encryption applied to spam detection.

3.1 Fully Homomorphic Encryption Applications

This section presents different applications of FHE, such as healthcare, Electronic Voting, and Machine Learning. Some of these research works derive from a survey on homomorphic encryption applications [14]. The Figure 3.1 provides an overview of major sectors that can benefit from implementing FHE [87].

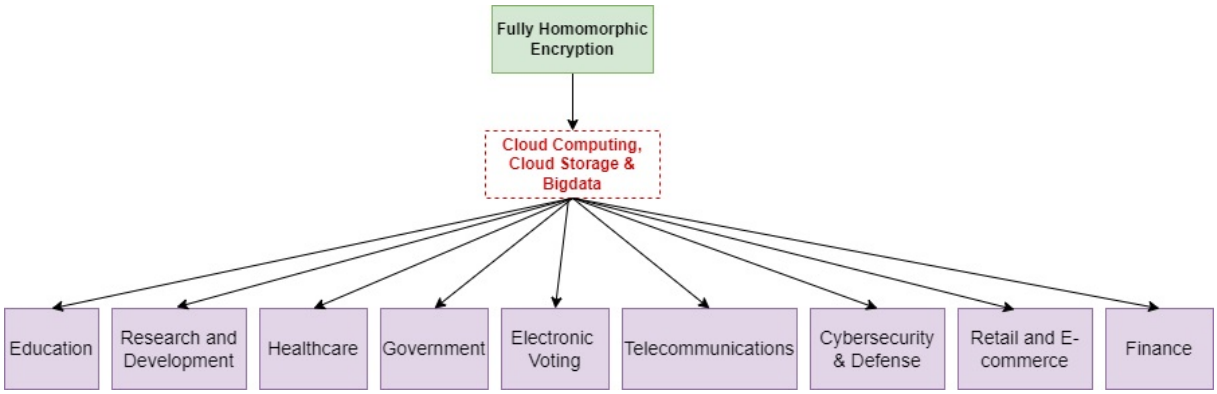


Figure 3.1: Domains and sectors where FHE can be applied

3.1.1 Machine Learning

The development of machine learning models must consider the security of the input data, output data, and the model itself. A handful of FHE schemes already support several ML algorithms. These schemes are useful in keeping the results of predictions secure, sharing homomorphically encrypted datasets for collaborative AI projects, and protecting ML models from adversaries' exploitation. Among the many

pieces of research on using FHE with machine learning are Concrete-ML and TFHE-rs, utilized in the application developed in this thesis. However, we noticed that CKKS-based solutions for ML are the most recurring in the related literature.

The first works we found are from Chen, Chillotti, and Song. They proposed a solution using the CKKS scheme to protect users' data from leakage during inferences on users' images in cloud providers. Their approach ensures that the cloud cannot access any user information during the inference process, thus protecting user privacy. The authors identified potential improvements in the integration of the network structure with the ciphertext structure in the neural network inference stage, which could effectively reduce the complexity of ciphertext transmission across different model layers. Therefore, they suggest a more efficient ciphertext allocation for the weight matrix to remedy this. Another contribution is a method that reduces the noise that could slightly blur the underlying pictures and affect the accuracy [113, 114].

Another notable work involves an efficient method capable of performing k-means training and classification on a set of encrypted data points using Euclidean distance based on the CKKS scheme. This method applies a sequence of interactions between the server and the client that takes place during the training phase. In essence, the server may conduct a single loop and provide the result. This result is then processed by the client, who may request another iteration by communicating a new set of centroids. The *client-performed* data management step is faster than a traditional k-means execution, reducing the complexity [115].

In the context of federated learning, Wang proposes an intrusion detection approach that utilizes federated learning to enhance client-side security in Advanced Metering Infrastructure (AMI) networks [116]. The CKKS scheme safeguards the model parameters and mitigates poisoning attacks in federated learning. Another federated learning system combined with FHE was proposed by Fang and Qian [117], where the security of the data and model is maintained during the training phase. The authors confirmed that the model trained using the suggested methodology outperforms models trained using standard methods.

3.1.2 Healthcare

Preserving the privacy and confidentiality of patients is an ethical obligation to protect their dignity, culture, and self-determination. Furthermore, preventing the unauthorized use of patient data for purposes such as research, blackmail, segregation, or any potential abuse [118]. This section presents works on data security in healthcare systems, applications, and storage systems for healthcare data. The first article proposes an arithmetic-based homomorphism that encourages secure information sharing in different communication settings. The proposed system relies on FHE and ElGamal cryptosystems to encrypt the data [119].

In the medical image processing scope, we have the Optimum Homomorphic Wavelet Fusion (OHWF),

a homomorphic implementation of multi-model medical image fusion. Medical Image fusion is a technique used in clinical analysis where multiple images in multiple modes are combined to obtain superior information and visual appearance, facilitating the diagnosis of diseases and tumors [120].

The work on secure searching of biomarkers proposes a protocol to securely outsource the process of matching biomarkers in DNA sequences by applying homomorphic operation bases in a hybrid GSW scheme. The work contribution helps advance the prevention of genetic discrimination and violation of personal privacy via genetic disclosure [121]. Another study explored the application of homomorphic encryption to secure data obtained through wireless medical sensors used to monitor patients. The solutions proposed utilized the Pallier and ElGamal cryptosystems [122].

3.1.3 Electronic Voting

According to Rajalakshmi, data streaming is the most a critical element of electronic voting. Systems using insecure streaming channels are unreliable and raise suspicion [123]. A study by Papadimitriou on segmented data stream has a similar view, concluding that electronic voting systems require homomorphic-based barriers to ensure the system's integrity [124]. Gibson identified verifiability, anonymity, dependability, security, and trust as the main challenges associated with voting systems [125]. Cortier's work emphasizes that the increasing adoption of electronic voting demands additional efforts to enhance the security of the systems and prevent human interference [126]. The article *"Efficient detection for malicious and random errors in additive encrypted computation"* highlights a potential use of homomorphic encryption for aggregating encrypted votes in electronic elections [127].

3.1.4 Blockchain

The paper *"Power data blockchain sharing scheme based on homomorphic encryption"* proposes a blockchain solution utilizing a scheme that enforces data-sharing rules and trustworthy computations through smart contracts. It employs homomorphic encryption methods to prevent data leakage [128]. Two other works combine the use of blockchain in electronic voting [129, 130] and in Internet of Things (IoT) based solutions [131].

3.1.5 Internet of Things

Homomorphic encryption can improve security in the Internet of Things (IoT) by ensuring that the data transferred and stored by IoT devices remains encrypted during processing. Studies have shown the prospect of combining Fully Homomorphic Encryption and IoT devices to achieve data privacy, data aggregation and analytics, and reliable outsourced data processing [132, 133]. The enhanced security during information transfer could also be applied to media data generated by IoT devices [134].

3.1.6 Cloud and Bigdata

Chakraborty and Patra explored the use of Functional Encryption (FE) combined with homomorphic encryption in big data analytics. They developed an asymmetric key encryption technique that allows users to encrypt and access only limited functionality of plaintext without revealing any undesired details [135]. There are works on applications of FHE in cloud and big data that discuss solutions for integrating homomorphic encryption into big data and online systems. The work by Menandas and Joshi [136] evaluates several candidate homomorphic algorithms for secure big data processing, such as Paillier, ElGamal, and Okamoto Uchiyama cryptosystem [137]. The study also highlighted the main challenges of big data, such as distributed storage.

3.2 Fully Homomorphic Encryption Schemes

Throughout the evolution of the FHE field, distinct generations have emerged, characterized by the adoption of common techniques in homomorphic schemes or significant advances in performance and optimization. These developments include the introduction of bootstrapping schemes and further improvements, leveled schemes, and utilizing LWE, RLWE, and FHEW security models [12, 14, 138]. This section presents an overview of the main FHE schemes, different branches, and a timeline in the Figure 3.2.

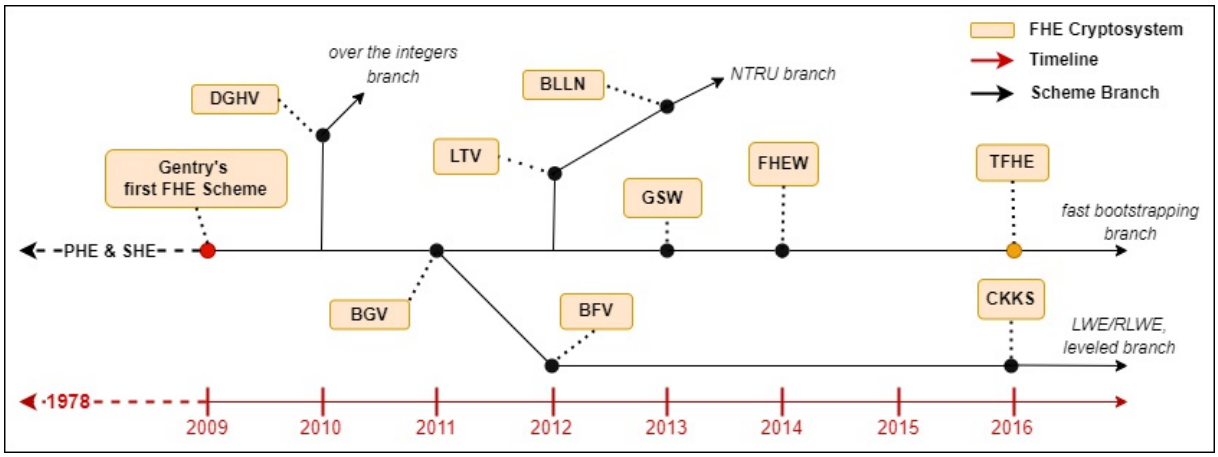


Figure 3.2: Fully Homomorphic Encryption Timeline

First Generation FHE: The first generation of FHE started in 2009, with Craig Gentry's first plausible example of a secure encryption scheme that supported both addition and multiplication homomorphically. Gentry's contribution opened new paths in the research of FHE, transforming it from a theoretical concept into a practical reality [139].

His algorithm for fully homomorphic encryption using lattice-based cryptography supports addition and multiplication operations on ciphertexts, from which circuits can be created to perform arbitrary cal-

culations. The construction follows several steps in which noise is introduced and then reduced using bootstrapping. It makes calculating any number of additions and multiplications possible without losing control of the noise generated. The implementation of Gentry's original cryptosystem reported a time of approximately 30 minutes per simple bit operation [140]. However, extensive optimization in the following years has significantly improved the runtime performance compared to the early implementations.

In 2010, Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathanan built the DGHV scheme utilizing parts of Gentry's first algorithm, and only addition and multiplication operations over the integers. As opposed to more complex structures in Gentry's scheme of ideal lattices, DGHV scheme is conceptually simpler and is based on the approximate GCD problem, a hardness assumption different from that of LWE [141].

Second Generation FHE: The Second generation of FHE starts around 2011 from the collective efforts of many cryptographers such as, Zvika Bakerski, Craig Gentry, Vinod Vaikuntanathan, Frederik Vercauteren, and Junfeng Fan. Their innovations enabled the development of more efficient encryption schemes. A notable characteristic of these second-generation cryptosystems is their significantly slower noise growth during homomorphic calculations. They are also efficient enough for some real applications, even without invoking bootstrapping, but by executing operations applying leveled techniques. Despite these advances, all second-generation cryptosystems still rely on the basic Gentry's blueprint, first establishing a somewhat homomorphic scheme and then transforming it into a fully homomorphic cryptosystem through bootstrapping.

The BGV scheme [105] was developed by Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan in 2011 based on the hardness of the Ring-Learning with Errors (RLWE) problem and founded in techniques by Brakerski-Vaikuntanathan scheme (BV) [142]. What distinguishes BGV is how it handles encryption error or noise growth, especially after homomorphic multiplications.

In 2012, the Brakerski/Fan-Vercauteren scheme (BFV) scheme was introduced in two works, one by Barkerski and the other by Fan and Vercauteren. The BFV [143] was built using Brakerski' scale-invariant cryptosystem and an LWE-based scheme, and Fan and Vercauteren ported Brakerski's scheme to the Ring-LWE setting. A relevant aspect of the BFV scheme was its scale-invariant nature that unlike in BGV, the message bits are encoded into the higher-order bits of a plaintext [144].

Those cryptosystems are in the branch of leveled schemes, and their security relies on the hardness of the RLWE problem. There is another branch of schemes, but based on the N-th degree Truncated polynomial Ring Units (NTRU) computational problems, whose security considers the lattice problem of the shortest vector problem (SVP) [7]. Examples in this branch are the LTV [145] and the BLNN [146], which applies the scale-invariance cryptosystem of LTV and Brakerski [144]. Both LTV and BLNN were found to be vulnerable to sub-field lattice attacks and are no longer used in practice, but there were advances and several new NTRU-based schemes implemented, three to note are the NTRU Prime,

Flattening NTRU and the BQTRU [147, 148].

Third Generation FHE: The Third generation FHE started in 2013 with the Craig Gentry, Amit Sahai, and Brent Waters (GSW) scheme, a technique for building FHE schemes that avoid an expensive relinearization step in homomorphic multiplications. Zvika Brakerski and Vinod Vaikantanathan noted that the GWS cryptosystem exhibits an even slower noise growth rate for certain circuit types, providing better efficiency and robust security. Based on that, a very efficient bootstrapping technique was implemented by Jacob Alpering-Sheriff and Chris Peikert [103, 149].

These techniques were further improved to develop an efficient ring variant of the GSW cryptosystem; the FHEW proved possible to reduce the bootstrapping time to a fraction of a seconds by refreshing the ciphertext after each operation. It was achieved by a method to calculate boolean gates on encrypted data, resulting in a new implementation of bootstrapping [12, 108]. Based on the FHEW scheme, the TFHE scheme was introduced and further enhanced its performance by implementing a ring variant of the bootstrapping procedure [109, 150].

Fourth Generation FHE: Most fourth-generation encryption schemes are somehow derivated from CKKS scheme [99]. However, some sources also consider TFHE part of the fourth generation due to its continual improvements, such as the inclusion of leveled techniques [110].

CKKS [151] is a leveled scheme that also relies on LWE and RLWE hard problems. It stands out for its enhanced performance, achieved using approximate numbers instead of exact ones. Rounding operations help reduce noise in multiplication operations on ciphertext, decreasing the need for bootstrapping. A paper published in 2021 raised concerns about a potential vulnerability associated with the rounding approach in CKKS. It discusses passive attacks against CKKS and argues that the traditional formulation of Indistinguishability Under Chosen Plaintext Attack (IND-CPA) security used by CKKS is insufficient. They suggest that a stronger definition is necessary to properly assess the security of such schemes [152] .

The following excerpt from the paper *"Survey on Fully Homomorphic Encryption, Theory, and Applications"* presents some interesting conclusions on the current state and applications of the schemes and corresponding generations studied in this section , from which derivates the summarization in Table 3.1 and Figure 3.3 [14].

> "To the best of the authors knowledge, BGV [105], B/FV [143], TFHE [109] and CKKS [153] are the most practical and widely adopted schemes. The second generation schemes, BGV and B/FV, are suitable to work with finite fields in the modular exact arithmetic. They are equipped with efficient packing which enables the use of Single Instruction Multiple Data (SIMD) instructions to perform computations over vectors of integers. Thus, these schemes are excellent candidates when large arrays of numbers are to be processed simultaneously. Second-generation schemes are not good candidates for circuits where bootstrapping is re-

quired (circuits with large multiplicative depth) or where non-linear functions are to be implemented. Third-generation schemes should be adopted instead, namely TFHE, which can outperform previous schemes for bit-wise operations when computations are expressed as boolean circuits [154]. The main limitation of TFHE is the lack of support for CRT packing (batching). Hence, the scheme can be outperformed by previous approaches when large amounts of data are processed simultaneously. The fourth generation, CKKS, is the best option for real numbers arithmetic. Table 3.1 provides a comparison among the schemes' families, and , and the Figure 3.3 depicts the main applications for each generation of schemes. It is worth clarifying that although TFHE provides the fastest bootstrapping procedure, the batching feature of 2^{th} and 4^{th} generation schemes allows for the parallel bootstrapping of several plaintexts. For the specific case of CKKS, it is possible to obtain a more efficient amortized bootstrapping than for TFHE, this special case has been highlighted in Table 3.1 with the ■ symbol. This is however not true for 2^{th} generation schemes because the number of slots is significantly lower than for CKKS. For example, in BGV, the number of slots is only about 1000, compared to 2^{15} slots for CKKS. This renders CKKS bootstrapping more than one order of magnitude (often two) faster than BGV bootstrapping".

Table 3.1: Common properties of FHE schemes by generation, adapted from [14]

Scheme	Fast Operations			Fast packing/ batching	Leveled Design	Fast Bootstrapping
	scalar mult	arithmetic	non arithmetic			
2^{th}	■	■	□	■	■	□
3^{th}	■	■	■	□	■	■
4^{th}	■	■	□	■	■	■

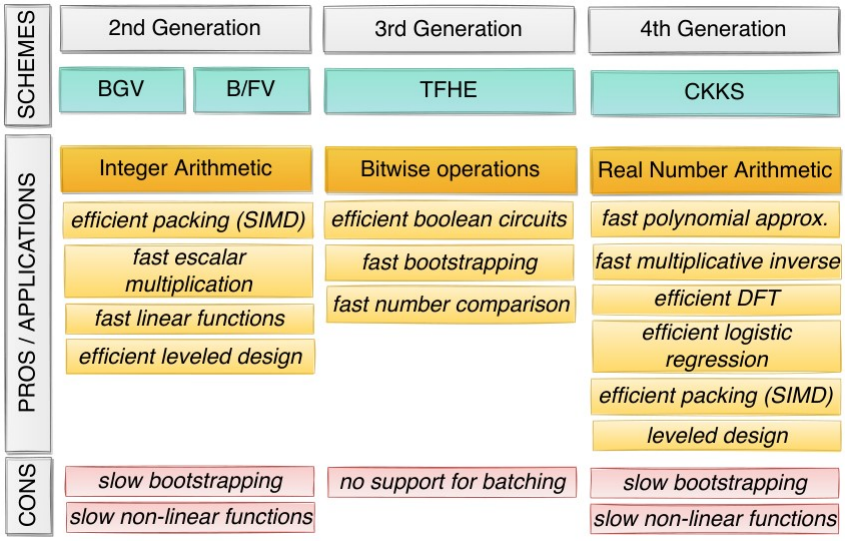


Figure 3.3: Pros/Cons of FHE schemes by generation [14]

3.3 Libraries and Compilers

This section overviews the most relevant works related to FHE libraries and compilers that are available to convert high-level programs to FHE-based implementations, contributing as a significant step towards making FHE accessible to non-cryptographers to design privacy-preserving solutions. FHE libraries have facilitated the advancement of FHE technology, allowing the implementation of various FHE applications without requiring a deep understanding of advanced mathematics and cryptography.

3.3.1 Libraries

FHE libraries' main objective is to make FHE scheme operations available via an Application Programming Interface (API). They incorporate features that allow ciphertext maintenance, manipulation, and homomorphic operation methods. However, the correct use depends on the developers, who must know what each API call entails in a given FHE-based solution.

Introduced in 2013 and built on the NTL library [155], HElib[1] was the very first FHE library [156], and implemented the Brakerski-Gentry-Vaikuntanathan (BGV) scheme. Then Microsoft also released the Simple Encrypted Arithmetic Library (SEAL)[2] in 2018 [157], which utilizes the Intel Homomorphic Encryption Acceleration Library (INTEL HEXL)[3], specifically focusing on Advanced Vector Extensions 512 (AVX512)[4] processors. As with the previous libraries that were written in C++ or C, we also have HEAAN [158], RNS-HEAAN [159], and the FV-NFLlib [160, 161]. Lattigo, unlike the others, was written using the Go language [162, 163].

Other libraries include, the CuFHE and NuFHE that are GPU-based libraries that implement TFHE in CUDA [164–166]. The TFHE library [109, 167] is an enhanced version of the FHEW library [168] that requires at least one Fast Fourier Transform (FFT) processor to run. Zama built Concrete[5] on top of a variant version of TFHE implemented in Rust language [169, 170].

In 2022, the authors of the library Palisade [171] that was originally developed by a DARPA consortium, started the OpenFHE[6] project. This project focuses on implementing APIs designed for usability, modularity, cross-platform support, and integration of hardware accelerators. It includes all relevant FHE schemes: BGV, B/FV, FHEW, TFHE, and CKKS [172].

The Table 3.2 lists several open-source FHE libraries, the language in which they are implemented, and related FHE schemes [14].

[1] https://github.com/homenc/HElib
[2] https://www.microsoft.com/en-us/research/project/microsoft-seal/
[3] https://www.intel.com/content/www/us/en/developer/articles/technical/introducing-intel-hexl.html
[4] https://www.intel.com.br/content/www/br/pt/products/docs/accelerator-engines/what-is-intel-avx-512.html
[5] https://docs.zama.ai/concrete
[6] https://www.openfhe.org

Table 3.2: Open-source FHE libraries [14]

Library	Language	Fast Operations					Last Update
		BGV	B/FV	FHEW	TFHE	CKKS	
HElib [173]	C++	X				X	01/10/2021
SEAL [157]	C++/C#	X	X			X	24/03/2022
PALISADE [171]	C++	X	X	X	X	X	30/04/2022
Lattigo [163]	Go		X			X	13/06/2022
FHEW [168]	C++			X			30/05/2017
TFHE [167]	C++/C				X		16/09/2021
Concrete [169]	Rust				X		10/05/2022
HEAAN [158]	C++					X	27/01/2022
RNS-HEAAN [159]	C++					X	26/10/2018
FV-NFLlib [160]	C++		X				26/07/2016
CuFHE [164]	Cuda/C++				X		09/02/2019
NuFHE [165]	Python				X		18/03/2020
OpenFHE [172]	C++	X	X	X	X	X	18/08/2022

3.3.2 Compilers

The libraries abstract complexity and speed up the process of writing FHE applications, but in recent years, another branch of tools emerged to further reduce the development effort. These tools are often called compilers and act as a type of API, just like the libraries. However a compiler is an additional abstraction layer that can facilitate tasks and optimizations that are still challenging even through a library [15]. Examples of compilers: Ramparts [174], Transpiler [175], and the compiler E^3 [176]. The Figure 3.4 illustrates the idea of layers presented in this section.

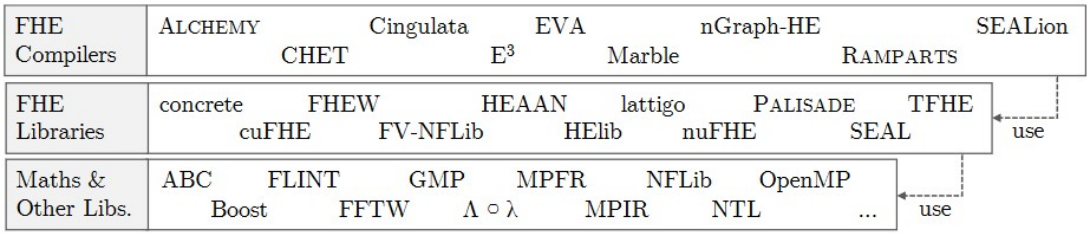


Figure 3.4: Layers representing the FHE tools space [15]

3.4 FHE Standardization and Post-Quantum Cryptography

Standardization is developing and implementing standards through consensus among several entities. The objective is to ensure consistent technology performance regardless of user, sector, and location. It is essential to allow the broad adoption of new technologies and interoperability, even when competing companies produce the same technology. Some organizations take over or centralize the standardization efforts. An example is what the IETF does for the Internet.

In 2017, a consortium involving academic researchers, government, and industry was initiated to work on the standardization of FHE under the initiative of Microsoft, IBM, and Duality Technologies [97]. Companies like Duality Technologies, Zama AI, and Cryptolab are actively working on deploying fully homomorphic encryption solutions, and major tech companies are also beginning to incorporate homomorphic encryption into their privacy-preserving products, such as Intel does by integrating Privacy-Preserving Machine Learning (PPML) to their Software Guard Extensions (SGX)[7] security solution [177].

In April 2021, the ISO/IEC Joint Technical Committee was working on a draft to add FHE to the standard ISO/IEC 18033, but it is marked as deleted on the current date. We could not identify if the project was canceled or if the document was moved [112].

Worldwide, the National Institute of Standards and Technology (NIST) is among the organizations that have the most influential role in cryptographic standards, including post-quantum cryptography (PQC). Post-quantum cryptography is a subset of cryptography that aims to develop secure cryptographic systems resistant to quantum computers, which poses a potential threat to traditional cryptographic schemes. NIST has been running a project to standardize post-quantum cryptographic algorithms [178, 179], having announced in 2022 the first four quantum-resistant cryptographic algorithms. Three are based on lattice hard problems, which endorses the potential of FHE [178]. For a more comprehensive approach to FHE standardization, the book *"Protecting privacy through homomorphic encryption"* is recommended [97].

3.5 Email spam and Anti-Spam

Email remains one of the most popular communication methods on the internet despite the increasing popularity of other social media platforms [180]. Kaspersky Lab and Cisco Talos report that spam emails make up between 50% and 85% of the total worldwide emails sent in a day, surpassing the mark of 200 billion [181].

Two literature reviews [181, 182] concluded that the most commonly used corpus used to train spam models not focused on phishing was Enron and Ling-Spam, and the most used ML algorithms were SVM, Naive Bayes, and Logistic Regression, all presenting the best performance in the reviewed works.

Traditional spam detection techniques, such as different types of blocklists, are still effective, but they are more prone to false positives. However, bayesian spam filtering was considered the most effective in this case. But when applied for mailing, machine learning-based techniques are usually static models and, therefore, more vulnerable to concept drift. Since 2017, there have been more efficient techniques to detect and mitigate concept drifting, such as incremental learning [180, 183].

Phishing has become the dominating type of spam; the Anti-Phishing Working Group (APWG) re-

[7] https://www.intel.com/content/www/us/en/products/docs/accelerator-engines/software-guard-extensions.html

ports that the number of phishing attacks increased by 250 000 from December 2020 to January 2021. In addition, the number of business impacted increased 56% from the last quarter of 2020 to the first quarter of 2021. The evolving sophistication of phishing makes it difficult to identify, even for cybersecurity experts, so they pay more attention to specific message elements to recognize phishing, such as message attachments and uniform resource locators. For phishing detection, deep learning algorithms present advantages because they can automatically extract Uniform Resource Locator (URL) features with good precision and without human intervention [184].

Supervised ML algorithms are the most commonly used among the identified in the survey on phishing attacks. To train the models, they use URL datasets, such as Phishload and Yandex. Hybrid models are trained using features such as HTML, email signatures, and JavaScript. The author concludes that Deep Learning algorithms are essential to detect phishing attacks due to the type of features present in the datasets [184].

Other studies address dataset shift or concept drift concerns, which can significantly affect model performance, leading to error rates as high as 48,81%. Supervised learning approaches assume that training and test data are drawn from the same distribution, making them more vulnerable. Different strategies are required to address dataset shift, combat adversarial manipulation, and intentional data modification to deceive classifiers. The concept drift or dataset drift in machine learning refers to the phenomenon where the underlying relationships and patterns within the data change over time. It decreases the performance of a machine learning model trained on historical data, which is often exploited by adversaries in ML model attacks. Adaptation techniques, such as online learning and ensemble methods, can mitigate the effects of concept drift [181, 182].

Among other common attacks on ML models are *"obfuscated word"* and *"text poisoning"*. Obfuscated word attacks change the text's appearance, making it more difficult for anti-spam filters to detect. It includes embedding special characters, using HTML comments, and modifying words without changing their readability. Text poisoning attacks may include inserting random words or commonly used legitimate terms into the email body, a technique known as *"word salad"*. These attacks evade detection by avoiding well-known spam words and adding noise to the messages. They also rely on the spammer's understanding of how the anti-spam filter operates and the recipient's ability to identify spam.

Similar to those attacks is *"text salting"*, which disrupts the functionality of spam filters by discreetly inserting random text into the background of an email. Spammers started applying this strategy in the mid-2000s across various communication platforms. They even started using image-based spam, which further disrupts text-based filtering methods [181]. Recent advances in Large Language Models (LLM) have also been applied in phishing attacks and in spam detection measures, creating new challenges but also potential solutions [185].

The Figure 3.5 aggregates data from 80 articles on spam detection published between 2017 and

2020, where (i) presents a graph with the number of times each ML algorithm was utilized in the reviewed articles and (ii) the overall highest performance obtained by each algorithm [16].

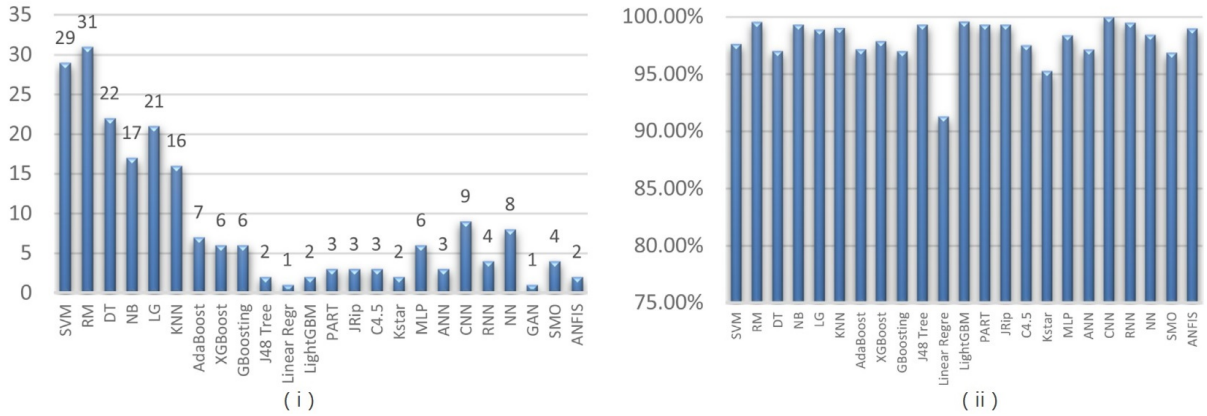


Figure 3.5: (i) Articles by utilized ML algorithm, (ii) Highest accuracy produced by each algorithm, adapted from [16]

3.6 HE Spam Detection Related Works

This section provides a review of existing literature analyzing the effectiveness of homomorphic encryption in the context of spam detection. The selected articles range from 2011 to 2022.

3.6.1 Privacy-Preserving Spam Filtering

This 2011 work was the earliest found with an explicit focus on using homomorphic encryption with anti-spam filters. The chosen classification model was gradient-based logistic regression. If the training data is separated, the gradients can be calculated individually and then aggregated by the server without compromising privacy [186].

The experiment used the dataset applied by the CEAS 2008 spam filtering challenge, which contained 3000 training emails and 3000 test emails manually marked as spam or ham. The metric chosen was the Area Under the ROC Curve (AUC), as it would facilitate measuring the classifier's performance at different precision-recall points corresponding to the classifier's reliability limit.

The experiment considered two groups: a set of users with access to private emails and the server responsible for training the classifier based on aggregated data.

The privacy protocol chosen is in the class of Secure Multiparty Computation (SMC), where multiple parties are required to compute a function that combines their individual inputs. The privacy constraint is that no party should learn anything about inputs belonging to any other party besides what can be inferred from the value of the function [187].

A prototype was implemented in C++ and utilized OpenSSL's variable precision arithmetic libraries to implement the Paillier cryptosystem [96], which allows computation over private data. The experiments were conducted on a 3.2 GHz Intel Pentium 4 machine with 2 GB of RAM.

The results demonstrated that the costs of encryption, decryption, and data transmission were linear in relation to the number of training instances and data dimensionality. Reducing dimensions was a crucial element for the proposed approach to perform well. Tests using a multi-threaded algorithm showed improvements of 6.3% on a machine with a single-core CPU, so there is potential for improvement if tested on multi-core architectures.

3.6.2 SHIELD: Scalable Homomorphic Implementation of Encrypted Data-Classifiers

This work published in 2015 describes a variant version of the GSW and BV schemes, which is used to implement a Bayesian anti-spam filter, a keyword searcher, and a secure binary decision tree using GPU parallelization. The classifier also implements Private Information Retrieve (PIR)[8] to obtain useful information from homomorphically encrypted data [188].

Regarding the classifier, the training phase was carried out on unencrypted data. Each word in the dataset had an associated probability of occurrence in spam emails, and the combined probability was used to define whether an email received was spam. As divisions with integers are expensive, the author modified Bayes' theorem so that homomorphic divisions are not used to obtain probabilities. Only words with a match in the database are counted in the probability calculation, thus reducing the volume of words. When emails are already encrypted, homomorphic operations are performed on the encrypted data obtained through a PIR scheme to combine words in the search.

Their GPU implementation outperforms the IBM HElib library, achieving a remarkable speedup of 1035× in Ctxt multiplication, an important bottleneck for most HE schemes. This performance improvement underscores the effectiveness of the approach.

3.6.3 Privacy Preserving Spam Email Filtering Based on Somewhat Homomorphic Using Functional Encryption

This 2015 work begins by exposing the growing need and motivations for delegating functionalities to third-party or cloud servers. Under this premise, the article compare the use of functional and homomorphic encryption to maintain the confidentiality of data stored on email servers [189].

In the context of email server security, the use of homomorphic encryption, as presented in this research, is instrumental in maintaining the confidentiality of the data on which the server performs homomorphic operations to detect spam. The functional encryption component, on the other hand, is

[8] PIR is a protocol that enables data retrieval without revealing what is retrieved, maintaining privacy over queries and requests.

designed to allow third parties to access specific information about the data on the server and serve a distinct purpose according to functional encryption protocols.

In the encryption process, the email server assumes the role of a semi-honest, or honest-but-curious, entity. It holds cryptographic keys from both the owners of the encrypted emails and third parties that perform functions on the data stored on the server, thereby ensuring a balanced and secure system.

In the authors' example, the email server has Alice's encrypted emails and a function that returns 1 when an email is spam. Bob, who is the administrator and has a key pair to use this function, provides it to the email server. Therefore, only Bob can use the function, getting results of just 0 or 1.

Problems highlighted by the authors:

- Make the process non-interactive and allow only one exchange of cryptographic keys.
- The number of functions applicable in functional cryptography still very restricted.
- The scheme used must be semi-homomorphic and not completely homomorphic.

Finally, the authors summarize the need to use both cryptographic approaches because homomorphic ciphers only allow marking emails as spam or not spam, and functional encryption could expand its application by allowing us to know whether a given email has been marked as spam.

3.6.4 i-SEAL2: Identifying Spam EmaiL with SEAL

End-to-end encryption during the email-sending process and anti-spam filtering are not possible locally on the users' machines. This would imply the need for each user to build their own classification models, which would have low accuracy, as they do not have a large and diverse enough volume of emails to train spam classifiers.

In this article from 2021, the author proposes a homomorphic encryption protocol to maintain user privacy when classifying emails as spam or ham. The proposal deals with a model based on machine learning without specifying an algorithm. Despite the article's title, the authors do not appear to have implemented an application or used the Microsoft SEAL library at all [190].

In the protocol described, the user sends an encrypted feature array to the provider; this vector is encrypted with a collective public key. Therefore, the content could only be exposed with collaboration between users and providers. After receiving the feature array, the provider classifies it as spam or non-spam homomorphically and returns the classified result to the user. The user's client can then decrypt the email with its secret key and perform the classification while preserving privacy.

The email provider periodically trains and updates the classification model using the encrypted feature arrays from users' emails. The server can also periodically decrypt the dataset with the collective public key, train and update the classification model, and send the model to users so that they can classify their emails locally. This approach can also be adapted so that users choose emails that are not sensitive and that could be made available to the provider in plaintext for training purposes.

The challenge pointed out by the author is the difficulty in encouraging providers to adopt this model, as today, they benefit from the analysis of data obtained in plaintext for commercial and advertising purposes. Other major obstacles this model highlights are the unavailability of end users and the management of keys for collective use.

3.6.5 Privacy-preserving spam filtering using homomorphic and functional encryption

This 2022 article describes an anti-spam filtering model on the email provider without decrypting message content through two different implementations, one based on homomorphic encryption and the other on functional encryption. Both approaches were evaluated for their performance, security, and accuracy [191].

According to the authors, the first works on using machine learning to implement anti-spam filters occurred between 1998 and 2000. Since then, it has become the most efficient way to classify spam. However, most existing solutions require users to reveal the content of their emails to the providers. To address privacy concerns, it proposes two solutions for email classification without decrypting the data.

The model comprises the sender user U_S, who wants to send a secure email to a recipient user U_R, which is encrypted in advance with public keys shared by both. For the email to arrive from U_S to U_R, the message first passes through the email provider SP so that spam emails can be classified and filtered out. Only emails identified as ham by SP are forwarded to U_R. SP is considered honest-but-curious; that is, it processes the input data, does not attempt to tamper with the processing, and passively observe the inputs, intermediate results, and outputs.

The classification occurs over vectors of integers that correspond to the feature arrays of the emails generated by U_S. The feature array generated for the email is always encrypted under different public keys and shared with the U_R. As SP is considered honest-but-curious, there would be no concern about tampering with the feature array. Therefore, it would be sufficient in this scenario to encrypt the vector with the same key provided by U_S to generate the encrypted version of the message.

The classification model used for both solutions is based on Qnet neural networks and Multilayer Perceptron (MLPnet). The model is fed with the feature array corresponding to the email, and its output generates a label indicating whether it is spam or ham. The main difference between the solutions is on how the classification result is treated.

In the homomorphic encryption solution, the classification result is unknown by SP, as only U_S and U_R keep the key, and the purpose is precisely to perform operations on the encrypted data without SP knowing the details of its content. Therefore, SP must send the encrypted result of the classification to U_R, which decrypts and return the result on-clear so that SP can then apply the anti-spam filter and determine whether the encrypted email is forwarded to U_R or discarded. It can be verified here that the

homomorphic encryption solution, although completely private, requires synchronous communication between SP and U_R.

The homomorphic prototype was developed using the BFV scheme through the SEAL library. It uses an approach called scale-invariant to reduce the noise accumulated in homomorphic multiplications. It also processes multiple plaintexts into one ciphertext to reduce complexity and improve performance.

As for the functional encryption solution, the classification result is obtained immediately, without additional interactions with U_D. This is possible because SP has a functional decryption key provided by U_D that SP uses to apply to the encrypted feature array. It provides SP with the intermediate result of the classification, post Qnet step. Then, the intermediate result is fed as input to the MLPnet algorithm, which returns the final label of the email as spam or ham.

The experiments used the TRECO7p, CEASS08-1, and ENRON datasets, which were preprocessed in advance for data cleaning and tokenization. Tests on the datasets were done separately, as each set represents a type of email. They also chose to use small feature arrays, as their size directly impacts the accuracy and performance of the algorithms. They measured this by varying the vector length between 2000 and 5000 positions.

The result analysis found that the homomorphic solution was the slowest to make predictions in each email, as the BFV scheme's operations increase overhead. The functional encryption had better performance without additional interactions between users and the server, but it presented data leakage at the stage in which the U_D sends the result on-clear back to SP. This vulnerability would allow an attacker to infer information about the content of the email.

The homomorphic encryption solution is safer, and the classification model never leaves the mail server. However, it requires more processing and is less efficient than the functional encryption solution in a spam classification scenario. Even so, there is room for performance improvements through hardware and software techniques, such as Craterlake [192] and F1 [193].

3.7 Discussion

Section 3.1 demonstrates potential applications for FHE in many sectors to protect data privacy during processing in cloud servers. With companies and governments adopting cloud computing, most of the population has private and sensitive data stored and computed in third-party servers. Such a tendency grows with broader access to faster internet connection technologies. This effect is verified by the growing predominance of video streaming services or even graphically demanding games played directly in the cloud [194].

In the scope of spam detection, Section 3.5 shows us that most traditional anti-spam methods are still used jointly for better results. However, the most recent advances in anti-spam measures come through

ML-based techniques. They are essential in countering the ever-evolving spamming techniques applied by adversaries to evade spam detection or for social engineering [195].

From the works in Section 3.5, and in Section 3.6, we can conclude that besides data privacy, FHE can be applied to help protect the machine learning models against attacks. When the model or its components, such as the weights, are homomorphically encrypted, they are less likely to suffer from adversarial concept drift. Adversarial concept drift is an attack where the adversarial infers information about the ML model and induces concept drift [196].

Section 3.1.1 described the application of FHE with machine learning based on schemes such as CKKS and TFHE, which are relatively recent advances. Reviewing the FHE progress in Section 3.2 has demonstrated significant performance enhancements since Craig Gentry's seminal work. Different branches of FHE, each with distinct properties and improvements, have provided new avenues for further research. For instance, schemes like TFHE have combined different methods to increase performance and control noise growth, such as by integrating leveled and bootstrapping approaches [110].

The promising outlook for FHE is also emphasized in Section 3.3 and Section 3.4. The complexity associated with developing FHE applications, a domain primarily accessible to mathematicians and cryptographers, has been substantially abstracted by the introduction of compilers and libraries. The standardization efforts involving several entities and sectors provide evidence of the growing interest in FHE applications. This brief literature review presented optimistic prospects for FHE but also underlined several challenges related to its performance, resource requirements, and implementation complexity compared to non-homomorphic counterparts.

3.8 Comparative Analysis

The research works in Section 3.6, which ranges from 2011 to the most recent in 2022, align with the present thesis by addressing spam detection solutions using homomorphic encryption. They are the most similar to FHE-SD and our purposes, but making a precise side-by-side comparison is impossible due to their motivations, approaches, and research advances since 2011. Therefore, we highlight individual aspects and contributions from each work and compare them to specific aspects of our proposed solution as follows.

1. Privacy-Preserving Spam Filtering (Section 3.6.1):

    (a) Contributions: Propose a protocol involving Secure Multiparty Computation (SMC) to address multiple parties' inputs, an aspect partially discussed in the FHE-SD solution. The Paillier cryptosystem also tends to have better performance and simpler implementation compared to FHE algorithms.

(b) Limitations: The Paillier cryptosystem is homomorphic only for addition operations and does not present the hardness of the underlying problems on which FHE algorithms are based. Therefore, it is not considered quantum-resistent and is less secure than FHE schemes.

2. SHIELD: Scalable Homomorphic Implementation of Encrypted Data-Classifiers (Section 3.6.2):

   (a) Contributions: Implement a variant solution of the GSW and BV schemes utilizing GPU parallelization, presenting outstanding performance improvements when compared to the version of IBM HElib available by 2015.

   (b) Limitations: Their low-level approach to implement an enhanced version of GSW and BV algorithms requires advanced cryptographic knowledge. The experiments are also conducted using specialized hardware (GPU). In contrast, our FHE-SD solution is designed to be accessible to non-cryptographer developers by using libraries that abstract the complexity of the FHE scheme and focus on general-purpose hardware.

3. Privacy Preserving Spam Email Filtering Based on Somewhat Homomorphic Using Functional Encryption (Section 3.6.3):

   (a) Contributions: Discuss a set of protocols using both HE and Functional Encryption (FE).

   (b) Limitations: No implementation, mainly speculates based on other works.

4. i-SEAL2: Identifying Spam EmaiL with SEAL (Section 3.6.4):

   (a) Contributions: Brief overview of HE spam detection implementation aspects and challenges.

   (b) Limitations: No implementation, lacking references to other works and a meaningful purpose.

5. Privacy-preserving spam filtering using homomorphic and functional encryption (Section 3.6.5):

   (a) Contributions: Implement FHE and FE solutions for spam detection, comparing each approach with in depth analysis of performance and security. Also provided detailed setup and presented results for three different training datasets.

   (b) Limitations: This is the most recent and rich article, however it focuses mainly on comparing the FHE and FE solutions and pointing out their strenghts and weaknesses.

Compared to these related works, our contributions go beyond implementing and measuring side-by-side FHE and on-clear equivalent spam detection algorithms based on four machine learning models. We provide a more comprehensive range of metrics for reliable evaluation and conclusions on the feasibility of FHE. This work also describes in more detail the developments from the early stages to obtain insights on the current effort required to implement real-world FHE applications using tools that abstract FHE's inherent complexity. That way, we demonstrate that most developers can implement FHE solutions with

reduced effort and on more affordable general-purpose hardware. These are critical aspects for the broad adoption of FHE and for turning it into a cybersecurity alternative outside the academic scope.

Summary

This chapter explored the development of Fully Homomorphic Encryption (FHE) schemes from the 1^{th} to the 4^{th} generation. Covering articles on practical applications of FHE, main FHE libraries and compilers, and ongoing standardization efforts in the area. Section 3.5 examined articles on the current state of anti-spam methods and on spam detection with homomorphic encryption. In Section 3.7, we discussed the literature reviewed and compared this thesis's contributions against the similar works addressed in Section 3.6.

4

Proposed Solution

Contents

The proposed solution to achieve this thesis's objectives is the FHE-SD application, which implements a set of spam detection algorithms based on machine learning, enabling us to execute experiments and comparisons on FHE and on-clear scenarios. This application was developed utilizing the Concrete compiler, which is based on TFHE-rs libraries written in Rust, and Concrete-ML (CML) libraries written in Python. FHE-SD supports four different machine learning algorithms for spam detection on FHE and their conventional non-encrypted counterparts using scikit-learn, also providing performance and resource usage metrics. We sought to make the scikit-learn and Concrete-ML versions of the machine learning algorithms parameterized and implemented under the most equivalent conditions possible to present meaningful results and comparisons, limitations to achieve such purpose are discussed in Chapter 5.

Although the primary purpose of FHE-SD is experimental, it also supports Linux-like command line instructions, including a help manual for testing and real modes. The code has been thoroughly documented using Sphinx[1] and developed in Python. As a result, FHE-SD is designed to be installed on different machines, and the comprehensive documentation can help reproduce the experimental setup presented here. The FHE-SD usage documentation is available in Appendix A, and there is also a GitLab repository for the FHE-SD project[2].

4.1 FHE-SD Objectives

FHE-SD implements the supervised learning workflow described in Section 2.5 and illustrated in Figure 2.10. Besides, it also enables the effective utilization of trained models to predict whether email messages provided as input are spam or ham. FHE-SD supports LinearSVC, Logistic Regression, Decision Trees, and XGBoost algorithms. These algorithms are available in the FHE versions, using the Concrete-ML libraries, and the conventional, non-encrypted or on-clear versions, using the scikit-learn libraries. As a result, there are two distinct implementations for each of the four ML algorithms, one for FHE and another for its on-clear counterpart.

FHE-SD supports two modes:

- **Test mode:** This mode executes the complete ML workflow, from dataset preprocessing to model training. By predicting the results from the trained model, we obtained part of the performance metrics required for the experimental objectives. In test mode, FHE-SD takes as input a dataset of 6000 emails labeled as spam or ham and generates a labeled feature array that is later split, where 80% is for model training and 20% for testing (4800/1200). The testing portion is input to the trained model, and the performance metrics are obtained from calculations over the prediction results and the labeled testing feature array.

[1] https://www.sphinx-doc.org
[2] https://gitlab.com/adrianorp/project-fhe-sd

- **Real mode:** A single email file functions as input, producing a feature array for a pre-trained model loaded from disk. Although not directly pertinent to our research objectives, the *"real mode"* could be valuable for future enhancements to a production application or as a potential subject for future work.

In addition to the listed features, the FHE-SD package includes supplementary scripts for generating additional metrics and simulating hypothetical network interactions, as outlined in Section 4.3. These scripts can be found inside the directory "test" within the FHE-SD deployment package. Since these scripts are standalone, they must be manually adjusted and executed individually to obtain the desired outcome.

The objective of FHE-SD is to compare the performance of ML-based spam detection algorithms in both FHE versions and on-clear versions under similar conditions. We also want to provide insights into the feasibility of using FHE for spam detection in real-world scenarios, even with constrained hardware resources and without dealing directly with the inherent complexity of the underlying FHE scheme.

4.2 Architecture

The FHE-SD is implemented using the Concrete-ML, which supports deploying systems under the client-server model, as per Figure 4.1.

FHE-SD considers three possible actors:

- **Developer Machine (DM):** DM is where the developers train and compile the ML model and generate the deployment files necessary for the client and server machines.
- **Server Machine (SM):** SM is an email provider where the compiled ML model detects spam over encrypted emails.
- **Client Machine (CM):** CM is the user's machine, where a secret key and an evaluation key are generated. CM converts the emails into an FHE encrypted feature array, and it is sent along with the evaluation key to SM to classify the emails.

The deployment files are composed of the following:

- **Server.zip:** Created and provided by DM, contains the compiled ML model that is machine-architecture specific (x86 in our case), JSON files with parameters, and client specifications. They are shared with the SM and when the ML model is compiled, it is converted to an equivalent FHE circuit.
- **Client.zip:** Created and provided by DM, contains JSON files with secure cryptographic parameters used by CM to generate the keys and quantization parameters.

As presented in Figure 4.1, the development machine (DM) trains the ML model and generates the deployment files. Then, DM deploys the compiled model and the *server.zip* file to the email provider or

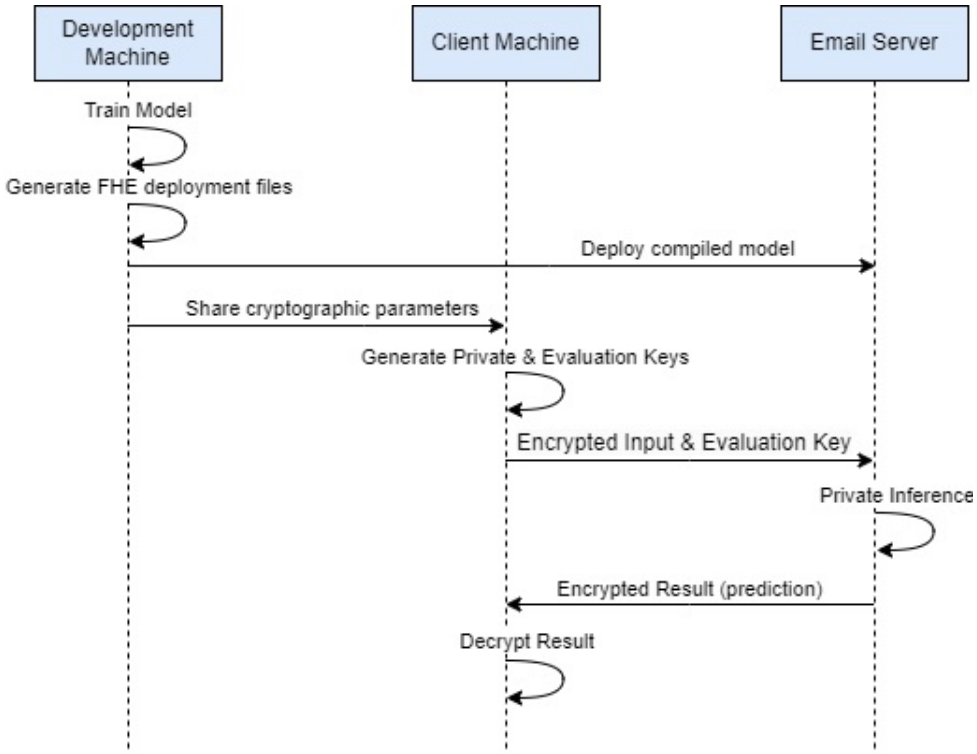


Figure 4.1: FHE-SD Client-Server model

server machine (SM) and shares with the client machine (CM) the *client.zip* file containing cryptographic parameters to generate the key pair. The CM generates a key pair composed of secret and evaluation keys. CM encrypts the email feature array with the secret key, then sends the encrypted input along with the evaluation key to SM. SM receives the encrypted input and uses the compiled model with the evaluation key to run the private inference over the FHE encrypted input. The resulting prediction is unknown to SM since it remains encrypted throughout the process. SM finally sends the encrypted prediction back to CM, where it is decrypted by the secret key to verify if the email is spam.

Concrete Architecture

Concrete is a framework developed by Zama that works on top of the Concrete compiler, which uses LLVM[3] and MLIR[4] to translate Python code into FHE programs at reduced development effort. Concrete is based in the rust implementation of the TFHE scheme for boolean and small integer arithmetics over encrypted data [17].

The Figure 4.2 describes the Concrete architecture presented in stack layers, where FHE-SD directly uses libraries in the *applications* and *transpilers* layers. The Concrete compiler translates the coding

[3]https://en.wikipedia.org/wiki/LLVM
[4]https://mlir.llvm.org

instruction through MLIR to the TFHE rust libraries, where the cryptographic primitives and operations are implemented and executed by the CPU. Among the third-party entities we have libraries such as scikit-learn[5], PyTorch[6], and XGBoost[7], that are used in the Concrete-ML machine learning algorithms and also in the FHE-SD implementation.

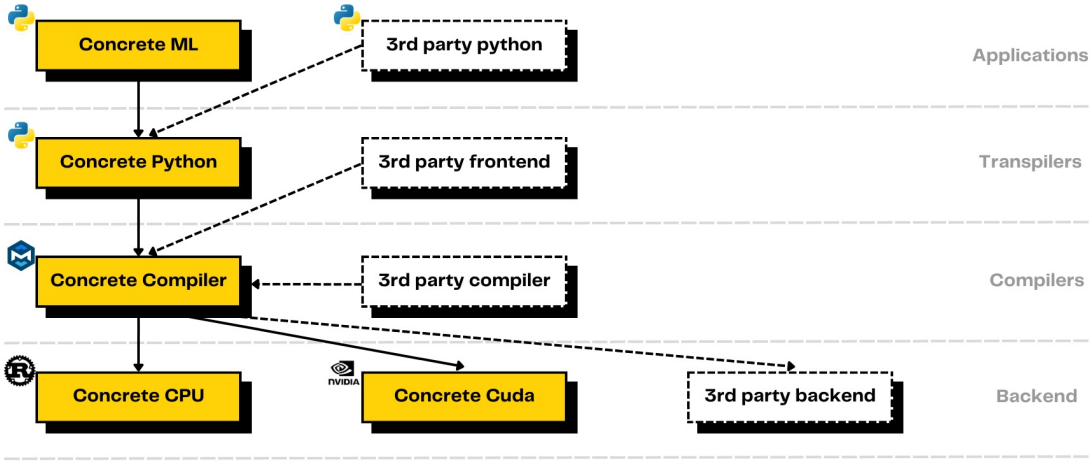


Figure 4.2: Concrete Architecture [17]

Concrete has GPU acceleration capabilities for CUDA[8]. However, FHE-SD supports only the *Concrete CPU backend* and makes no active thread management besides what is already provided automatically by the Concrete compiler.

4.3 Experimental Environment

Limited hardware resources could be an advantage when analyzing the feasibility of FHE for spam detection. FHE inherently requires significant CPU power and memory usage; for instance, the FHE encrypted data from our tests was three times larger than their plaintext equivalent.

The experiments were performed on a Linux Virtual Machine (VM) running on the Microsoft Hypervisor (Hyper-V). The Linux VM system information is detailed in Table 4.2, and the system information for the host machine is available in Table 4.1.

4.4 Application Development

FHE-SD is designed to output metrics and experimental data on spam detection over plaintext and FHE ciphertext, supporting real and testing modes. However, it has limited support in real mode, where the

[5]https://scikit-learn.org
[6]https://pytorch.org
[7]https://xgboost.readthedocs.io
[8]https://www.zama.ai/post/concrete-core-v1-0-gamma-with-gpu-acceleration

Table 4.1: Host System Information

Item	Specification
Architecture	x86_64
CPU	Intel Core i7-8550U
CPU Frequency	1.80GHz
N^o CPU Cores	4
N^o CPU Threads	8
System Memory	16GB DDR4
Operating System	Windows 10 22H2(19045)

Table 4.2: VM (Hyper-V) System Information

Item	Specification
Architecture	x86_64
CPU	Intel Core i7-8550U
CPU Frequency	1.80GHz
N^o CPU Cores	4
N^o CPU Threads	8
System Memory	8GB DDR4
Operating System	Kali GNU/Linux 2023.3

machine learning model is pre-trained, loaded from disk, and used to classify individual email messages as spam or ham - taking as input a single text file per email. The application is fully equipped with features focused on testing scenarios and metric extraction, from which the results presented in Chapter 5 are obtained.

Figure 4.3 provides a detailed description of the machine learning workflow for FHE-SD execution in testing mode. The bluish shapes illustrate the execution flow for traditional machine learning, using scikit-learn library on plaintext (on-clear). The reddish shapes are the processes for the execution in FHE mode, where the input data is encrypted using Fully Homomorphic Encryption. The green shapes denote the initial execution stages, shared by both FHE and on-clear modes, where the dataset is preprocessed and the models trained. Further details on the preprocessing methods are presented in Section 4.4.3, and information about the email corpus utilized as dataset can be found in Section 4.4.2.

FHE-SD is executed using command lines compatible with Linux-like systems and requires the following basic inputs:

- **Type of scenario:** Specify whether the execution is for testing or a real scenario.
- **Dataset path:** Provide the path to the directory where the email files are stored.
- **ML Algorithm:** Choose the machine learning algorithm to train the model and execute the inference.
- **Encryption mode:** Specify whether the execution should be performed with or without encryption.

This experiment utilizes a portion of the Enron corpus as a dataset. The dataset is preprocessed to enhance data quality, extract features, and divide the resulting feature array into test and train datasets.

When FHE-SD is executed in clear mode, the trained model is loaded, and a copy of the model is saved to disk. The test dataset is fed into the model, which returns predictions that are processed to obtain the performance metrics.

In FHE mode, the trained model is first compiled to generate its FHE circuit, but it does not secure the model architecture, because the model weights and parameters are not encrypted. The deployment files are then created, with the file *server.zip* stored in a directory representing a server and *client.zip* in another

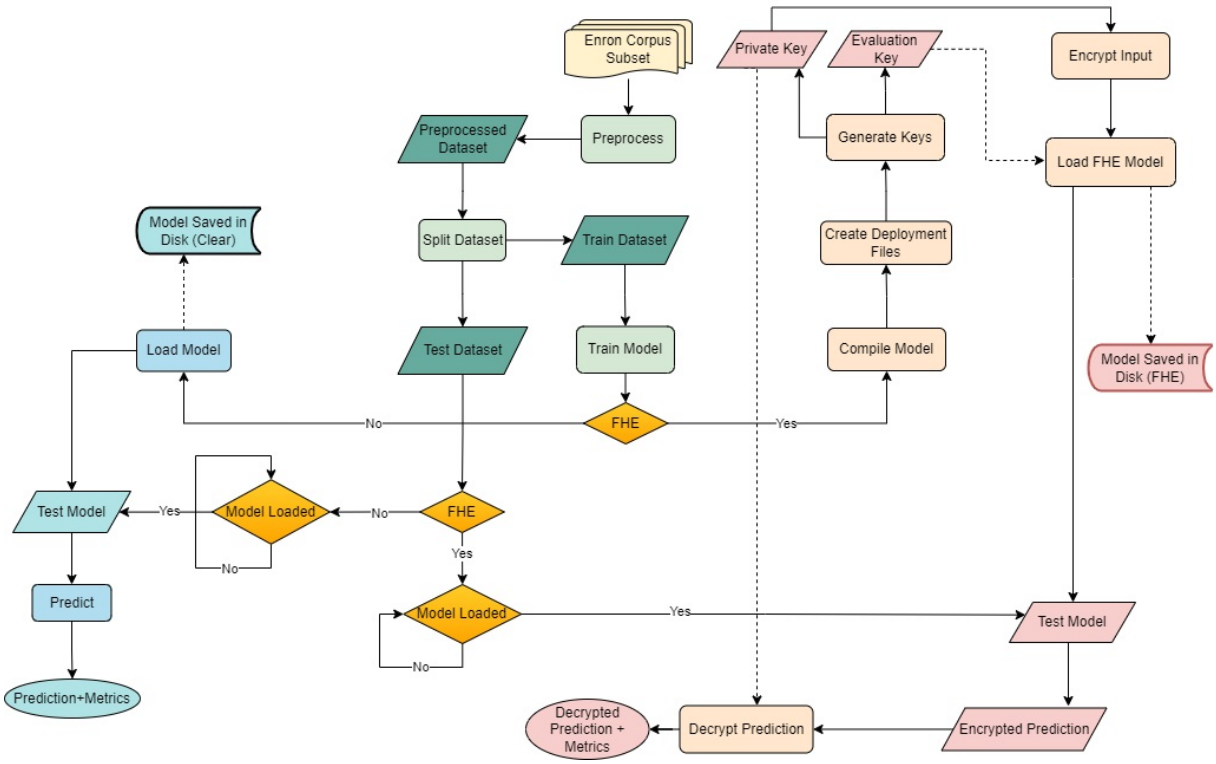


Figure 4.3: FHE-SD Design Details

directory representing a client. The client utilizes the *client.zip* file to generate the secret and evaluation keys.

The secret key encrypts the feature array of the test data, and then the encrypted data is used as input to the compiled model. The evaluation key is then applied to the compiled model to predict over the encrypted feature array. Finally, the model outputs the encrypted prediction, which is decrypted by the client using the secret key. From the decrypted results derive the performance metrics that are detailed in Section 4.4.4 and a copy of the FHE model is also saved to disk and can be reloaded for use in real mode.

4.4.1 Standalone Scripts

This section describes the standalone scripts necessary to configure the FHE-SD or extract specific metrics outside the primary workflow in Figure 4.3.

- **create_config_ini.py:** This script can be edited to customize FHE-SD parameters, such as the directories' path and constant values. It has to be manually executed to generate the file *config.ini*, from where global parameters are retrieved automatically.
- **read_config_ini.py:** This module contains Python functions to read the parameters stored in file *config.ini*. Several other modules and scripts in FHE-SD utilize these functions.

- **config.ini:** This file contains constants and parameters used by FHE-SD. It is generated by the script *create_config_ini.py* and reading routines are implemented in file *read_config_ini.py*.
- **network_simulation.py:** This script simulates a complete network flow for spam detection using FHE-SD, detailed in Section 4.5.

The following scripts require manual adjustments by setting $mode = fhe$ for metrics over FHE or $mode = clear$ for metrics on-clear.

- **Utils_Calibration_Plot_Multiple.py:** This script plots the calibration curve for the different ML algorithms.
- **Utils_ConfusionMatrix_Plot_Multiple.py:** This script plots the confusion matrixes for each algorithm.
- **Utils_ROC_AUC_Plot_Multiple.py:** This script plots the ROC curve and AUC for the ML algorithms.

4.4.2 Enron Email Dataset

This work focuses on FHE, but it is important to ensure a certain level of quality in the ML training, which is directly influenced by the dataset used. Ideally, the training should be conducted using a large dataset composed of different email corpus. However, due to limited resources and the fact that the ML models trained do not need to achieve perfect results to evaluate the FHE application, we have opted to use only the Enron-Spam[9] dataset.

The Enron-Spam dataset contains the mailboxes of five Enron employees, in raw and preprocessed versions. We specifically use a portion of the preprocessed emails in which unnecessary data, such as HTML tags and message headers, were already removed. The messages are saved in separate text files and stored in *"ham"* or *"spam"* directories representing the email category or class. More details on the original dataset can be found in the paper published by Metsis, Androutsopoulos, and Paliouras [197].

The subset utilized in the experiment consists of six thousand emails, detailed in Table 4.3. These emails required additional preprocessing to enhance data quality, remove remaining stopwords, and discard emails without a message body.

4.4.3 Preprocessing and Feature Extraction

This section describes the preprocessing and feature extraction techniques used in FHE-SD, corresponding to the green shapes processes in Figure 4.3. FHE-SD receives as input the path to the directory where the text files representing the emails are stored.

[9] https://www2.aueb.gr/users/ion/data/enron-spam/

Table 4.3: Enron Corpus - Email distribution and owners

Owners (ham + spam)	**ham:spam**
farmer-d + GP	500:500
kaminski-v + SH	500:500
kitchen-l + BG	500:500
williams-w3 + GP	500:500
beck-s + SH	500:500
lokay-m + BG	500:500

The preprocessing uses NTLK[10] libraries to support tokenization and removal of stopwords, approaches discussed in Section 2.5.2. Through the Punkt[11] NTLK module the tokenization targeted english words. However, besides Enron Corpus being from the extinct American company, the sampling had a few instances of emails in portuguese. During feature engineering, it is essential to extract the most representative features, those that are meaningful in the dataset and for classification purposes. Therefore, we removed strings such as stopwords and words that were rarely used in the dataset.

FHE-SD also removes strings shorter than three characters and emails with fewer than four lines, because the dataset has many files where the third line contains a reference to attachments, we also ignored the subject lines and considered only the content of the bodies of the emails. Some anti-spam solutions consider the subject of the emails for spam detection. However, it requires specific and sometimes more complex approaches, for example, calculating probabilities only over the subjects, then over the body of the emails, and finally pondering the distinct outcomes. It could even consider concurrent models, one trained over email subjects and the other on the emails' content, and infer over both resulting probabilities.

In the *config.ini* file, it is possible to define the maximum number of features to be utilized during the training process. A more significant number of features, up to a specific limit, can improve the accuracy of the model predictions, but it implies increased processing time and memory usage. After testing with different values, we found that 5000 features were the optimal value for balancing accuracy and performance. To select the relevant features, we extract a list of the most common words, from which we retrieve 5000 words. The final product of the preprocessing is the feature array exemplified in Figure 4.4(a) and the label array exemplified in Figure 4.4(b).

The features matrix $F_{i,j}$ used in this experiment is described by Equation (4.1). The upper limit for i reflects the size of the sampling, where we have 6000 emails, and the upper limit for $C_{i,j}$ and j could vary if we change the maximum number of features defined in *config.ini* file.

[10] https://www.nltk.org/
[11] https://www.nltk.org/api/nltk.tokenize.punkt.html

$$F_{i,j} = \begin{pmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,j} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,j} \\ \vdots & \vdots & \ddots & \vdots \\ c_{i,1} & c_{i,2} & \cdots & c_{i,j} \end{pmatrix} \tag{4.1}$$

Where:

$$F_{i,j} \begin{cases} C, i, j \in \mathbb{Z} \\ 0 \leq i \leq 6000 \\ 0 \leq j \leq 5000 \\ 0 \leq C_{i,j} \leq 5000 \end{cases}$$

6,000 rows × 5,000 columns

	0	1	2	3	4	5	6
0	1	1	1	1	4	1	2
1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0
6	0	0	0	0	1	0	0
7	0	0	0	0	1	0	0
8	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0

(a)

6,000 rows × 1 columns

	0
0	1
1	1
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	1
10	1
11	1
12	1
13	1

(b)

Figure 4.4: Features Array(a) and Labels Array(b)

The words, or tokens, are the features extracted from the dataset, each individual feature is converted in an unique integer number ranging from 0 to 5000. This numeric representation of each feature is accessed through the index of the columns j in the feature array F. At the same time, the individual email samples are also identified with unique numbers, which are the row indexes (i), as in Figure 4.4(a). While $C_{i,j}$ has the counting of occurrences of the feature j in the email sample i. For example: in the given feature array, where $0 \leq j \leq 5000$, the word *"com"* is identified by an integer $j = 4$ and it has one occurrence in two email samples identified by the coordinates $F[6, 4]$ and $F[7, 4]$.

The label array in Figure 4.4(b) is used for the supervised learning procedures, measuring the performance of the models, and extracting several metrics such as the accuracy. This is done by comparing the models' prediction with the label array, which has labels indicating whether each email is spam or ham. The picture shows 13 email samples identified by the first column and the second column filled with *ones*, indicating that those samples are all spam - ham samples would be labeled with *zeros*.

The module *utils.py* in directory "common" contains most of the FHE-SD preprocessing source code and can be seen in the documentation available in Appendix A.

4.4.4 Machine Learning Models and Performance Metrics

FHE-SD supports four machine learning algorithms: LinearSVC, Logistic Regression, Decision Tree, and XGBoost. Initially, we also had k-Nearest Neighbors (KNN) and Random Forest as candidates, but both were discarded. KNN presented low accuracy results due to the dimension and distribution of our dataset. The Random Forest took days to finish the inference and training processes, also showing poor accuracy, which was caused by the high depths of the tree ensemble. So, we replaced Random Forest with the XGBoost algorithm because it has better performance and requires lower depths[12] to achieve good accuracy. A brief description of each ML algorithm implemented in FHE-SD is available in Section 2.5.3.

The experimental metrics generated by FHE-SD are the primary artifacts to evaluate the FHE application for spam detection, compare against traditional approaches, and analyze its feasibility. FHE-SD considers the execution duration for the following processes or states, where items 3, 5, 6, 7, 8, and 10 are specific to FHE scenarios:

1. Feature extraction.
2. Model training.
3. FHE Model compilation.
4. Model loading.
5. FHE files deployment.
6. Load FHE client files.
7. Public-key generation.
8. Feature array encryption.
9. Prediction or inference.
10. Decrypting the prediction.

To measure the machine learning algorithms' performance, we used Accuracy, F-score, Precision, Recall Rate, and confusion matrixes; the theory behind these metrics is explained in Section 2.5.1. Besides these outputs, the stand-alone scripts mentioned in Section 4.4.1 generate graphics for each ML model: Roc curves and AUC, calibration plots, and confusion matrixes. We also have registered the memory peaks utilized by the application processes and a comparison of the data size for on-clear against FHE encrypted data.

4.5 Network Simulation

The FHE-SD has a standalone script that simulates an entire network interaction for spam detection, similar to what we see in Figure 4.1, using Concrete features for client-server interactions. Due to resource limitations, we could only emulate the data transfer in a network through three distinct local directories representing: the Development Machine (DM), Client Machine (CM), and Server Machine (SM). The transference of data and files from one directory to another is an analogy to the traffic in a network. Those directories can have their path defined in the *config.ini* file. The default name for the DM directory

[12] Lower depths imply fewer layers or levels in each tree structure.

is *"development_machine"*, for SM is *"server_machine"*, and for CM is *"client_machine"*. See Appendix A for more details on this setup.

Before describing the simulation steps, we have the following assumptions in the case of a real scenario through a network:

- The connections occur on top of TCP/IP protocols as described in Section 2.1;
- SM is an email provider with SMTP support, details in Section 2.1.
- CM has MTA and MUA, both approached in Section 2.1, enhanced to support FHE routines.
- SMTP Envelopes received by CM, whose definition is in Section 2.1, are scanned in advance for viruses before the feature arrays are generated and encrypted.
- The role of DM could be delegated to SM to simplify the setup. However, the email provider would have more privileges than necessary, increasing the risk of users' privacy being violated.

Having the premisses defined, the standalone script *network_simulation.py* in directory "test" executes the following steps to simulate FHE client-server interactions over a network for an FHE model trained using LinearSVC algorithm:

(1) DM train and compile the ML model.

(2) DM generates the deployment files: *client.zip*, *server.zip*, and *version.json*.

(3) DM transfer *client.zip* to CM.

(4) DM transfer *server.zip* to CM, which also contains the compiled model.

(5) CM generates the secret and evaluation keys.

(6) CM sends the serialized evaluation key to SM.

(7) CM receives the email, or SMTP Envelope, and preprocess the content to generate the feature array.

(8) CM uses the secret key to encrypt the feature array and send it encrypted to SM.

(9) SM receives the encrypted feature array, and without decrypting it, SM uses the evaluation key to run the inference.

(10) From the inference, SM obtains the encrypted prediction result, which SM cannot decrypt.

(11) SM sends the encrypted prediction result to CM.

(12) CM uses the secret key to decrypt the result and learn if the email is spam or ham.

4.6 Limitations and Challenges

When the research and implementations for this thesis started in 2023, Concrete-ML (CML) was at version 1.3, with tests on tree-based algorithms presenting severe performance issues. For example, FHE-SD took up to four days to finish the FHE processing using the Decision Tree algorithm, which was almost discarded. Such an issue was not observed in the linear ML algorithms because they usually do

not require TLU operations - at least not for the two algorithms tested.

In CML 1.4, optimizations for Table Lookup (TLU) operations for CML trees-based algorithms were introduced, mainly achieved through rounded table lookups. This approach rounds the least significant bits of large integers and then applies the table lookup on the smaller resulting values. The discussion on Table Lookup can be found in Section 2.7.8.

The rounded TLUs reduced the execution time for tree-based algorithms from four days to ten hours on average. But the new CML version caused runtime errors on the linear algorithms during FHE prediction. Initially, we assumed that the defect was in FHE-SD, from inputs causing overflows in the Concrete compiler. However, we could not identify any specific problems during debugging. As an alternative to progress with the work, we decided to provisionally use CML 1.3 for the experiments on linear algorithms and switch to CML 1.4 for tree-based algorithms.

As previously mentioned, in test mode, the FHE-SD creates a feature array that includes all emails from the dataset. We were concerned about the potential high memory usage during prediction if we input the entire encrypted feature array simultaneously. We addressed that by encrypting and inputting each row of the feature array into the ML model individually. After obtaining the resulting prediction for a row, the next row is encrypted and fed as input to the model.

We speculate that since the model was trained using the entire array, once compiled using the defective CML versions, it caused a post-processing shape mismatch for the linear models when the input was provided through individual rows of the encrypted feature array. Anyhow, we did not succeed in implementing batch inference to test that theory, possibly because Concrete does not support it at all.

Zama fixed the defect with the release of CML 1.6, just in time for the tests and evaluations to be re-executed using the latest version available to this date (1.6.1). A final note to consider is that Zama added to Concrete-ML support for training models on encrypted data from version 1.4. Given the implementation of FHE-SD predating CML 1.4, there is potential for future exploration of FHE model training on encrypted data.

5

Evaluation

Contents

This chapter evaluates the experimental process described in Chapter 4, comparing the prediction results for spam detection using traditional ML algorithms from scikit-learn with their FHE equivalent from the Concrete framework. The analysis comes through the performance of the ML models, memory usage, and processing time in distinct execution stages. We also explore the effort required from developers to create and deploy real-world FHE solutions utilizing low-power hardware without in-depth knowledge of the associated mathematical and cryptographic principles.

5.1 Evaluation Methodology

The evaluation methodology can be divided into two aspects: functionality and performance. The functionality aspect considers whether the FHE-SD application achieves its purpose as a spam detector.

The performance aspect considers the following:

- **Model Performance:** Metrics to compare the degree of success of the ML model's inferences or predictions.
- **Runtime Performance:** Measure the execution times and memory usage by FHE-SD for the supported algorithms.

The primary objective is to assess how FHE-SD performs using Fully Homomorphic Encryption against scenarios where FHE cryptography is not applied. The machine learning models are the main subjects of the experiment, helping evaluate the application of Fully Homomorphic Encryption and guiding our conclusions.

5.2 Preprocessing Results

Despite FHE being the focus of this work, we tried to achieve acceptable results during the machine learning stages, such as preprocessing and feature extraction, so FHE-SD could output results that are closer to a real-world solutions. Therefore, the resulting data from the preprocessing stage is critical to train the ML models correctly, directly impacting the subsequent stages.

The *"wordclouds"* in Figure 5.1 shows a snapshot of the most relevant features selected during the preprocessing, similar to what we have described in Section 4.4.3, which also considers the theoretical aspects presented in Section 2.5.2. Figure 5.1(a) has some of the most representative features in the set of 5000 words. Figure 5.1(b) shows only representative samples of words from ham emails where we see words such *"meeting"* and *"gas"* that are consistent with the kind of emails that would be exchanged in Enron's type of business. Figure 5.1(c) shows representative features extracted from spam emails, such as the word *"Viagra"*, which is very common in spam.

(a) (b) (c)

Figure 5.1: Most Representative Features

To help detect and prevent overfitting and underfitting, we applied the scikit-learn function *train_test_split* to divide the feature array into two subsets, 80% for training and 20% for testing. That function implements a pseudo-random algorithm to sort the data, which we used with a fixed salt value (random_state parameter) to ensure that the data split was reproducible across the different executions. Therefore, from the 6000 email samples, 4800 emails were used for training and 1200 for testing. For real scenarios using single email inputs, FHE-SD still requires improvements to address features missing in the training dataset, and larger datasets are required so the ML models can have proper generalization.

5.3 Models Performance

This section presents metrics to evaluate how well the models perform in making classifications or predictions. We applied seven distinct metrics for which the theory was presented in Section 2.5.1: accuracy, precision, recall rate, F-score (F1), confusion matrix, and two graphical representations using the ROC and Calibration curves. Most FHE and on-clear scenarios achieved identical results on the classification metrics, as presented in Table 5.1 for FHE and in Table 5.2 for on-clear scenarios - summarized in Figure 5.2.

Table 5.1: FHE Performance Metrics

	LinearSVC	Logistic Regression	Decision Tree	XGBoost
Accuracy	0,954	0,964	0,881	0,866
Precision	0,952	0,958	0,874	0,823
Recall Rate	0,956	0,972	0,894	0,935
F-Score	0,954	0,965	0,884	0,876

Table 5.2: On-Clear Performance Metrics

	LinearSVC	Logistic Regression	Decision Tree	XGBoost
Accuracy	0,954	0,964	0,880	0,866
Precision	0,952	0,958	0,872	0,823
Recall Rate	0,956	0,972	0,893	0,935
F-Score	0,954	0,965	0,882	0,876

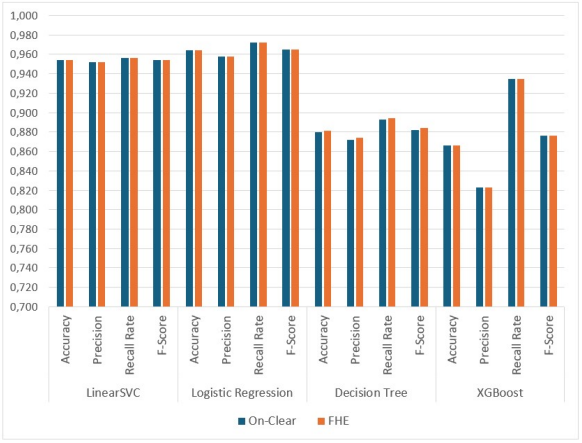


Figure 5.2: Algorithms Machine Learning Performance Metrics

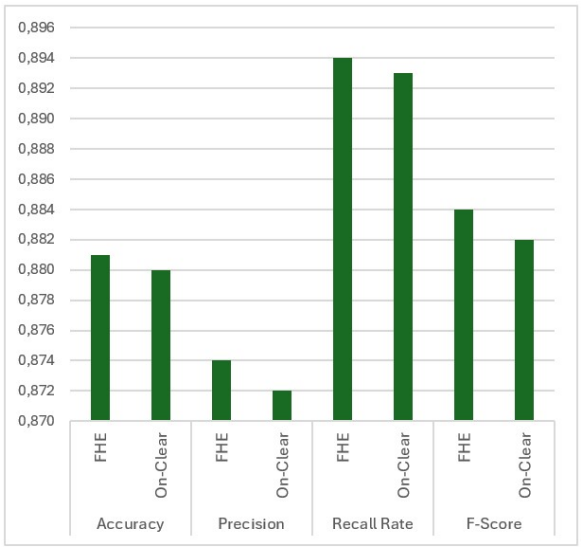


Figure 5.3: Decision Tree ML Performance Metrics

Table 5.1 and Table 5.2 show values up to 3 decimal places, but we verified that the results were identical even when using up to 15 decimal places. The exception is the Decision Tree model, which shows slightly different values, possibly caused by the FHE quantizations - highlighted in Figure 5.3. In the experiments, the FHE version of the Decision Tree utilizes 8 bits for quantization, a value that we considered optimal. Increasing it could improve the precision but reduce the runtime performance. Besides, Concrete uses TLUs for certain operations in tree-based models, which are much slower than basic integer operations utilized in linear algorithms. These minimal differences did not affect the prediction results, as shown in the confusion matrixes in Figure 5.4, where the on-clear scenario is in blue, and the FHE scenario is in red.

The XGBoost is also a tree-based model, but unlike the Decision Tree model, there were no differences in the results. What could justify this are the models' topologies and approaches to address quantization. The concrete version of Decision Tree is built based on scikit-learn[1], while the concrete version of XGBoost is built based on the XGBoost[2] library.

Regardless, XGBoost performed poorly, resulting in numerous false negatives, with 143 ham emails incorrectly classified as spam. It could have been caused by the type of dataset not being adequate for XGBoost or bad calibration and parametrization of the model. However, tree-assemble models, such as XGBoost, are less prone to overfitting, so the possibility of overfitting in the other models cannot be totally discarded.

The confusion matrixes show better results for LinearSVC and Logistic Regression, consistent with related works on spam detection reviewed in Chapter 3, both linear algorithms outperformed the tree-

[1] https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier
[2] https://xgboost.readthedocs.io/en/stable/python/python_api.html

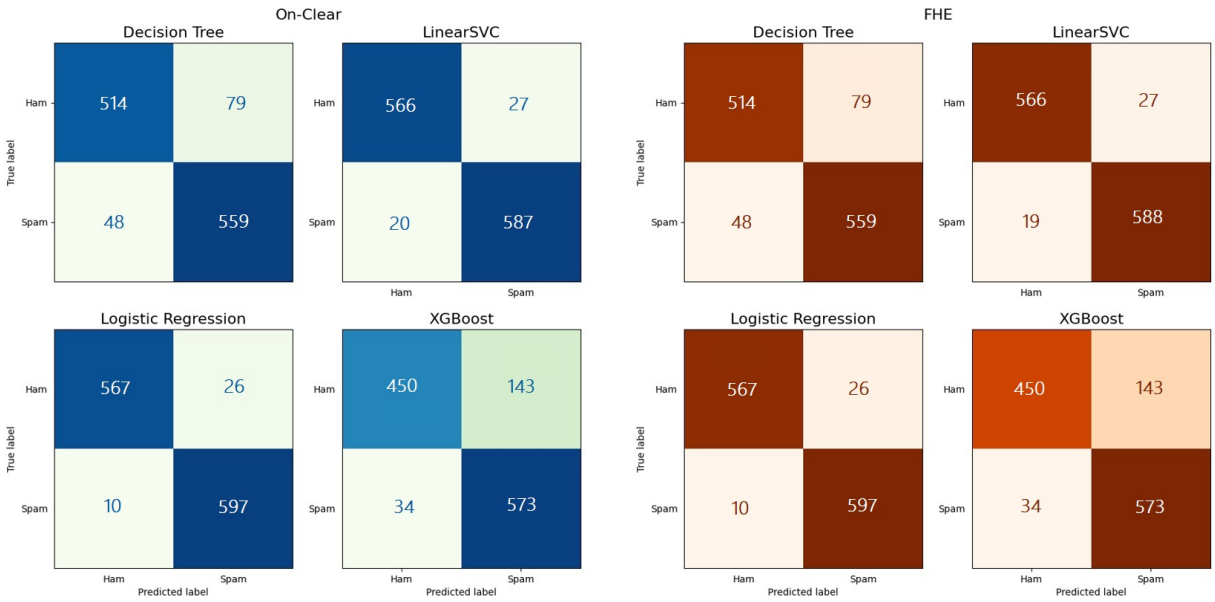


Figure 5.4: Confusion Matrix (On-Clear in Blue and FHE in Red)

based algorithms.

The calibration curves in Figure 5.5 gives a general idea of how reliable the models' predictions are. In both scenarios, FHE and on-clear, the graphics were identical. The calibration curves show that the models were similarly calibrated, except for the Decision Tree, which shows overconfident predictions and could lead to overfitting.

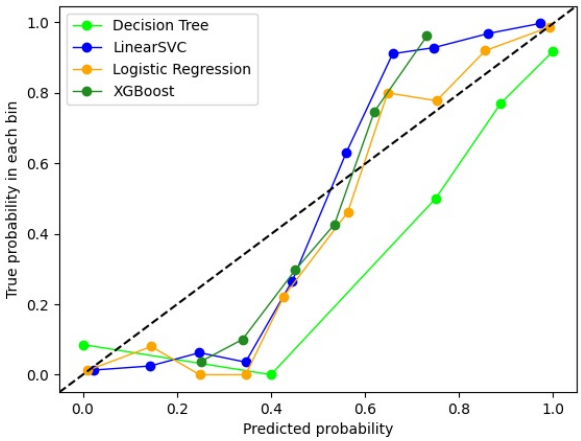


Figure 5.5: Models Calibration Curves

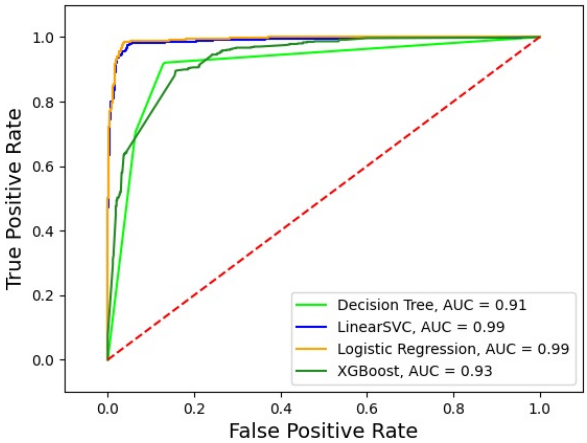


Figure 5.6: ROC Curves and AUC

Figure 5.6 shows the Roc curves for the tested models. The AUC values are the numeric representation of the area under each curve, ranging from 0 to 1. We see that Linear SVC and Logistic regression achieved the highest AUC of 0,99 - consistent with the previous performance metrics.

5.4 Runtime Performance

The feature array used as model input, corresponding to 20% of the dataset, represents 1200 email samples. Before encryption it had the size of 40 128 kilobytes, after encrypted using Fully Homomorphic Encryption it raised to 120 297 kilobytes. On average, the FHE data used as input for the models was three times larger than the clear data.

To evaluate the runtime performance, we considered the processing time in various stages and the peaks in memory usage during the FHE-SD execution. When the data is on-clear, there are only two stages, as detailed in Table 5.3. The Linear SVC algorithm presented the fastest processing time of 0,185 second, while the Decision Tree algorithm was the slowest, with a total time of 2,292 seconds.

Table 5.3: On-clear Execution Time in Seconds

	Linear SVC	Logistic Regression	Decision Tree	XGBoost
Model Training	0,172	2,275	3,550	3,227
Inference	0,013	0,017	0,015	0,032
Total Time	0,185	2,292	3,565	3,259

During memory usage peaks, the models executed on-clear presented the following results:

- **Linear SVC:** 40 kilobytes
- **Logistic Regression:** 1,5 megabyte
- **Decision Tree:** 85,98 megabytes
- **XGBoost:** 30 kilobytes

For FHE scenarios, there are several additional stages, which are described in Table 5.4. The execution times are measured in seconds, and longer stages are highlighted in orange and red, indicating several minutes or hours.

Table 5.4: FHE Execution Time in Seconds

	Linear SVC	Logistic Regression	Decision Tree	XGBoost
(1) **Model Training**	1,669	3,666	13,893	14,384
(2) **Model Compilation**	4,979	4,690	37,197	65,587
(3) **Create Deployment Files**	0,020	0,016	1,903	1,942
(4) **Load Client Files**	0,007	0,007	0,284	0,333
(5) **Generate Public-Key Pair**	0,006	0,001	1,693	1,541
(6) **Input Encryption**	664,990 (11,08 minutes)	671,060 (11,18 minutes)	1249,140 (20,81 minutes)	1324,138 (22,06 minutes)
(7) **Model Inference**	741,559 (12,35 minutes)	764,915 (12,74 minutes)	39709,688 (11,03 hours)	71988,285 (19,99 hours)
(8) **Result Decryption**	1,793	1,772	4,318	5,162
(9) **Total Testing Inference**	1408,473 (23,47 minutes)	1437,825 (23,96 minutes)	40963,300 (11,37 hours)	73317,746 (20,36 hours)
(10) **Total Runtime**	1413,488 (23,55 minutes)	1442,543 (24,04 minutes)	41004,389 (11,39 hours)	73387,160 (20,38 hours)

The results in Table 5.4 and Figure 5.7 can be summarized as follows:

- (1) The ML models trained using the Concrete-ML library achieved execution times ranging from 1,6 to 14,3 seconds.
- (2) The trained model is compiled to generate the FHE circuit, with times ranging from 4,9 to 65,5 seconds.
- (3) The average time to create deployment files is less than 2 seconds.
- (4) The client machine loads the *client.zip* file under 1 second.
- (5) The client machine generated the secret and evaluation keys in less than 2 seconds.
- (6) The client encrypts the input data using the secret key, taking 11 minutes for the linear models and around 20 minutes for the tree-based models.
- (7) The compiled model is loaded in the server and predicts on the encrypted inputs, taking around 12 minutes for the linear models, 11 hours for the Decision Tree, and 20 hours for the XGBoost.
- (8) The predicted results from stage (7) are decrypted by the client using the secret key, taking less than 2 seconds for the linear models and about 5 seconds for the tree-based models.
- The item (9) sums up the entire inference stages, which includes (6), (7), and (8).

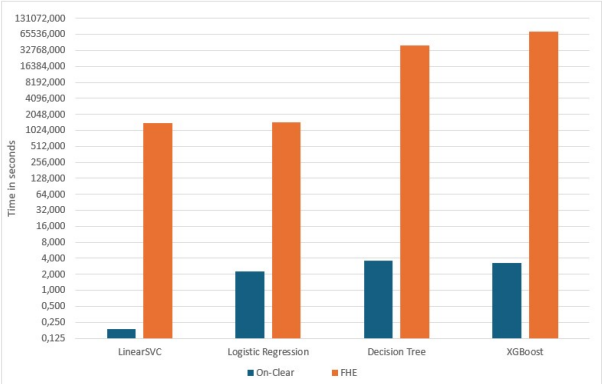


Figure 5.7: Execution Times

The total run time in (9) has the LinearSVC as the fastest, taking 23,55 minutes, and XGBoost as the slowest, taking 20,38 hours to finish. Regarding the memory usage peaks, the FHE models presented the following results:

- **Linear SVC:** 556,33 megabytes
- **Logistic Regression:** 556,34 megabytes
- **Decision Tree:** 193,1 megabytes
- **XGBoost:** 160,2 megabytes

Development and Overhead

Besides the constrained resources described in Section 4.3 and the limitations in Section 4.6, the developments required no direct intervention in the TFHE scheme. We put more effort into calibrating the machine learning models than actually applying the FHE techniques. The main challenge lies in finding the best ML model parameter values for their FHE versions. These values are necessary to balance the tradeoff between model performance and runtime performance. The model performance relates to the metrics in Section 5.3 and runtime performance to the execution times in Section 5.4. Another challenge was debugging the Concrete compiler and analyzing its source code to address the issues presented in Concrete-ML 1.4 and 1.5 - discussed in Section 4.6.

As FHE-SD was implemented and the experiments executed in a VM using Hyper-V, there may have been some overhead impacting the performance, but we tried to mitigate it as much as possible by avoiding communication between the guest VM and the host machine, storing the data directly in the VM disk area, and trying to keep in memory all data that would require more frequent accesses. The Hyper-V creates a virtual network adapter or switch, which enables communications between the VMs and the host operational system, resulting in increased latency.

Another potential overhead could arise from the approach of running predictions not in batch, but from each row of the feature array. It implies multiple calls to the model until the final result is obtained, with each encrypted input row being written to the simulated SM directory on disk, and then to the CM directory for encrypted prediction, which is then accessed and decrypted by CM. However, it is important to note that such overhead would not be present in real scenarios, where the feature array would represent a single email. In our tests, we deliberately used a feature array with 1200 rows, or 1200 email samples, to gather more data.

5.5 Preliminary Synthesis

This evaluation indicates that the development of FHE-based applications is within the reach of non-cryptographers, even using machines with limited processing power and memory. The results are optimistic, also because the experiments using FHE-SD involved all phases from training to predictions in a low-power machine. In real scenarios, it would be addressed by three distinct machines: a deployment machine that would train the model once and a server where the model is loaded to run the inferences.

The execution times presented were based on each model processing 1200 input emails without hardware acceleration, such as GPU parallelization. The tree-based models showed poor performances, which can be related to bad calibration or to the kind of dataset, further we did not implement custom TLUs nor programmable bootstrapping (PBS) on attempt to optimize the operations, which may not be possible when relying solely on Concrete API calls.

The linear models presented good results, indicating that they are the most fit for the purpose of spam detection using Fully Homomorphic Encryption.

Regarding security, we have concerns about the compiled models because their weights and parameters are not encrypted. Another concern is the introduction of the rounding feature for TLUs in CML 1.4, which significantly increased the performance of the tree-based models but may present vulnerabilities similar to those identified in CKKS [152]. The research community is still studying if TFHE is affected in the same way. In terms of privacy, the preprocessing and training were entirely executed on clear data, reducing the level of privacy. Concrete-ML has limited support for encrypted training, which was added in version 1.4 and only for SGD Classifiers - ML algorithm not addressed in this work.

5.6 Answered Research Questions

In this section, we answer the research questions raised in Chapter 1.

1. Is FHE feasible and possible to adopt in real-world applications such as spam detection?

A: Yes. However, as for machine learning or AI [58], FHE also requires significant energy resources, resulting in a high carbon footprint. With the growing impact of climate change and human predatory economic activities, much of what we have as certain today might change in the coming years. Another aspect is that Big Tech corporations such as Google, Yahoo, Microsoft, Meta, and Apple profit from exploiting users' private data and may create barriers against adopting FHE that would only be overcome through societal pressure and government regulation.

(a) Can FHE spam detection solutions achieve identical inference and ML performance results as their on-clear counterparts?

A: Yes, we achieved identical results as presented in tables Table 5.1 and Table 5.2, except for the decision tree model, which does not pose a significant concern, as discussed in the previous sections of this chapter.

(b) Can the development process of FHE-based applications be facilitated and made attainable to non-cryptographers?

A: Yes. In Section 2.6 and Section 2.7, we have a brief overview of the underlying complexity of Fully Homomorphic Encryption. However, FHE-SD did not require advanced knowledge for its implementation; this aspect is verifiable by analyzing the FHE-SD source code and through the concepts in Section 3.3 and Section 4.2, which summarize how Concrete abstracts the complexity of FHE.

(c) Do applications like spam detection require hardware acceleration or specialized hardware? To address this question, we establish that specialized hardware is unnecessary if a single spam email can be classified as spam or ham within 10 seconds.

A: Based on research works and empirically, it is acknowledged that sending and receiving emails takes less than a second to around 30 minutes. This duration can vary depending on the network and server load [198]. Thus, we first defined 10 seconds as a reasonable threshold time to classify a single email in real-world scenarios. This value is based on the statement from a commercial anti-spam solution [199] and a conversation with the *CERT.br*[3]. After establishing the threshold, we selected LinearSVC as the most suitable reference algorithm because it delivered the best results in our tests. Besides, it is the most adopted machine learning algorithm for spam detection, as indicated by the related works discussed in Section 3.5 and Section 3.6. The LinearSVC classified 1200 emails in 23,47 minutes as per Table 5.4, row nine (Total Testing Inference). Dividing that result by 1200, we have 1,17 second to classify a single email, a value below the threshold of 10 seconds. We successfully achieved good results with no use of specialized hardware. However, the impact of utilizing datasets containing other types of features, such as images, still requires evaluation.

[3]CERT.br is the Brazilian National Computer Emergency Response Team, maintained by NIC.br, which is the executive branch of the Brazilian Internet Steering Committee: https://www.cert.br

6

Conclusion

Contents

6.1 Conclusions

The prevalence and often compulsory use of cloud computing has resulted in many security incidents, such as data breaches in both public and private sectors, exposing millions of individuals' information. Besides the security concerns, service providers lack transparency, disregard regulations, and use private data for purposes such as targeted advertising and training large language models (LLM) based services, which has recently become a highly profitable sector. Additionally, many electronic mail (email) providers have been found to cooperate with intelligence agencies for illegal surveillance of individuals and governments. Fully Homomorphic Encryption is a known potential solution for these security and privacy concerns.

However, Big Tech corporations such as Google, Microsoft, Yahoo, Meta, and Apple might be a major obstacles for FHE since they built their business on exploiting private data. Therefore, FHE's progress somehow depends on how successful society is in advocating for its rights and governments in regulating and enforcing the law over these corporations. It also requires active participation of the academic sector by reaching the general lay public. No change comes without a collective effort.

In this thesis, we conducted a series of experiments to assess the feasibility and perspectives for wide adoption of Fully Homomorphic Encryption (FHE), having as an experimental subject the FHE-SD, a spam detection application crafted to achieve our research objectives. FHE-SD enabled us to evaluate the performance of machine learning-based spam detection algorithms in both FHE-encrypted data and on-clear data, providing us with reliable results under comparable setups. Moreover, it offered valuable insights into the practicality of leveraging FHE in real-world applications, even when dealing with limited hardware resources and constrained knowledge of FHE's mathematical basis, such as lattices. We demonstrated that the time and effort required to develop FHE applications is greatly reduced by using compilers and libraries to abstract the inherent complexity of FHE schemes such as TFHE.

Through FHE-SD, we evaluated the implementation of an FHE spam detection solution compatible with a client-server model from the early development stages up to deployment while partially simulating networking interactions using disk or memory space.

The data preprocessing approach was uniform for both FHE and non-encrypted scenarios, as the machine-learning models were initially trained on unencrypted datasets. The FHE-SD does not support encrypted training as it was developed before the release of CML 1.4, which introduced limited support for encrypted training exclusively for SGD Classifiers - algorithms not addressed in this study. Our tests covered linear models (LinearSVC and Logistic Regression) and tree-based models (Decision Tree and XGBoost), with the linear models consistently outperforming the tree-based models.

The linear models proved to be highly effective for spam detection, aligning with existing research in the field. We conducted a thorough comparison of various metrics and successfully demonstrated the potential of performing Fully Homomorphic Encryption (FHE) inference on encrypted data. The robust-

ness of these results was further reinforced by the fact that each model yielded consistent outcomes with both encrypted and unencrypted data. Although the tree-based models showed poor performances, this could also be due to bad calibration or the nature of the dataset.

Despite continuous improvements, runtime performance remains the primary limitation when adopting Fully Homomorphic Encryption. In our testing environment, we observed that FHE linear models were 1152 times slower than their on-clear counterparts and the tree-based models 16 879 times slower. When comparing the different FHE scenarios, we discovered that the linear models required approximately 23 minutes to complete the execution, while the Decision Tree model took over 11 hours and XGBoost over 20 hours.

The performance of FHE models is primarily determined by factors such as the bitwidth, quantization, and the number of TLU operations required. Linear models are much faster than tree-based models because they do not use TLUs. Tree-based models involve many TLU operations, which relates to the number of nodes and leaves in the trees. These can grow significantly based on the number of estimators and the maximum depth allowed.

XGBoost is a tree-ensemble algorithm known for its superior performance. In the on-clear scenario, it surpassed the performance of the decision tree model. Regardless, our analysis indicated that quantization has the most impact on the performance in the FHE scenario. Tests with reduced bitwidth exhibited a significant performance boost, although at the cost of accuracy. It is also worth noting that we did not implement custom TLU nor programmable bootstrapping (PBS) in an attempt to optimize the operations, which may not be possible when relying solely on Concrete API.

The variations in quantization methods significantly impact the comparability between fully homomorphic encryption (FHE) and on-clear scenarios, thereby influencing the reliability of our comparisons. For instance, this study did not include Neural Networks (NN) algorithms due to the substantial difference in the way NN is implemented in Concrete compared to pure scikit-learn implementation, not just in terms of quantization but also because CML uses the *skorch* library to integrate with scikit-learn dependencies.

The findings outlined in this study are promising despite the performance challenges. FHE-SD was deployed in a low-power machine, where the entire machine-learning workflow was executed, from training to inference. This would typically involve three distinct machines in real scenarios: a deployment machine for training the model, a dedicated server for loading the model and performing inferences, and a client machine for providing encrypted inputs to the server. So, the execution times are satisfactory, especially considering that we have combined two resource-intensive technologies: fully homomorphic encryption and machine learning. We obtained our metrics using a feature array with 1200 rows and 5000 features as input for the models. It means that we classified 1200 emails, which requires more computational resources while relying solely on the automatic parallelization provided by the Concrete compiler. Taking as reference the LinearSVC results, it classified a single email in around 1,17 second,

value bellow the defined threshold of 10 seconds. Therefore, we successfully obtained good results using general-purpose hardware and without relying on advanced knowledge of cryptographic and mathematic principles to implement a FHE solution. Such results and the ongoing investments in libraries and standardization presented in Section 3.3 and Section 3.4 reinforce FHE's potential for broad adoption.

6.2 Future Work

This research primarily utilized Fully Homomorphic Encryption (FHE) for private inference. The models made predictions on encrypted inputs, and the encrypted results were only decrypted by the client or user. Protocols should be established to protect the data used for training and ensure complete privacy. A possible approach is through federated learning, which maintains data privacy using a trusted gradient aggregator and differential privacy. Another potential research direction would be implementing the SGD Classifier model to test encrypted training and deploy it in a real network under a client-server architecture, as shown in Figure 4.1. We also did not evaluate email datasets containing images and other types of features, which may present higher computing costs and are not currently supported by FHE-SD.

Further evaluations could also be achieved by adding to FHE-SD support for hardware acceleration. For example, the current results could be compared in a similar setup using GPU parallelization. In any case, with the growing impact of climate change and human predatory economic activities, most research works should consider its impact on the environment. Hardware acceleration can be useful only in cases where the energy requirements are compatible with international goals for carbon emissions reduction [58, 200]. Otherwise, research efforts should be focused on continuously improving the performance of the algorithms and schemes.

Bibliography

[1] W. Goralski, *The illustrated network: how TCP/IP works in a modern network*, ser. The Morgan Kaufmann series in networking. Amsterdam; Boston: Elsevier/Morgan Kaufmann Publishers, 2009.

[2] J. F. Kurose and K. W. Ross, *Computer networking: a top-down approach*, eighth edition ed. Hoboken: Pearson, 2021.

[3] NIC.br, CGI.br, and CERT.br, "Antispam.br ::," https://www.antispam.br/admin/greylisting/, [Accessed 22-06-2024].

[4] S. Badillo, B. Banfai, F. Birzele, Iakov, L. Hutchinson, T. Thong, J. Polster, B. Steiert, and J. D. Zhang, "An introduction to machine learning," *Clinical Pharmacology & Therapeutics*, vol. 107, no. 4, p. 871–885, Apr. 2020. [Online]. Available: https://ascpt.onlinelibrary.wiley.com/doi/10.1002/cpt.1796

[5] "Issue information," *Expert Systems*, vol. 41, no. 2, p. e13344, 2024. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/exsy.13344

[6] M. Azhari, A. Alaoui, Z. Achraoui, B. Ettaki, and J. Zerouaoui, "Adaptation of the random forest method: solving the problem of pulsar search," in *Proceedings of the 4th International Conference on Smart City Applications*, ser. SCA '19. New York, NY, USA: Association for Computing Machinery, Oct. 2019, p. 1–6. [Online]. Available: https://doi.org/10.1145/3368756.3369004

[7] R. Agrawal and A. Joshi, *On architecting fully homomorphic encryption-based computing systems*. Springer, 2023.

[8] S. Abels and E. Khisamutdinov, "Nucleic acid computing and its potential to transform silicon-based technology," *DNA and RNA Nanotechnology*, vol. 2, 12 2015.

[9] D. Harris and S. Harris, *Digital Design and Computer Architecture, Second Edition*, 2nd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2012.

[10] B. Pulido-Gaytan, A. Tchernykh, J. M. Cortes-Mendoza, M. Babenko, G. Radchenko, A. Avetisyan, and A. Y. Drozdov, "Privacy-preserving neural networks with homomorphic encryption: C hallenges and opportunities," *Peer-to-Peer Networking and Applications*, vol. 14, no. 3, pp. 1666–1691, 2021.

[11] D. Stehlé, R. Steinfeld, K. Tanaka, and K. Xagawa, "Efficient public key encryption based on ideal lattices," in *Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ser. ASIACRYPT '09. Berlin, Heidelberg: Springer-Verlag, 2009, p. 617–635. [Online]. Available: https://doi.org/10.1007/978-3-642-10366-7_36

[12] M. Joye, "Homomorphic encryption 101," Dec. 2021. [Online]. Available: https://zamafhe.substack.com/p/homomorphic-encryption-101

[13] Zama, "Tfhe deep dive - part ii - encodings and linear leveled operations," 2022. [Online]. Available: https://www.zama.ai/post/tfhe-deep-dive-part-2

[14] C. Marcolla, V. Sucasas, M. Manzano, R. Bassoli, F. H. P. Fitzek, and N. Aaraj, "Survey on fully homomorphic encryption, theory, and applications," *Proceedings of the IEEE*, vol. 110, no. 10, pp. 1572–1609, 2022.

[15] A. Viand, P. Jattke, and A. Hithnawi, "Sok: Fully homomorphic encryption compilers," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 1092–1108.

[16] A. Safi and S. Singh, "A systematic literature review on phishing website detection techniques," *Journal of King Saud University-Computer and Information Sciences*, vol. 35, no. 2, pp. 590–611, 2023.

[17] Zama, "Concrete: TFHE Compiler that converts python programs into FHE equivalent," 2022, https://github.com/zama-ai/concrete.

[18] P. Resnick, *Internet Message Format*, Oct. 2008, no. RFC 5322. [Online]. Available: https://datatracker.ietf.org/doc/rfc5322/

[19] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer, *OpenPGP Message Format*, Nov. 2007, no. RFC4880. [Online]. Available: https://www.rfc-editor.org/info/rfc4880

[20] Google, "Google terms of service," [Accessed 22-06-2024].

[21] K. O'Flaherty, "How private is your gmail, and should you switch?" *The Observer*, May 2021. [Online]. Available: https://www.theguardian.com/technology/2021/may/09/how-private-is-your-gmail-and-should-you-switch

[22] Sep. 2024. [Online]. Available: https://legal.yahoo.com/us/en/yahoo/privacy/products/communications/index.html

[23] Sep. 2024. [Online]. Available: https://www.microsoft.com/en-us/servicesagreement

[24] T. Post, “The PRISM program,” https://www.washingtonpost.com/wp-srv/special/politics/prism-collection-documents/, [Accessed 09-08-2024].

[25] E. MacAskill, G. Dance, F. Cage, G. Chen, and N. Popovich, “Nsa files decoded: Edward snowden’s surveillance revelations explained,” Nov. 2013. [Online]. Available: http://www.theguardian.com/world/interactive/2013/nov/01/snowden-nsa-files-surveillance-revelations-decoded

[26] May 2019. [Online]. Available: https://theintercept.com/series/snowden-archive/

[27] G. Greenwald, E. MacAskill, L. Poitras, S. Ackerman, and D. Rushe, “Microsoft handed the nsa access to encrypted messages,” *The Guardian*, Jul. 2013. [Online]. Available: https://www.theguardian.com/world/2013/jul/11/microsoft-nsa-collaboration-user-data

[28] I. Borges Monroy, “Immobilized or petrified? explaining privacy concerns and the (de) mobilization against mass online surveillance in 21st-century advanced democracies,” 2023.

[29] M. Rojszczak, *Bulk surveillance, democracy and human rights law in Europe*. London: Routledge, Jun. 2024.

[30] M. Zajko, “Security against surveillance: It security as resistance to pervasive surveillance,” *Surveillance & Society*, vol. 16, no. 1, p. 39–52, Apr. 2018. [Online]. Available: http://dx.doi.org/10.24908/ss.v16i1.5316

[31] R. T. Braden, *Requirements for Internet Hosts - Communication Layers*, Oct. 1989, no. RFC 1122. [Online]. Available: https://datatracker.ietf.org/doc/rfc1122/

[32] S. Kumar, S. Dalal, and V. Dixit, “The osi model: overview on the seven layers of computer networks,” *International Journal of Computer Science and Information Technology Research*, vol. 2, no. 3, pp. 461–466, 2014.

[33] H. T. Alvestrand, *A Mission Statement for the IETF*, Oct. 2004, no. RFC 3935. [Online]. Available: https://datatracker.ietf.org/doc/rfc3935/

[34] C. Koymans and J. Scheerder, *Email*. Elsevier, 2008, p. 147–172. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/B9780444521989500094

[35] *UNIX and Linux system administration handbook*, Fifth edition ed. Boston, MA: Addison-Wesley, 2018.

[36] J. C. Klensin, *Simple Mail Transfer Protocol*, Oct. 2008, no. RFC 5321. [Online]. Available: https://datatracker.ietf.org/doc/rfc5321/

[37] D. Crocker, J. C. Klensin, E. A. Stefferud, M. T. Rose, and N. Freed, *SMTP Service Extensions*, Nov. 1995, no. RFC 1869. [Online]. Available: https://datatracker.ietf.org/doc/rfc1869/

[38] M. Crispin, *INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1*, Mar. 2003, no. RFC3501. [Online]. Available: https://www.rfc-editor.org/info/rfc3501

[39] M. T. Rose and J. G. Myers, *Post Office Protocol - Version 3*, May 1996, no. RFC 1939. [Online]. Available: https://datatracker.ietf.org/doc/rfc1939/

[40] W. Goralski, *The illustrated network: how TCP/IP works in a modern network*, second edition ed. Cambridge, MA: Morgan Kaufmann Publishers, 2017.

[41] N. Freed and N. Borenstein, *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*, Nov. 1996, no. RFC2045. [Online]. Available: https://www.rfc-editor.org/info/rfc2045

[42] F. Brunton, *Spam: a shadow history of the Internet*, ser. Infrastructures series. Cambridge (Mass.): The MIT press, 2013.

[43] L. F. Cranor and B. A. LaMacchia, "Spam!" *Communications of the ACM*, vol. 41, no. 8, pp. 74–83, 1998.

[44] R. G. Brody, S. Kern, and K. Ogunade, "An insider's look at the rise of nigerian 419 scams," *Journal of Financial Crime*, vol. 29, no. 1, pp. 202–214, 2022.

[45] "Monthly share of spam in the total e-mail traffic worldwide from january 2014 to december 2023." [Online]. Available: https://www.statista.com/statistics/420391/spam-email-traffic-share/

[46] A. M. Shibli, M. M. A. Pritom, and M. Gupta, "Abusegpt: Abuse of generative ai chatbots to create smishing campaigns," *arXiv preprint arXiv:2402.09728*, 2024.

[47] P. H. C. Guerra, D. Guedes, W. Meira, C. Hoepers, M. Chaves, and K. Jessen, "Spamming chains: a new way of understanding spammer behavior," in *The 6th g Conference on Email and Anti-Spam*, 2009.

[48] E. P. Sanz, J. M. Gómez Hidalgo, and J. C. Cortizo Pérez, "Chapter 3 email spam filtering," in *Software Development*, ser. Advances in Computers. Elsevier, 2008, vol. 74, pp. 45–114. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0065245808006037

[49] D. F. Cook, J. Hartnett, K. Manderson, and J. Scanlan, "Catching spam before it arrives: domain specific dynamic blacklists," 2006.

[50] H. Evan, "The next step in the spam control war: Greylisting," *http://projects. puremagic. com/greylisting/whitepaper. html*, 2003.

[51] W. Z. Khan, M. K. Khan, F. T. B. Muhaya, M. Y. Aalsalem, and H.-C. Chao, "A comprehensive study of email spam botnet detection," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2271–2295, 2015.

[52] E. Harris, "Greylisting," *http://projects. puremagic. com/greylisting/*, 2013.

[53] *Sieve: An Email Filtering Language*, Jan. 2008, no. RFC5228. [Online]. Available: https://www.rfc-editor.org/info/rfc5228

[54] S. J. Russell, P. Norvig, M.-w. Chang, J. Devlin, A. Dragan, D. Forsyth, I. Goodfellow, J. Malik, V. Mansinghka, J. Pearl, and M. J. Wooldridge, *Artificial intelligence: a modern approach*, fourth edition, global edition ed., ser. Pearson series in artificial intelligence. Harlow: Pearson, 2022.

[55] A. M. Turing, *Computing Machinery and Intelligence*. Dordrecht: Springer Netherlands, 2009, pp. 23–65. [Online]. Available: https://doi.org/10.1007/978-1-4020-6710-5_3

[56] T. Luong, L. Dinh, H. Nguyen, and L. Tran, "Novel hardware implementation of deduplicating visually identical jpeg image chunks," *IEEE Access*, vol. 12, pp. 69 568–69 577, 2024.

[57] A. Anžel, D. Heider, and G. Hattab, "The visual story of data storage: From storage properties to user interfaces," *Computational and Structural Biotechnology Journal*, vol. 19, pp. 4904–4918, 2021.

[58] A. de Vries, "The growing energy footprint of artificial intelligence," *Joule*, vol. 7, no. 10, pp. 2191–2194, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2542435123003653

[59] S. M. Weiss and C. A. Kulikowski, *Computer systems that learn: classification and prediction methods from statistics, neural nets, machine learning, and expert systems*. San Mateo, Calif: M. Kaufmann Publishers, 1991.

[60] H. Inam, N. U. Islam, M. U. Akram, and F. Ullah, "Smart and automated infrastructure management: A deep learning approach for crack detection in bridge images," *Sustainability*, vol. 15, no. 3, p. 1866, 2023.

[61] O. A. M. LOPEZ, *Multivariate Statistical Machine Learning Methods for Genomic Prediction*. SPRINGER, 2022.

[62] T. Hu and X.-H. Zhou, "Unveiling llm evaluation focused on metrics: Challenges and solutions," *arXiv preprint arXiv:2404.09135*, 2024.

[63] A. Yedidia, "Against the f-score," *URL: https://adamyedidia. files. wordpress. com/2014/11/fscore. pdf*, 2016.

[64] W. J. Krzanowski and D. J. Hand, *ROC curves for continuous data*, ser. Monographs on statistics and applied probability. Boca Raton: CRC Press, 2009.

[65] Z. H. Hoo, J. Candlish, and D. Teare, “What is an roc curve?” *Emergency Medicine Journal*, vol. 34, no. 6, pp. 357–359, 2017. [Online]. Available: https://emj.bmj.com/content/34/6/357

[66] X. Zhang, X. Li, Y. Feng, and Z. Liu, “The use of roc and auc in the validation of objective image fusion evaluation metrics,” *Signal Processing*, vol. 115, pp. 38–48, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0165168415001085

[67] scikit-learn, “Probability calibration,” 2024. [Online]. Available: https://scikit-learn.org/stable/modules/calibration.html

[68] *Encyclopedia of Machine Learning and Data Science*. New York, NY: Springer US, 2020. [Online]. Available: https://link.springer.com/10.1007/978-1-4899-7502-7

[69] C. Albon, *Machine learning with Python cookbook: practical solutions from preprocessing to deep learning*, first edition ed. Sebastopol, CA: O’Reilly Media, 2018.

[70] Y. Goldberg, *Neural network methods for natural language processing*, reprint of original edition ©morgan&claypool 2017 ed., ser. Synthesis lectures on human language technologies. Cham: Springer nature Switzerland AG, 2022.

[71] C. Manning, P. Raghavan, and H. Schutze, *Introduction to information retrieval*. New York: Cambridge University Press, 2008.

[72] *Encyclopedia of algorithms*, ser. Springer reference. New York; London: Springer, 2007.

[73] scikit-learn, “LinearSVC,” 2024. [Online]. Available: https://scikit-learn/stable/modules/generated/sklearn.svm.LinearSVC.html

[74] scikit learn, “Linear models,” 2024. [Online]. Available: https://scikit-learn/stable/modules/linear_model.html

[75] A. B. Musa, “Comparative study on classification performance between support vector machine and logistic regression,” *International Journal of Machine Learning and Cybernetics*, vol. 4, no. 1, p. 13–24, Feb. 2013. [Online]. Available: https://doi.org/10.1007/s13042-012-0068-x

[76] R. Panigrahi and S. Borah, *Classification and Analysis of Facebook Metrics Dataset Using Supervised Classifiers*. Elsevier, 2019, p. 1–19. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/B9780128154588000013

[77] T. Chen and C. Guestrin, “Xgboost.” [Online]. Available: https://xgboost.ai/

[78] Chen, Tianqi and Guestring, Carlos, “Xgboost: A scalable tree boosting system,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD ’16.

New York, NY, USA: Association for Computing Machinery, 2016, p. 785–794. [Online]. Available: https://doi.org/10.1145/2939672.2939785

[79] E. Conrad, S. Misenar, and J. Feldman, *Eleventh Hour CISSP*. Elsevier Science 2016., 2016.

[80] J. Soni and R. Goodman, *A mind at play: how Claude Shannon invented the information age*. New York: Simon & Schuster, 2017.

[81] R. A. Mollin, *An introduction to cryptography*, 2nd ed., ser. Discrete mathematics and its applications. Boca Raton: Chapman & Hall/CRC, 2007.

[82] J. Katz and Y. Lindell, *Introduction to modern cryptography*, second edition ed., ser. Chapman & hall/crc cryptography and network security series. Boca Raton: CRC Press/Taylor & Francis, 2015.

[83] L. Velvindron, K. Moriarty, and A. Ghedini, *Deprecating MD5 and SHA-1 Signature Hashes in TLS 1.2 and DTLS 1.2*, Dec. 2021, no. RFC 9155. [Online]. Available: https://datatracker.ietf.org/doc/rfc9155/

[84] T. Knoll, "Adapting kerckhoffs's principle," *Advanced Microkernel Operating Systems (2018)*, vol. 93, 2018.

[85] R. Ganjewar, "Diffie hellman key exchange," *Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA*, vol. 93106, 2010.

[86] D. R. Stinson and M. B. Paterson, *Cryptography: theory and practice*, fourth edition. ed., ser. Textbooks in mathematics. Boca Raton, FL: Chapman & Hall/CRC Press, 2022.

[87] S. L. Nita and M. I. Mihailescu, *Advances to Homomorphic and Searchable Encryption*. Cham: Springer Nature Switzerland, 2023. [Online]. Available: https://link.springer.com/10.1007/978-3-031-43214-9

[88] R. L. Rivest, L. Adleman, M. L. Dertouzos *et al.*, "On data banks and privacy homomorphisms," *Foundations of secure computation*, vol. 4, no. 11, pp. 169–180, 1978.

[89] R. L. Rivest, A. Shamir, and L. M. Adleman, "Cryptographic communications system and method," Sep. 1983. [Online]. Available: https://patents.google.com/patent/US4405829A/en

[90] C. Paar and J. Pelzl, *Understanding cryptography*, 2010th ed. Berlin, Germany: Springer, Nov. 2014.

[91] V. F. da Rocha and J. López, "An overview on homomorphic encryption algorithms," 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:202667135

[92] J. L. Hennessy, *Computer architecture: a quantitative approach*, sixth edition ed. Cambridge, MA: Morgan Kaufmann Publishers, 2019.

[93] J. Daghlian, *Logica e algebra de Boole*, 4th ed. Sao Paulo (SP): Atlas, 2009.

[94] M. Babu and S. K. GA, “In-depth survey on xor gate design,” *IN-DEPTH*, vol. 7, no. 18, p. 2020, 2020.

[95] K. Munjal and R. Bhatia, “Analysing rsa and paillier homomorphic property for security in cloud,” *Procedia Computer Science*, vol. 215, pp. 240–246, 2022.

[96] P. Paillier, “Public-key cryptosystems based on composite degree residuosity classes,” in *International conference on the theory and applications of cryptographic techniques*. Springer, 1999, pp. 223–238.

[97] *Protecting privacy through homomorphic encryption*, corrected publication ed. Cham, Switzerland: Springer Nature, 2022.

[98] K. Munjal and R. Bhatia, “A systematic review of homomorphic encryption and its contributions in healthcare industry,” *Complex & Intelligent Systems*, vol. 9, no. 4, pp. 3759–3786, 2023.

[99] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, “A survey on homomorphic encryption schemes: Theory and implementation,” *ACM Computing Surveys (Csur)*, vol. 51, no. 4, pp. 1–35, 2018.

[100] A. Silverberg, “Fully homomorphic encryption for mathematicians,” *Women in numbers 2: research directions in number theory*, vol. 606, p. 111, 2013.

[101] J. Zhang and Z. Zhang, *Lattice-Based Cryptosystems*. Springer, 2020.

[102] M. Sipser, *Introduction to the theory of computation*, 3rd ed. Belmont, CA: Wadsworth Publishing, Jun. 2012.

[103] C. Gentry, A. Sahai, and B. Waters, “Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based,” in *Advances in Cryptology – CRYPTO 2013*, R. Canetti and J. A. Garay, Eds. Berlin, Heidelberg: Springer, 2013, p. 75–92.

[104] I. Chillotti, D. Ligier, J.-B. Orfila, and S. Tap, “Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for tfhe,” in *Advances in Cryptology–ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6–10, 2021, Proceedings, Part III 27*. Springer, 2021, pp. 670–699.

[105] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, “(leveled) fully homomorphic encryption without bootstrapping,” *ACM Transactions on Computation Theory (TOCT)*, vol. 6, no. 3, pp. 1–36, 2014.

[106] Zama, “TFHE-rs: A Pure Rust Implementation of the TFHE Scheme for Boolean and Integer Arithmetics Over Encrypted Data,” 2022, https://github.com/zama-ai/tfhe-rs.

[107] Zama, “Concrete-ml: a privacy-preserving machine learning library using fully homomorphic encryption for data scientists,” 2022, https://github.com/zama-ai/concrete-ml.

[108] L. Ducas and D. Micciancio, “FHEW: Bootstrapping homomorphic encryption in less than a second,” Cryptology ePrint Archive, Paper 2014/816, 2014, https://eprint.iacr.org/2014/816. [Online]. Available: https://eprint.iacr.org/2014/816

[109] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, “Tfhe: fast fully homomorphic encryption over the torus,” *Journal of Cryptology*, vol. 33, no. 1, pp. 34–91, 2020.

[110] Chillotti, Ilaria and Gama, Nicolas and Georgieva, Mariya and Izabach, Malika, “Faster packed homomorphic operations and efficient circuit bootstrapping for tfhe,” in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2017, pp. 377–408.

[111] I. Chillotti, M. Joye, and P. Paillier, “Programmable bootstrapping enables efficient homomorphic inference of deep neural networks,” in *Cyber Security Cryptography and Machine Learning: 5th International Symposium, CSCML 2021, Be’er Sheva, Israel, July 8–9, 2021, Proceedings 5*. Springer, 2021, pp. 1–19.

[112] “ISO/IEC WD 18033-8,” https://www.iso.org/standard/83139.html, [Accessed 23-06-2024].

[113] C. Song and R. Huang, “Secure convolution neural network inference based on homomorphic encryption,” *Applied Sciences*, vol. 13, no. 10, p. 6117, 2023.

[114] H. Chen, I. Chillotti, and Y. Song, “Improved bootstrapping for approximate homomorphic encryption,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2019, pp. 34–54.

[115] L. Rovida, “Fast but approximate homomorphic k-means based on masking technique,” *International Journal of Information Security*, vol. 22, no. 6, pp. 1605–1619, 2023.

[116] J. Wang, Z. Xia, Y. Chen, C. Hu, and F. Yu, “Intrusion detection framework based on homomorphic encryption in ami network,” *Frontiers in Physics*, vol. 10, p. 1102892, 2022.

[117] H. Fang and Q. Qian, “Privacy preserving machine learning with homomorphic encryption and federated learning,” *Future Internet*, vol. 13, no. 4, p. 94, 2021.

[118] World Health Organization, “Human rights,” https://www.who.int/news-room/fact-sheets/detail/human-rights-and-health, [Accessed 22-06-2024].

[119] T. B. Patil, G. K. Patnaik, and A. T. Bhole, “Big data privacy using fully homomorphic non-deterministic encryption,” in *2017 IEEE 7th International Advance Computing Conference (IACC)*, 2017, pp. 138–143.

[120] E. Daniel, “Optimum wavelet-based homomorphic medical image fusion using hybrid genetic–grey wolf optimization algorithm,” *IEEE Sensors Journal*, vol. 18, no. 16, pp. 6804–6811, 2018.

[121] M. Kim, Y. Song, and J. H. Cheon, “Secure searching of biomarkers through hybrid homomorphic encryption scheme,” *BMC medical genomics*, vol. 10, pp. 69–76, 2017.

[122] X. Yi, A. Bouguettaya, D. Georgakopoulos, A. Song, and J. Willemson, “Privacy protection for wireless medical sensor data,” *IEEE Transactions on Dependable and Secure Computing*, vol. 13, no. 3, pp. 369–380, 2016.

[123] V. Rajalakshmi, S. A. STINA *et al.*, “Private searching on streaming data based on homomorphic encryption.” *International Journal on Information Sciences & Computing*, vol. 10, no. 2, 2016.

[124] A. Papadimitriou, R. Bhagwan, N. Chandran, R. Ramjee, A. Haeberlen, H. Singh, A. Modi, and S. Badrinarayanan, “Big data analytics over encrypted datasets with seabed,” in *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, 2016, pp. 587–602.

[125] J. P. Gibson, R. Krimmer, V. Teague, and J. Pomares, “A review of e-voting: the past, present and future,” *Annals of Telecommunications*, vol. 71, pp. 279–286, 2016.

[126] V. Cortier, “Formal verification of e-voting: solutions and challenges,” *ACM SIGLOG News*, vol. 2, no. 1, p. 25–34, jan 2015. [Online]. Available: https://doi.org/10.1145/2728816.2728823

[127] N. G. Tsoutsos and M. Maniatakos, “Efficient detection for malicious and random errors in additive encrypted computation,” *IEEE Transactions on Computers*, vol. 67, no. 1, p. 16–31, Jan. 2018. [Online]. Available: http://ieeexplore.ieee.org/document/7967774/

[128] Y. Lin, G. Liu, J. Wen, K. Hua, F. Jin, Y. Hu, X. Liu, and Q. Zhang, “Power data blockchain sharing scheme based on homomorphic encryption,” in *2022 IEEE 5th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, vol. 5. IEEE, 2022, pp. 625–629.

[129] S. A.-B. Salman, S. Al-Janabi, and A. M. Sagheer, “Valid blockchain-based e-voting using elliptic curve and homomorphic encryption.” *International Journal of Interactive Mobile Technologies*, vol. 16, no. 20, 2022.

[130] W. Qu, L. Wu, W. Wang, Z. Liu, and H. Wang, “A electronic voting protocol based on blockchain and homomorphic signcryption,” *Concurrency and Computation: Practice and Experience*, vol. 34, no. 16, p. e5817, 2022.

[131] L. Zhou, L. Wang, Y. Sun, and P. Lv, “Beekeeper: A blockchain-based iot system with secure storage and homomorphic computation,” *IEEE Access*, vol. 6, pp. 43 472–43 488, 2018.

[132] R. Shrestha and S. Kim, “Integration of iot with blockchain and homomorphic encryption: Challenging issues and opportunities,” in *Advances in computers*. Elsevier, 2019, vol. 115, pp. 293–331.

[133] G. Đorđević, M. Marković, and P. Vuletić, “Evaulation of homomorphic encryption implementation on iot device,” *JITA-APEIRON*, vol. 23, no. 1, pp. 32–39, 2022.

[134] S. Gupta, R. Garg, N. Gupta, W. S. Alnumay, U. Ghosh, and P. K. Sharma, “Energy-efficient dynamic homomorphic security scheme for fog computing in iot networks,” *Journal of Information Security and Applications*, vol. 58, p. 102768, 2021.

[135] N. Chakraborty and G. Patra, “Functional encryption for secured big data analytics,” *International Journal of Computer Applications*, vol. 107, no. 16, 2014.

[136] J. J. Menandas and J. J. Joshi, “Secure big data processing through homomorphic encryption in cloud computing environments,” 2016.

[137] T. Okamoto and S. Uchiyama, “A new public-key cryptosystem as secure as factoring,” in *Advances in Cryptology - EUROCRYPT ’98, International Conference on the Theory and Application of Cryptographic Techniques, Espoo, Finland, May 31 - June 4, 1998, Proceeding*, ser. Lecture Notes in Computer Science, vol. 1403. Springer, 1998, pp. 308–318.

[138] M. Creeger, “The rise of fully homomorphic encryption: Often called the holy grail of cryptography, commercial fhe is near.” *Queue*, vol. 20, no. 4, p. 39–60, sep 2022. [Online]. Available: https://doi.org/10.1145/3561800

[139] C. Gentry, “A fully homomorphic encryption scheme,” Ph.D. dissertation, Stanford, CA, USA, 2009, aAI3382729.

[140] C. Gentry and S. Halevi, “Implementing gentry’s fully-homomorphic encryption scheme,” Cryptology ePrint Archive, Paper 2010/520, 2010, https://eprint.iacr.org/2010/520. [Online]. Available: https://eprint.iacr.org/2010/520

[141] M. Van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, “Fully homomorphic encryption over the integers,” in *Advances in Cryptology–EUROCRYPT 2010: 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30–June 3, 2010. Proceedings 29*. Springer, 2010, pp. 24–43.

[142] Z. Brakerski and V. Vaikuntanathan, “Efficient fully homomorphic encryption from (standard) lwe,” *SIAM Journal on computing*, vol. 43, no. 2, pp. 831–871, 2014.

[143] J. Fan and F. Vercauteren, “Somewhat practical fully homomorphic encryption,” *Cryptology ePrint Archive*, 2012.

[144] Z. Brakerski, “Fully homomorphic encryption without modulus switching from classical gapsvp,” in *Annual cryptology conference*. Springer, 2012, pp. 868–886.

[145] A. López-Alt, E. Tromer, and V. Vaikuntanathan, “On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption,” in *Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing*, ser. STOC ’12. New York, NY, USA: Association for Computing Machinery, 2012, p. 1219–1234. [Online]. Available: https://doi.org/10.1145/2213977.2214086

[146] J. W. Bos, K. Lauter, J. Loftus, and M. Naehrig, “Improved security for a ring-based fully homomorphic encryption scheme,” in *Cryptography and Coding: 14th IMA International Conference, IMACC 2013, Oxford, UK, December 17-19, 2013. Proceedings 14*. Springer, 2013, pp. 45–64.

[147] M. Albrecht, S. Bai, and L. Ducas, “A subfield lattice attack on overstretched NTRU assumptions: Cryptanalysis of some FHE and graded encoding schemes,” Cryptology ePrint Archive, Paper 2016/127, 2016, https://eprint.iacr.org/2016/127. [Online]. Available: https://eprint.iacr.org/2016/127

[148] S. Mittal and K. Ramkumar, “A retrospective study on ntru cryptosystem,” in *AIP Conference Proceedings*, vol. 2451, no. 1. AIP Publishing, 2022.

[149] J. Alperin-Sheriff and C. Peikert, “Practical bootstrapping in quasilinear time,” Cryptology ePrint Archive, Paper 2013/372, 2013. [Online]. Available: https://eprint.iacr.org/2013/372

[150] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, “Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds,” Cryptology ePrint Archive, Paper 2016/870, 2016, https://eprint.iacr.org/2016/870. [Online]. Available: https://eprint.iacr.org/2016/870

[151] J. H. Cheon, A. Kim, M. Kim, and Y. Song, “Homomorphic encryption for arithmetic of approximate numbers,” in *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*. Springer, 2017, pp. 409–437.

[152] B. Li and D. Micciancio, “On the security of homomorphic encryption on approximate numbers,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2021, pp. 648–677.

[153] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, “Bootstrapping for approximate homomorphic encryption,” in *Advances in Cryptology–EUROCRYPT 2018: 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29-May 3, 2018 Proceedings, Part I 37*. Springer, 2018, pp. 360–384.

[154] M.-Y. Kao, *Encyclopedia of algorithms*. Springer Science & Business Media, 2008.

[155] V. Shoup, “NTL: A Library for doing Number Theory,” https://libntl.org/, [Accessed 22-06-2024].

[156] S. Halevi and V. Shoup, "Algorithms in helib," in *Advances in Cryptology–CRYPTO 2014: 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I 34*. Springer, 2014, pp. 554–571.

[157] "Microsoft SEAL (release 4.1)," https://github.com/Microsoft/SEAL, Jan. 2023, microsoft Research, Redmond, WA.

[158] "Heaan," Online: https://github.com/snucrypto/HEAAN, Sep. 2018.

[159] "Rns-heaan," Online: https://github.com/KyoohyungHan/FullRNS-HEAAN, Sep. 2018.

[160] "FV-NFllib," Online: https://github.com/CryptoExperts/FV-NFLlib, 2016.

[161] "Nfllib," Online: https://github.com/quarkslab/NFLlib, 2016.

[162] C. V. Mouchet, J.-P. Bossuat, J. R. Troncoso-Pastoriza, and J.-P. Hubaux, "Lattigo: A multiparty homomorphic encryption library in go," in *Proceedings of the 8th Workshop on Encrypted Computing and Applied Homomorphic Cryptography*, 2020, pp. 64–70.

[163] "Lattigo v5," Online: https://github.com/tuneinsight/lattigo, Nov. 2023, ePFL-LDS, Tune Insight SA.

[164] "cuFHE," Online: https://github.com/vernamlab/cuFHE, 2018.

[165] "nuFHE," Online: https://github.com/nucypher/nufhe, 2019.

[166] W. Dai and B. Sunar, "cuhe: A homomorphic encryption accelerator library," in *Cryptography and Information Security in the Balkans: Second International Conference, BalkanCryptSec 2015, Koper, Slovenia, September 3-4, 2015, Revised Selected Papers 2*. Springer, 2016, pp. 169–186.

[167] "Tfhe," Online: https://github.com/tfhe/tfhe, Feb. 2021.

[168] "Fhew," Online: https://github.com/lducas/FHEW, May 2017.

[169] Zama, "Zama concrete: Fully homomorphic encryption compiler," 2022, https://github.com/zama-ai/concrete.

[170] I. Chillotti, M. Joye, D. Ligier, J.-B. Orfila, and S. Tap, "Concrete: Concrete operates on ciphertexts rapidly by extending tfhe," in *WAHC 2020-8th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, 2020.

[171] "PALISADE Lattice Cryptography Library (release 1.11.5)," https://palisade-crypto.org/, 2021.

[172] A. Al Badawi, J. Bates, F. Bergamaschi, D. B. Cousins, S. Erabelli, N. Genise, S. Halevi, H. Hunt, A. Kim, Y. Lee *et al.*, "Openfhe: Open-source fully homomorphic encryption library," in *Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, 2022, pp. 53–63.

[173] “Helib,” https://github.com/homenc/HElib, [Accessed 22-06-2024].

[174] D. W. Archer, J. M. Calderón Trilla, J. Dagit, A. Malozemoff, Y. Polyakov, K. Rohloff, and G. Ryan, “Ramparts: A programmer-friendly system for building homomorphic encryption applications,” in *Proceedings of the 7th acm workshop on encrypted computing & applied homomorphic cryptography*, 2019, pp. 57–68.

[175] S. Gorantala, R. Springer, S. Purser-Haskell, W. Lam, R. Wilson, A. Ali, E. P. Astor, I. Zukerman, S. Ruth, C. Dibak, P. Schoppmann, S. Kulankhina, A. Forget, D. Marn, C. Tew, R. Misoczki, B. Guillen, X. Ye, D. Kraft, D. Desfontaines, A. Krishnamurthy, M. Guevara, I. M. Perera, Y. Sushko, and B. Gipson, “A general purpose transpiler for fully homomorphic encryption,” Cryptology ePrint Archive, Paper 2021/811, 2021, https://eprint.iacr.org/2021/811. [Online]. Available: https://eprint.iacr.org/2021/811

[176] E. Chielle, O. Mazonka, H. Gamil, N. G. Tsoutsos, and M. Maniatakos, “E3: A framework for compiling c++ programs with encrypted operands,” Cryptology ePrint Archive, Paper 2018/1013, 2018, https://eprint.iacr.org/2018/1013. [Online]. Available: https://eprint.iacr.org/2018/1013

[177] L. Coppolino, S. D’Antonio, V. Formicola, G. Mazzeo, and L. Romano, “Vise: Combining intel sgx and homomorphic encryption for cloud industrial control systems,” *IEEE Transactions on Computers*, vol. 70, no. 5, pp. 711–724, 2020.

[178] “NIST Announces First Four Quantum-Resistant Cryptographic Algorithms,” https://www.nist.gov/news-events/news/2022/07/nist-announces-first-four-quantum-resistant-cryptographic-algorithms, 2022, [Accessed 23-06-2024].

[179] D.-T. Dam, T.-H. Tran, V.-P. Hoang, C.-K. Pham, and T.-T. Hoang, “A survey of post-quantum cryptography: Start of a new race,” *Cryptography*, vol. 7, no. 3, p. 40, 2023.

[180] L. Das, L. Ahuja, and A. Pandey, “Existing spam filtering methods considering different technique: A review,” in *2021 International Conference on Technological Advancements and Innovations (ICTAI)*. IEEE, 2021, pp. 515–520.

[181] F. Jáñez-Martino, R. Alaiz-Rodríguez, V. González-Castro, E. Fidalgo, and E. Alegre, “A review of spam email detection: analysis of spammer strategies and the dataset shift problem,” *Artificial Intelligence Review*, vol. 56, no. 2, pp. 1145–1173, 2023.

[182] A. Karim, S. Azam, B. Shanmugam, K. Kannoorpatti, and M. Alazab, “A comprehensive survey for intelligent spam email detection,” *Ieee Access*, vol. 7, pp. 168 261–168 295, 2019.

[183] J.-J. Sheu, K.-T. Chu, N.-F. Li, and C.-C. Lee, “An efficient incremental learning mechanism for tracking concept drift in spam filtering,” *PloS one*, vol. 12, no. 2, p. e0171518, 2017.

[184] S. Asiri, Y. Xiao, S. Alzahrani, S. Li, and T. Li, “A survey of intelligent detection designs of html url phishing attacks,” *IEEE Access*, vol. 11, pp. 6421–6443, 2023.

[185] S. Jamal, H. Wimmer, and I. H. Sarker, “An improved transformer-based model for detecting phishing, spam and ham emails: A large language model approach,” *Security and Privacy*, p. e402, 2024.

[186] M. A. Pathak, M. Sharifi, and B. Raj, “Privacy preserving spam filtering,” *arXiv preprint arXiv:1102.4021*, 2011.

[187] A. C. Yao, “Protocols for secure computations,” in *23rd annual symposium on foundations of computer science (sfcs 1982)*. IEEE, 1982, pp. 160–164.

[188] A. Khedr, G. Gulak, and V. Vaikuntanathan, “Shield: scalable homomorphic implementation of encrypted data-classifiers,” *IEEE Transactions on Computers*, vol. 65, no. 9, pp. 2848–2858, 2015.

[189] S. Jaiswal, S. C. Patel, and R. S. Singh, “Privacy preserving spam email filtering based on somewhat homomorphic using functional encryption,” in *Proceedings of the 4th International Conference on Frontiers in Intelligent Computing: Theory and Applications (FICTA) 2015*. Springer, 2016, pp. 579–585.

[190] I. Demertzis, D. Froelicher, N. Luo, and M. N. Hovd, “i-seal 2: Identifying spam email with seal,” *Protecting Privacy through Homomorphic Encryption*, pp. 129–132, 2021.

[191] T. Nguyen, N. Karunanayake, S. Wang, S. Seneviratne, and P. Hu, “Privacy-preserving spam filtering using homomorphic and functional encryption,” *Computer Communications*, vol. 197, pp. 230–241, 2023.

[192] N. Samardzic, A. Feldmann, A. Krastev, N. Manohar, N. Genise, S. Devadas, K. Eldefrawy, C. Peikert, and D. Sanchez, “Craterlake: a hardware accelerator for efficient unbounded computation on encrypted data,” in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ser. ISCA ’22. New York, NY, USA: Association for Computing Machinery, 2022, p. 173–187. [Online]. Available: https://doi.org/10.1145/3470496.3527393

[193] N. Samardzic, A. Feldmann, A. Krastev, S. Devadas, R. Dreslinski, C. Peikert, and D. Sanchez, “F1: A fast and programmable accelerator for fully homomorphic encryption,” in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO ’21. New York, NY, USA: Association for Computing Machinery, 2021, p. 238–252. [Online]. Available: https://doi.org/10.1145/3466752.3480070

[194] R. Shea, J. Liu, E. C.-H. Ngai, and Y. Cui, “Cloud gaming: architecture and performance,” *IEEE Network*, vol. 27, no. 4, pp. 16–21, 2013.

[195] S. Gallagher, B. Gelman, S. Taoufiq, T. Vörös, Y. Lee, A. Kyadige, and S. Bergeron, "Phishing and social engineering in the age of llms," in *Large Language Models in Cybersecurity: Threats, Exposure and Mitigation*. Springer Nature Switzerland Cham, 2024, pp. 81–86.

[196] Ł. Korycki and B. Krawczyk, "Adversarial concept drift detection under poisoning attacks for robust data stream mining," *Machine Learning*, vol. 112, no. 10, pp. 4013–4048, 2023.

[197] V. Metsis, I. Androutsopoulos, and G. Paliouras, "Spam filtering with naive bayes-which naive bayes?" in *CEAS*, vol. 17. Mountain View, CA, 2006, pp. 28–69.

[198] L. Yan, "Workload characterization of spam email filtering systems," *International Journal of Network Security and Its Applications*, vol. 2, 01 2010.

[199] "What Is Anti-Spam? How Anti-Spam Works and Evaluating Solutions," https://perception-point.io/guides/email-security/what-is-anti-spam-how-anti-spam-works-and-how-to-evaluate-solutions/, [Accessed 22-09-2024].

[200] U. Nations, "UN Secretariat adopts climate action plan | Department of Management Strategy, Policy and Compliance," https://www.un.org/management/news/un-secretariat-adopts-climate-action-plan, 2019, [Accessed 22-09-2024].

FHE-SD Documentation

FHE-SD

The program and scripts of this project implement spam detection using Fully Homomorphic Encryption (FHE). It is based on the Concrete libraries provided by Zama and in a TFHE (Fast Fully Homomorphic Encryption over the Torus) compiler.

FHE enables performing computations on encrypted data directly without the need to decrypt it first. This prevents developers from accessing private user data and provides additional security in the event of data leaks.

Project is available at: https://gitlab.com/adrianorp/project-fhe-sd

(or in the zip file provided)

Purpose

This project contains the experimental stages of the thesis to obtain a master's degree in Information Security and Cyberspace Law from the Instituto Superior Técnico of the University of Lisbon (IST).

Table of Contents

Getting Started

Features Summary

- Extensive documentation with Sphinx.
- Command line parametrization and --help
- Dataset preprocessing compatible with Enron Corpus.
- FHE and On-clear model training and spam detection.
- Model training support on Linear SVC (SVM), Logistic Regression, Decision Tree, XGBoost.
- Several metrics for resource usage, accuracy, precision, etc. (some in separated scripts in the test directory).
- Test mode and real mode (test mode is advised for complete and precise metrics).

Installation

- **FHE-SD** works only in Linux or Windows Subsystem for Linux and may be installed with pip.
- **FHE-SD** requires Python `3.10`.
- It is recommended to use the *requirements.txt* to install the dependencies.
- Extract the project zip file in the desired Linux directory.

Pip

To install **FHE-SD**, run the following:

```
pip install -U pip wheel setuptools
pip install concrete-ml
```

To install all dependencies, run the following command inside project-fhe-sd directory:

```
pip install -r requirements.txt
```

Working with FHE-SD

Usage

- The main program is in module *perform_detection.py*, located in the root directory **project-fhe-sd**.
- stand-alone scripts are located in **..\project-fhe-sd\test**.

- To customize the FHE-SD runtime parameters edit the file *config.ini* located in **..\project-fhe-sd\config** or edit and execute the script *create_config_ini.py* located in the same directory.

For details on how to use FHE-SD run the help manual through the command line:

```
python perform_detection.py --help
```

Example

Execution on-clear scenario in test mode using Linear SVC algorithm:

```
python perform_detection.py --test-mode --email-data '<path_to_dataset>/enron-spam/1-preprocessed-small-set' --model-alg 'svc' --mode fhe
```

Limitations

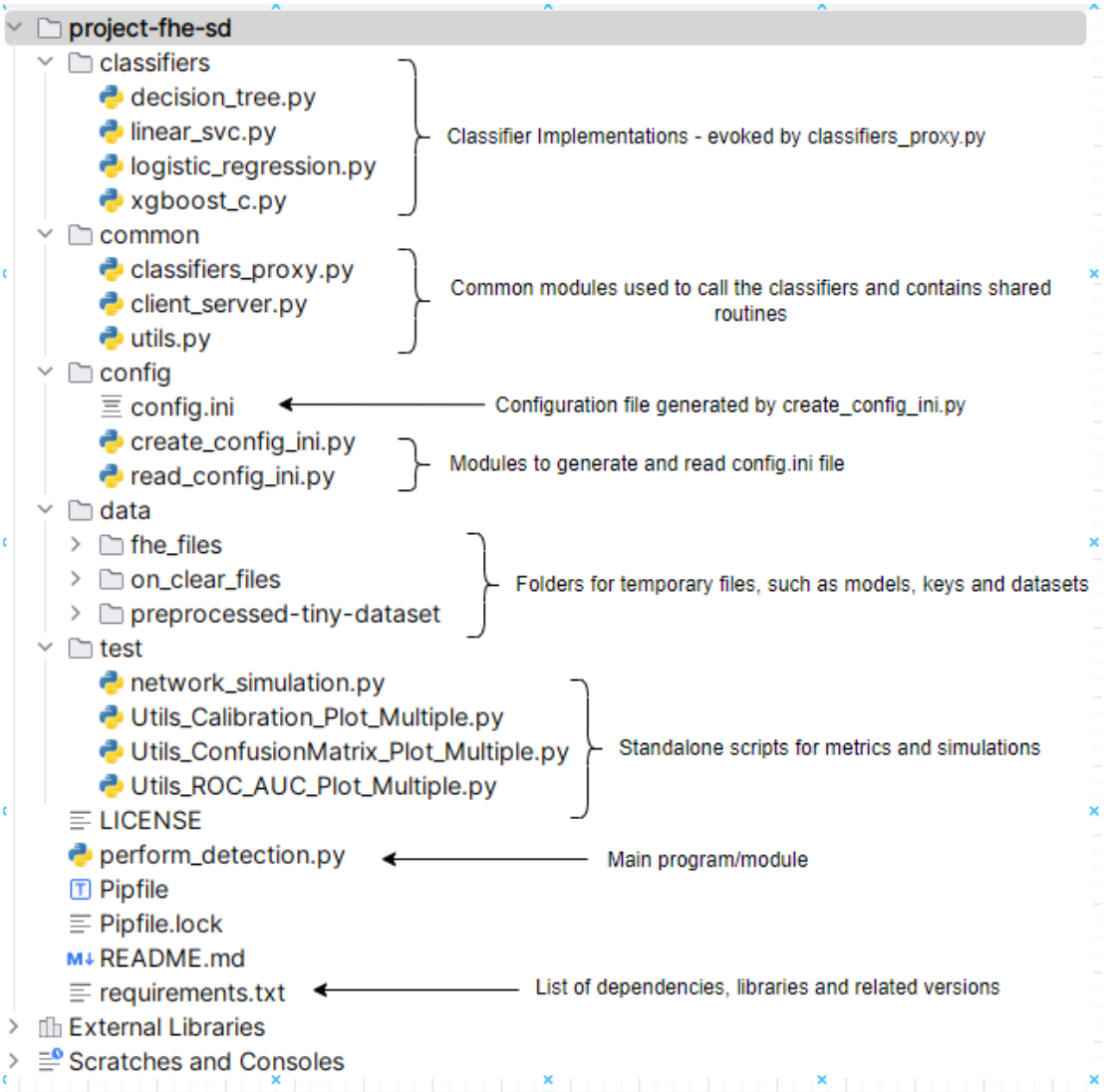
- Tree-based algorithms require `Concrete-ml (CML) 1.4` or higher for improved performance through rounded quantizers.
- Linear models are defective in specific scenarios using CML between versions `1.4` and `1.5` (Bug fixed in `CML 1.6.0`).
- The model weights and parameters are not encrypted.
- **FHE-SD** supports training only on plaintext/on-clear data. Support for model training on FHE-encrypted data was introduced in `CML version 1.6` after this project was concluded and for SGDClassifier only.

License

MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Note:This software uses several dependencies that have different licenses. It is the responsability of the user of this software to ensure they have the proper rights to the underlying dependencies.

FHE-SD Structure Overview



Basic Manual

Execute the command: *python perform_detection.py --help*

usage: perform_detection.py [-h] [--mode {fhe,clear}] [--email-data EMAIL_DATA] [--model-alg {svc,dectree,logreg,xgb}] [--test-mode]

This application is an study on spam detection using Fully Homomorphic Encryption (FHE) implemented using the Concrete libraries provided by Zama. The application supports two modes (test and real mode) and for comparison purposes it can be executed over FHE encrypted data and on-clear/plaintext data. Several metrics are supported to measure performance, resources usage and accuracy of the results. For details on Concrete library: https://docs.zama.ai/concrete-ml

On-Clear Real mode usage example:

$ python perform_detection.py --email-data '/../DATASET/enron-spam/4-preprocessed-small-set/enron1/spam/0006.2003-12-18.GP.spam.txt' --model-alg 'dectree' --mode clear

On-Clear test mode usage example:

$ python perform_detection.py --test-mode --email-data '/../DATASET/enron-spam/4-preprocessed-small-set' --model-alg 'dectree' --mode clear

FHE Real mode usage example:

$ python perform_detection.py --email-data '/../DATASET/enron-spam/4-preprocessed-small-set/enron1/spam/0006.2003-12-18.GP.spam.txt' --model-alg 'svc' --mode fhe

FHE test mode usage example:

$ python perform_detection.py --test-mode --email-data '/../DATASET/enron-spam/4-preprocessed-small-set' --model-alg 'svc' --mode fhe

options:

-h, --help show this help message and exit

--mode {fhe,clear} Define the type or mode of processing, where:

- fhe = Train models and predict using Full Homomorphic Encryption

- clear = Train models and predict on clear/plaintext (non-encrypted data)

--email-data EMAIL_DATA

When executed in real mode the the absolute path to the email file should be provided (for real mode just omit the argument --test-mode). If in test mode then the path to the dataset should be provided in place of the email file. An email file represents a email message to be checked for spam (real mode only) and the dataset path is where the dataset used for training is stored (--test-mode only).

--model-alg {svc,dectree,logreg,xgb}

Define the algorithm used to train the model and for the prediction, where:

+ svc = LinearSVC

+ dectree = DecisionTree

+ logreg = LogisticRegression

+ XGB = XGBoost

--test-mode The test mode uses a subset of the training data through train_test_split(). The model is trainedfrom scratch and saved to disk. If --test-data is omitted, then the program will be executed in real mode and load the model from disk.In real mode the input data is a feature matrix created from a single email file representing an email message (see --email-data). In real mode it is required to have the model saved to disk in advance. To generate a model file you must run the program in test mode first. Once you have the model file in disk it is not required to run in test mode again.