A Framework for Secure Logging and Analytics in Precision Healthcare Cloud-based Services

by

Parisa Moghaddam
M.Sc., K.N.Toosi University of Technology, 2013

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of

Master of Applied Science

in the Department of Electrical and Computer Engineering

A Framework for Secure Logging and Analytics in Precision Healthcare Cloud-based Services

by

Parisa Moghaddam
M.Sc., K.N.Toosi University of Technology, 2013

Supervisory Committee

Dr. Issa Traore, Supervisor
(Department of Electrical and Computer Engineering)

Dr. Imen Bourguiba, Departmental Member
(Department of Electrical and Computer Engineering)

ABSTRACT

Precision medicine is an emerging approach for disease treatment and prevention that delivers personalized care to individual patients by considering their genetic make-ups, medical histories, environments, and lifestyles. Despite the rapid advancement of precision medicine and its considerable promise, several underlying technological challenges remain unsolved. One such challenge of great importance is the security and privacy of precision health–related data, such as genomic data and electronic health records, which stifle collaboration and hamper the full potential of machine-learning (ML) algorithms. To preserve data privacy while providing ML solutions, this thesis explores the feasibility of machine learning with encryption for precision healthcare datasets. Moreover, to ensure audit logs' integrity, we introduce a blockchain-based secure logging architecture for precision healthcare transactions. We consider a scenario that lets us send sensitive healthcare data into the cloud while preserving privacy by using homomorphic encryption and develop a secure logging framework for this precision healthcare service using Hyperledger Fabric. We test the architecture by generating a considerable volume of logs and show that our system is tamper-resistant and can ensure integrity.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Dr. Issa Traoré, from the Faculty of electrical engineering at the University of Victoria for his continuous support. It would not have been possible to conduct this research without his encouragement and excellent mentorship. I am greatly appreciative to Dr. Issa Traoré who provided me an opportunity to be his research student in ISOT lab.

Chapter 1

Introduction

1.1 Context

Precision healthcare refers to the personalization of care to individuals. Despite the exciting prospects of precision healthcare, it faces several technical and societal hurdles related to the identification of health risks, diagnoses, and outcomes by analyzing data extracted from integrated biomedical databases. Security and privacy concerns are among these hurdles. While precision health provides tremendous benefits by enabling better care, it can lead to personal privacy breaches through genetic disclosure or genetic discrimination (treating differently because of a gene mutation). To deliver targeted, personalized care, personal data (e.g., specific human genome sequencing) must be shared with many professionals in possibly diverse geographic locations or jurisdictions and sometimes over unreliable channels, such as the internet. This poses several risks, such as insider threats, social engineering, distributed denial of service (DDoS), illicit data inferences, cyber bullying/blackmailing, etc. [41]. Unlike protected health information (PHI), precision health data, such as genomic data, not only identifies patients but also multiple generations of their families. Hence, such data can be leveraged to conduct targeted security and privacy attacks against vulnerable individuals or groups of related individuals if fallen in the hands of malicious actors.

Recent advancements in machine learning (ML) have led to significant progress towards personalized predictions [32]. In principle, a machine-learning model can be trained on either confidential or public data, allowing more training samples, data distributions, and therefore more complex, predictive, and generalizing models. These complex models can theoretically achieve higher predictive performance and find novel associations within precision healthcare. However, performing analytics on new cases provided by hospitals or medical centers should be treated with the utmost privacy preservation level for the reasons introduced above.

Nowadays, Cloud computing has become an ubiquitous technology in personalized predictions. While cloud services can enhance personalized predictions, sharing private healthcare data with cloud services is a challenging task since this technology suffers from severe security issues [42]. One way to deal with this is to use homomorphic encryption and let cloud services run machine learning models on encrypted data.

Audit logs are one of the critical assets of any enterprise systems. Audit logs keep a record of system events and in case of any system misuse or attack, log data can be processed to identify the responsible person in a forensic investigation. However, an attacker can modify audit logs in order to avoid detection. This is why maintaining integrity of audit logs is very important. In a cloud-based service scenario, logs can be compromised either on the server-side or the client-side. Since healthcare data is sensitive, we need to make sure that there is a secure logging mechanism for systems that interact with them.

1.2 Research Problem

The main research question explored in this thesis is about the feasibility of a secure framework which enables predictive machine learning for precision healthcare while ensuring both data and log security. Any privacy-preserving precision health analytics builds on two main components: (1) the security and privacy features required to protect interactions with the data by stakeholders and (2) ML predictive models for this data.

In our work, we propose a machine learning with encryption (MLE) framework and investigate the scope of secure logging in a precision healthcare cloud-based system using MLE. An important aspect of the machine learning framework is to consider needed requirements and constraints and facilitate performing analysis on a real precision healthcare dataset while preserving its privacy. Another key aspect is since logging is a crucial element in forensic investigations, it is essential to store logs securely and make logs tamper-resistant. Healthcare data are very sensitive data, and this level of sensitivity can be attractive to hackers or intruders. Audit logs can be very beneficial in any attacks or any unauthorized access to the system, but most of the time, the main problem is attackers try to remove their footprints by deleting or editing the audit logs. In this case, using audit logs could not be very helpful. One way to prevent such scenarios is to use a mechanism to ensure the integrity and confidentiality of the data while delivering the predictive capability offered by machine learning.

1.3 Approach Outline

To ensure data security in predictive ML for precision healthcare we use homomorphic encryption (HE). HE is a special cryptosystem which encrypts data in such a way that certain operations could be performed on them without possessing the secret encryption key (i.e., without decryption). The term “homomorphic encryption” describes a class of encryption algorithms that satisfy the homomorphic property; that is, certain operations, such as addition, can be carried out on ciphertext directly so that upon decryption the same answer is obtained as operating on the original messages. Therefore, HE allows other parties (e.g., the cloud and service providers) to calculate certain mathematical functions expressed only in terms of these operations on the encrypted data while preserving the function and format of the encrypted data.

For secure logging, we can use methods that rely on specialized hardware to achieve the goal. Hardware-based write-only devices are a cost-intensive solution, especially when there is a large amount of continuously generated log data [40]. Another secure logging mechanism is to use a trusted third party (TTP) entity. This approach makes the system rely on a third party. Most of the third-party auditors do not generally consider credibility and centralization, and such solutions may not be easily scalable [33]. Another approach that we explore in our work is to use Blockchain technologies. This technology can improve auditing by creating immutable logs [18].

1.4 Thesis Contribution

We make two contributions in this thesis. First, we propose a machine learning with encryption (MLE) framework that facilitates performing analysis on a real precision healthcare dataset while preserving its privacy. Second, we design a blockchain-based logging framework for precision healthcare services. In our secure logging framework, we assume that there are multiple data providers and one cloud-based service provider. Our proposed mechanism offers a solution to preserve the integrity of logs and provide access-protected storage based on blockchain. We use Hyperledger Fabric which is a permissioned blockchain system that makes it easier to control the transactions on the ledger and is typically faster when we compare it to public blockchains that are used in most cryptocurrencies [9].

A practical application of our framework is when a health practitioner initiates a request to find out if a patient can be prone to cancer or not. In this case, the health practitioner sends patient genomic data in a homomorphically encrypted format and gets back the machine learning scores from the server. In the client side there is an application that is deployed in the hospital side and client side users can use it to get the service from the machine learning service provider. The service provider sends the result in an encrypted format and the client side application needs to decrypt the result and make the decision about the possibility of cancer for the patient. Secure logging framework also makes sure logs that are generated in the transaction have integrity and are private.

1.5 Thesis Outline

The remaining chapters of this dissertation are structured as follows:

Chapter 2 will present an overview of the literature underlying this research, by providing background and review on homomorphic encryption on cloud services and secure logging with blockchain.

Chapter 3 will describe our proposed framework for a precision healthcare system which provides ML solutions while preserving privacy.

Chapter 4 will present a generic architecture for secure logging and extend the architecture by incorporating relevant modules for machine learning encryption framework.

Chapter 5 will present experiments to find parameters for homomorphic encryption and discuss the implementation of the proposed framework and secure logging architecture.

Chapter 6 will make concluding by discussing the proposed framework. In addition, it will suggest improvements for future works.

Chapter 2

Background and Related Works

In this chapter, we review works related to prediction models using homomorphic encryption and secure logging by blockchain. This chapter is organized as follows. Section 2.1 will discuss usage of homomorphic encryption in cloud services. Section 2.2 discusses works related to secure logging using blockchain.

2.1 Background and Review on Homomorphic Encryption on Cloud Services

2.1.1 Homomorphic Encryption

Encryption is the process of converting data from something intelligible into something unintelligible. The main purpose of this is to prevent unauthorized personnel from viewing this data [30]. In the cryptography field, homomorphism is used as an encryption type. Homomorphic encryption was originally proposed by [43] as a way to encrypt data such that certain operations can be performed on it without decrypting it first. Homomorphic Encryption (HE) is a kind of encryption scheme that allows a third party (e.g., cloud, service provider) to perform certain computable functions on the encrypted data while preserving the features of the function and the format of the encrypted data [1]. Like other types of encryption schemes, an HE scheme has three main functions: Gen (Key Generation), Enc (Encryption), and Dec (Decryption) [27]. The term homomorphic encryption describes a class of encryption algorithms which satisfy the homomorphic property: that is certain operations, such as addition, can be carried out on cipher texts directly so that upon decryption the same answer is obtained as operating on the original messages [6]. Encryption is a tool which essentially allows one to seal data in a metaphorical vault, which can only be opened by somebody holding the secret decryption key. Homomorphic Encryption (HE) allows other parties to operate on the data without possession of the secret key [31] A data mart that hosts the homomorphically encrypted data would perform the analyses and control the number of performed tests, with no knowledge about the raw data except its structure [38].

2.1.2 Fully Homomorphic Encryption (FHE)

Fully Homomorphic Encryption (FHE) was introduced for the first time in 2009 by Gentry [21]. Gentry in his Ph.D. thesis which was a seminal work in this scope discussed a new scheme of homomorphic encryption which enables performing an unlimited number of additions and multiplications. Before Gentry's work, existing schemes allowed only one algebraic operation on ciphertext; this limitation to only one single operation was very restrictive. Gentry's original scheme was highly inefficient [19]. Following Gentry's work, many FHE models have been introduced to make FHE more practical. One of these works is known as levelled FHE. Levelled FHE allows adding and multiplying encrypted input, but requires knowing in advance the complexity of the arithmetic circuit that is to be applied to the input. Recently, new implementations, data encoding techniques, and applications which help address some of the challenges have been introduced but much still remains to be done [14].

2.1.3 Basic Operations

As mentioned above, a key property of homomorphic encryption is the ability to perform certain basic operations such as addition and multiplication. Below is an example presented by [44] to introduce the high-level concept of homomorphic encryption:

1. Let m be the plaintext message
2. Let a shared public key be a random odd integer p
3. Choose a random large q, small r, ($|r| \leq p \div 2$)
4. Ciphertext c = pq + 2r +m (Ciphertext c is close to multiple of p)
5. Perform homomorphic addition/multiplication as required
6. Decrypt: m = (c mod p) mod 2

homomorphic addition can be illustrated as follows:

$$c1 = q1 * p + 2 * r1 + m1 \tag{2.1}$$

$$c2 = q2 * p + 2 * r2 + m2 \tag{2.2}$$

$$c1 + c2 = (q1 + q2) * p + 2 * (r1 + r2) + (m1 + m2) \tag{2.3}$$

and Homomorphic multiplication as follows:

$$c1 = q1 * p + 2 * r1 + m1 \tag{2.4}$$

$$c2 = q2 * p + 2 * r2 + m2 \tag{2.5}$$

$$c1*c2 = ((c1*q2)+q1*c2*q1*q2)*p+2(2*r1*r2+r1*m2+m1*r2)+m1*m2 \tag{2.6}$$

The most notable shortcoming of HE is that operations in practical schemes are limited to addition and multiplication [27]. Based on this shortcoming not all of the functions can be "HE-friendly". Formally, HE-friendly functions can be expressed as

$$Dec[k_s,\ Enc(k_p, m_1) \diamond Enc(k_p, m_2)] = m_1 \circ m_2, \tag{2.7}$$

where $k_s,\ k_p$ are the secret and public keys, respectively (since they are not equal, this is called "asymmetric encryption"), $m_1, m_2 \in M$ are two values on which we wish to perform encrypted operations on, M is the message space of the HE scheme (i.e., the set of all possible values acceptable by the scheme), and $\diamond, \circ$ are operations in encrypted and plain-text spaces, respectively.

2.1.4 Encryption Schemas in Homomorphic Encryption

An encryption scheme is said to be homomorphic if certain mathematical operations can be applied directly to the cipher text in such a way that decrypting the result renders the same answer as applying the function to the original unencrypted data. The remarkable properties of homomorphic encryption schemes are not without limitations, which typically include slow evaluation and the fact that the set of functions which can be computed in cipher text space is very restricted [6]. The computational complexity of the homomorphic encryption scheme depends primarily on the number of levels of multiplications to be carried out on the encrypted data [23]. In fact, all these different HE attempts can neatly be categorized under three types of schemes with respect to the number of allowed operations on the encrypted data as follows

[1]:

- Partially Homomorphic Encryption (PHE) allows only one type of operation with an unlimited number of times (i.e., no bound on the number of usages).
- Somewhat Homomorphic Encryption (SWHE) allows some types of operations a limited number of times.
- Fully Homomorphic Encryption (FHE) allows an unlimited number of operations for an unlimited number of times.

PHE schemes can only be used for particular applications, whose algorithms include only addition or multiplication operations[1]. In SWHE schemes that were proposed before the first FHE scheme, the size of the ciphertext grows with each homomorphic operation and hence the maximum number of allowed homomorphic operations is limited [1] and as the complexity of the function grows, the SWHE parameters become prohibitively large [27]. Despite the advantages of using HE schemes, they have some limitations. One is ciphertext size. The size of the message increases considerably by encryption. Another important limitation is related to the noise. After each operation, the amount of noise in ciphertext increases. Multiplication increases noise much more than addition [27]. FHE schemes enable the computation of arbitrary functions on encrypted data. This property makes FHE the most sophisticated homomorphic encryption scheme and the "holy grail" of modern cryptography [12]. The FHE scheme supports basic arithmetic computations on encrypted data. FHE supports an unlimited number of arithmetic operations. In spite of being a potential cryptographic technique, some of the FHE schemes remain quite impractical for real-world applications due to their computational overhead and the amount of resources that they require for computations [41].

2.1.5 Challenges in Using Homomorphic Encryption

One line of criticism against homomorphic encryption is its inefficiency, which is commonly thought to make it impractical for nearly all applications [19]. Theoretically sound, FHE schemes are not quite ready to be deployed for practical applications. Homomorphic implementations need efficient implementation of expensive mathematical operations which makes the operation order of magnitude slower than conventional software applications that operate on plaintext data [35]. Several constraints should be considered for choosing the right homomorphic encryption solution. First, using (Somewhat/Fully) HE schemes will lead to a huge ciphertext expansion (say from 2,000 to 500,000 or even 1,000,000 times according to the scheme and the targeted security level). Second, the underlying operations are intrinsically expansive, which will drastically penalize the global running time [10].

2.2 Background and Review on Secure Logging with Blockchain

2.2.1 Blockchain Framework

Unlike conventional databases that are managed by a central authority, a blockchain is a type of distributed database that stores append-only log of time-stamped records cryptographically protected from changing and revision [26]. The blockchain can be conceptualized as a state machine that runs on a network of computers, or nodes. The blockchain's state is stored in a ledger, which constitutes a full record of all transactions, or state transitions, that have ever occurred on the network [4]. The so-called ledger is a string of data blocks generated and chained cryptographically in a chronological manner, and each block can contain multiple transactions [33].

A blockchain is an immutable transaction ledger, maintained within a distributed network of peer nodes. These nodes each maintain a copy of the ledger by applying transactions that have been validated by a consensus protocol, grouped into blocks that include a hash that bind each block to the preceding block [28]. Blockchains distribute trust by having each node in the network manage its copy of the blockchain, but for each node to have its copy, they need a distributed mechanism to agree on the current state of the blockchain [25].This mechanism is known as a consensus protocol, and is the algorithm that forms the foundation for security, accountability and trust in a blockchain [47].

Blockchain inherently is resistant to modification of data because the ledger is replicated among participating nodes which makes a modification to the replica on the nodes to be meaningless. In the blockchain, nodes are working together to achieve agreement on the up-to-date snapshot of the ledger by the consensus algorithm. Some blockchains support smart contracts, which allow developers to encode arbitrary logic that is uploaded to, and executed by the nodes in the blockchain network. Blockchains are inherently slower than traditional databases, and therefore access control implemented on blockchains will be slower [25].

2.2.2 Permissionless Blockchain vs Permissioned Blockchain

Permissionless blockchain systems (like Bitcoin and Ethereum) employ peer-to-peer (P2P) networks of relays to disseminate transactions and blockchain updates throughout the network using a best-effort gossip protocol. Such P2P networks typically experience considerable churn, with relays joining, leaving, and rejoining the network at will [26]. Permissionless blockchains can be accessed and utilized by anyone with Internet access. Typically, in such networks, the participants are rewarded, self-sustainable, open-source and, therefore, have more support from the community [8].

On the other hand, permissioned blockchain systems (like Ripple, Corda, and Hyperledger) apply a clique of highly available validator nodes for agreeing on transactions and blocks [26]. Permissioned blockchain systems regulate transaction read and write permissions for users of the blockchains, allowing users to read-only transactions in which they took part [25]. The advantage of using a permissioned network is that it does not rely on a paid third-party service provider. Instead, the blockchain node operators provide the immutability service for each other. Since all members evenly share the operational cost, no additional costs arise besides operating the network [40]. A permissioned solution seems suitable for companies aiming for the competitiveness of blockchain technology while protecting sensitive information [8].

2.2.3 Hyperledger Fabric

Hyperledger Fabric [28] is a permissioned blockchain, namely a closed system, where one must obtain credentials to read the ledger or write to it [9]. Based on the access mechanism, blockchain can broadly be categorized into public blockchain, private blockchain, and consortium blockchain. In a public blockchain, all nodes are free to join or exit at will. However, nodes cannot access the blockchain in either private or consortium blockchain until the administrator has authorized them. Consortium blockchain offers better decentralization since the administrators comprise more than one organization (unlike a private blockchain) [33]. One of the well-known consortium blockchain systems is Hyperledger Fabric, which is a modular and extensible open-source system for deploying and operating blockchain [5].

In Hyperledger Fabric, smart contracts are implemented via a chaincode, an arbitrary program (e.g., in Java), executed by peers before a transaction can be recorded on the ledger. The chaincode has access to the current ledger and the details of the new transaction, and it determines whether that transaction will go through and what data to add to the ledger [9]. Chaincode is a critical element in a Fabric network, as it dictates the rules to be followed by member participants. It is run in Docker containers and is, thereby, isolated from the shared ledger [8].

Lu et al. [33] addressed Hyperledger fabric characteristics as follow:

- **Tamper-proof**: In Hyperledger fabric, once consensus is reached, the ledger will be maintained by all nodes. Hence, any change on a single node is invalid. Therefore, it is challenging to modify the contents of historical ledger.

- **Access permission**: Hyperledger Fabric uses Public Key Infrastructure (PKI) to build the Membership Service Provider (MSP) module, which is then used to generate digital certificates to identify and manage the members' identities.

- **Anonymity**: In Hyperledger Fabric, each entity publishes a transaction with a new pseudonym instead of using a constant pseudonym (like in public blockchain). To achieve anonymity, each transaction is accompanied by a zero-knowledge proof of the user, and others can only learn the validity but not the user's true identity. To achieve unlinkability, each zero-knowledge proof differs between transactions, even for the same user. Hence, no other entity can analyze these proofs to identify the user.

- **Efficient processing**: Hyperledger Fabric divides all nodes into three roles, namely: the endorsement nodes for executing and endorsing transactions, the ordering nodes for ordering and packaging transactions into blocks, and the normal nodes for publishing transactions to endorse nodes and receive new blocks from ordering nodes. Thus, this allows one to avoid a bottleneck situation during execution and ordering a single node. Hence, Hyperledger Fabric is more efficient than Bitcoin and Ethereum.

- **Faster consensus algorithm**: The consensus process in Hyperledger Fabric is faster than many public blockchains (e.g., Bitcoin). In Bitcoin, for example, the consistency among nodes is maintained by proof of work (PoW), which requires nodes to calculate a block hash value for the accounting right. This process takes approximately 10 minutes on average, making the throughput very low. However, in Hyperledger Fabric, the consensus module is designed to be pluggable and supports consensus algorithms, such as Practical Byzantine Fault Tolerance (PBFT) and Raft. The latter uses election to replace complex computations in Bitcoin, and thus achieves significant savings in time (i.e., significantly faster than PoW).

2.2.4 Using Blockchain to Provide Auditing Capabilities

Audit logs are an inevitable part of any software system. They can record all events and interactions of the system that can be used in case of any uncertainty in system's operations or attacks. Although audit logs help track changes made and ensure correctness in the system, they are vulnerable to a series of attacks that may compromise the system's integrity [3]. Secure logging means that all entries in the log must have a corresponding event that happened, and all events that occurred must have a corresponding log entry [45]. Although blockchain stems from cryptocurrency, many studies have investigated the adoption of blockchain in different application scenarios beyond the financial domain. That typically involves multiple parties with a conflict of interest, such as personal data sharing, supply chain, identity management and medical data management [34]. On the other hand, blockchain enables secure, transparent, and immutable record keeping in distributed systems without a trusted intermediary [3]. Distributed trust and data immutability of blockchains can also provide auditing capabilities [25]. This attractive trait can make blockchain an excellent candidate to be used in audit logging systems.

Schorradt et al. [45] elaborate that blockchain technology offers three security characteristics that make it a secure logging protocol. Firstly, it provides integrity protection via signatures. Secondly, it provides integrity protection via hash pointers, and thirdly, it uses the distribution of data across nodes, with each node having a full copy of the blockchain and being a verifier for overall blockchain integrity. Distributed trust and data immutability of blockchain provides wide opportunities for combating fraud, reducing operational costs and optimizing processes in the health care industry[15]. In [16], Chernyshev et al. proposed an audit logging architecture for electronic medical record systems by leveraging forensic as a service (FaaS) concept, focusing mainly on privilege misuse. They also investigated different types of treat actions (such as hacking, ransomware, phishing, privilege abuse, etc.) regarding healthcare data. Although they discussed forensic readiness in the context of health systems, they only offered a conceptual architecture. [34] investigated using blockchain to build a query genomic dataset audit trail. In this study, the authors mainly focus on the time and space efficiency of the log to reduce the time it takes to query the audit trail; to achieve this, they assumed the blockchain network has been well-established under a specific consensus algorithm. They started the experiment with a provided genomic data access log file and designed a mechanism to store and retrieve the logs based on Multichain. They offered a primary method and an enhanced version. Their implementation can be a compatible component for the existing blockchain platforms. In their enhanced implementation, they assumed that the percentage of reading operations is not that high, so they traded retrieval speed for storage cost. Since some of the blockchain systems use LevelDB as a backend database, they compared their two methods (baseline and enhanced) alongside LevelDB as reference. They claimed that their design could be adapted to any Blockchain framework with the help of an intermediary. Schorradt et al. [45] focused on using blockchain to enable secure logging in industrial control systems. They connected the Syslog functionality of a programmable logic controller to the public Ethereum blockchain network. Their experiment found that the transaction time for the public Ethereum blockchain harshly restricts the usefulness of this type of secure logging for industrial control systems. They also provided their system prototype and investigated challenges and security levels by increasing computational and network load. Their work provides practical usage of a public blockchain network and offers a good insight about using a public blockchain network and considerations about the implementation. Ahmad et al. [3] constructed a design schema named BlockAudit, which leverages Hyperledger Fabric as a blockchain framework to enable trustworthy audit logs. They showed the applicability of their design in online transaction processing (OLTP) systems. They implemented an end-to-end system that encompasses all steps from generating logs in a system to audit them. After designing an application, they integrated it into a blockchain network to construct a distributed peer-to-peer network. They converted generated audit logs in another format (JSON) to be used in blockchain nodes. They evaluated their architecture by measuring the latency over the consensus achieved by the peers by increasing the payload size. An interesting result about their implementation is that the latency actor rises considerably when the network size grows beyond 30 nodes and a visible increase in latency when the payload size changes from 5MB to 10MB.

Chapter 3

Machine Learning with Encryption (MLE) Framework

In this section, we discuss eight different scenarios for machine learning with encryption and propose a four-component system architecture to accommodate any of the eight scenarios. This architecture, illustrated in Figure 3.1, fulfills the privacy-preserving requirements that are mandatory for future ML-based precision medicine.

3.1 Machine Learning with Encryption

Any ML algorithm trains on some training dataset **tr**, fits a model‘s parameters $\mathcal{M}$, and finally tests on a testing dataset **ts**. Therefore, in principle, there are eight possible combinations or scenarios to introduce privacy via encryption to the learning process by encrypting (or leaving unencrypted) each of these three components.

Table 3.1 summarizes those eight scenarios; below, we provide more details on them. We use 0 to denote an unencrypted component, where it still can only be encrypted using the public key k_p of another component without having access to its private key k_s, and we use 1 to denote an encrypted component, where its private key k_s is

Table 3.1: The eight possible scenarios of encrypting the three components: the training dataset **tr**, the ML model parameters $\mathcal{M}$, and the testing dataset **ts**.

#	**tr** $\mathcal{M}$ **ts**	Literature	Dataset	ML	Enc. Library
0	0 0 0	Ordinary ML			
1	0 0 1	Our present approach	MSK	many	SEAL
		Dowlin et al. [19]	MINIST	NN	SEAL
		Hesamifard et al. [27]	MINIST, CIFAR-10	DNN	Helib
2	0 1 0	Bost et al. [11]	Multiple	NB, HP, DT	self-implementation
3	0 1 1	——			
4	1 0 0	Graepel et al. [23]	Wisconsin	FDA	Magma
		Aselett et al. [7]	Multiple	NB, RF	EncryptedStats
		Nandakumar et al. [37]	MNIST6	DNN7	HElib
5	1 0 1	—— Not possible under the current theory			
6	1 1 0	—— Not practical: training on encrypted data already produces encrypted model			
7	1 1 1				

not available for the other two components.

Scenario 0 is denoted by the binary combination 000, when **tr**, $\mathcal{M}$, and **ts** are all not encrypted; this is the typical ML paradigm, where no privacy is of concern.

Scenario 1 is denoted by 001, where only **ts** needs to be encrypted; this is the scenario of this thesis. In such a scenario, since the model has access to an unencrypted training set, such as a public dataset, only the client data **ts**, which could be patients' genome records, are sensitive. Although the standard homomorphic property as defined in (2.7) would imply that $\mathcal{M}$ must be encrypted with k_p to predict on an encrypted **ts**, this is not the case for our work, which leverages special techniques implemented in SEAL [36], that allow encryption with plain-text multiplication with the caveat that the results themselves are encrypted and can only be decrypted with k_s. Research exists in this category but in areas of application other than precision medicine. [19] showed that a cloud service is capable of applying a neural network (NN) to encrypted **ts** to make encrypted predictions and return them in encrypted forms. They constructed a convolution NN (CNN) model from the unencrypted MINIST dataset and then produced a simpler FHE-friendly version of the CNN constructed only from addition and multiplication operations so that the parameters could be encrypted using the public key of the private testing dataset **ts**. [27] developed new techniques to allow testing CNN on encrypted **ts**. First, they designed methods to approximate the activation functions commonly used in CNNs with low-degree, FHE-friendly polynomials. Then, they trained a CNN on unencrypted **tr** with the approximation polynomials instead of the original activation functions. Finally, they converted the trained CNN to make predictions on encrypted **ts**.

Scenario 2 is denoted by 010, where the model is trained on an unencrypted training dataset **tr**. However, the model parameters themselves are then encrypted, which may imply privacy in **tr**, as well if the training is pursued locally where **tr** resides. Although the testing data **ts** is denoted by 0, it must be sent to $\mathcal{M}$ encrypted with its public key, as it is not possible, according to the theory of FHE, to pursue binary operations on encrypted numbers (parameters of $\mathcal{M}$) and unencrypted numbers (**ts**), without the results being encrypted and only decryptable with the same k_s that can decrypt $\mathcal{M}$. The virtue of scenario 2 is that it entails more freedom in choosing the model $\mathcal{M}$ as opposed to scenarios 4–7, where **tr** is encrypted and a stringent limitation is incurred for choosing the model $\mathcal{M}$ that can train on encrypted data. We are not aware of any literature that applies scenario 2 explicitly; however, [11] provided a very nested, layered model that could be classified as 010 scenario, but without relying solely on HE. They implemented a decision tree, naive Bayes and hyperplane decision that could test (not train) on encrypted data and built their models using cryptographic "building blocks" that emphasized protecting the model parameters and test data. They also used garbled circuits to compare encrypted data, which allowed a construction of argmax with alterations to ensure the ordering was not leaked. These building blocks allowed the implementation of decision tree, naive Bayes, and hyperplane decision with some minor changes. The building blocks also allowed the construction of other ML methods and a combination of methods using AdaBoost, which the authors demonstrated.

Scenario 3 is like 2 in that the model is trained on an unencrypted **tr**, and $\mathcal{M}$'s parameters are then encrypted; however, the test dataset **ts** is also encrypted with a different k_s than $\mathcal{M}$. Since there is no known method in the literature that allows the use of binary operations on two numbers encrypted with different k_s, scenario 3 (011) is not theoretically feasible under the current theory of cryptography.

Scenario 4 is denoted by 100, where an ML model is trained on encrypted **tr** (as in scenarios 5–7, as well). Hence, the model $\mathcal{M}$ will have encrypted weights by product, and the testing data must be sent to $\mathcal{M}$ encrypted with the same public key of **tr**, as explained above. Therefore, the reason scenario 5 (101) is not theoretically possible is the same as scenario 3. Furthermore, scenarios 6 and 7 (11x) are not of any practical interest, since the produced encrypted $\mathcal{M}$ does not need further encryption; this is possibly the reason for the absence of literature on these two scenarios. Under scenario 4, [22] defined a fully confidential version of linear means and Fisher's discriminant analysis (FDA), which can train and test on encrypted data. Linear means are rewritten to avoid division when learning the weights. The resulting decision function returns a multiple of the original decision function with the same sign. However, FDA requires the inverse of the covariance matrix to obtain the feature weights. This is found using gradient descent, the r^{th} iteration of which is shown to be a d-degree polynomial, where $d = 2(r-1)+1$. [7] provided a completely random forest (CRF) implementation that could train and test on encrypted data. Among other alterations to the algorithm, the key difference was encoding feature values using one hot encoding after quantile partitioning. CRFs have important benefits, especially learning incrementally. The authors also provided a naive Bayes classifier that could train and test on encrypted data. [37] evaluated the feasibility of training NNs on

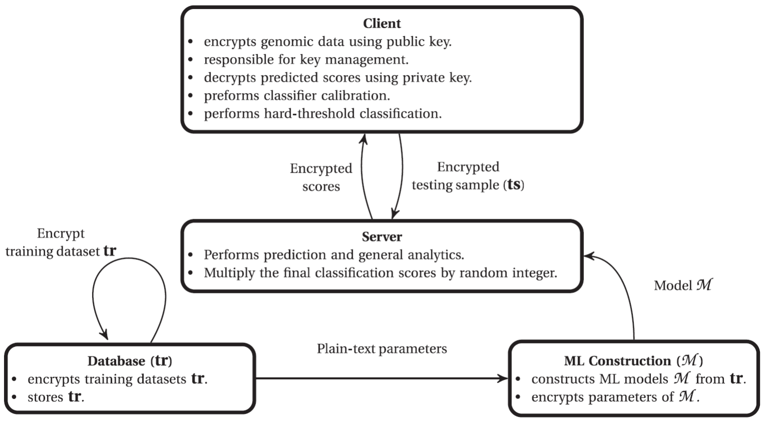
Client
encrypts genomic data using public key.
responsible for key management.
decrypts predicted scores using private key.
preforms classifier calibration.
performs hard-threshold classification.
Encrypted scores
Encrypted testing sample (ts)
Encrypt training dataset tr
Server
Performs prediction and general analytics.
Multiply the final classification scores by random integer.
Model $\mathcal{M}$
Plain-text parameters
Database (tr)
encrypts training datasets tr.
stores tr.
ML Construction ($\mathcal{M}$)
constructs ML models $\mathcal{M}$ from tr.
encrypts parameters of $\mathcal{M}$.

Figure 3.1: A block diagram of a privacy-preserving MLE framework for precision medicine

encrypted data completely non-interactively. His proposed system used the FHE toolkit HElib to implement stochastic gradient descent (SGD) for training. He used "ciphertext packing" to minimize the number of required bootstrapping operations and to enable the parallelization of computations at each neuron, thereby significantly reducing the computational complexity. This, in combination with simplifying the network architecture, allowed him to practically train neural networks over encrypted data despite the computational hurdles.

3.2 Proposed System

To prove the concept of our secure logging mechanism, we present a solution to analyze patients' health data in the cloud while preserving patients' privacy. Particularly, we propose a client application that enables health workers to submit sensitive health data and send them to the cloud and also an application on cloud. In the cloud, any operation will be done and the result will be sent back to the client application. Our solution makes use of homomorphic encryption to protect patients' data during the analysis. The proposed architecture depicted by Figure 3.1 consists of following components:

Database **(tr)** is a reservoir for both publicly available genetic datasets, which do not require preserving privacy, and private datasets, which need encryption prior to public sharing. Whenever a new dataset is revealed, it can be added to this reservoir for more accurate future analytics.

ML Construction **($\mathcal{M}$)** is the engine that constructs models—including transformation, feature selection, resampling, etc.—from the datasets in the *database* module. This module can be open-sourced for the entire community and can always be updated as new ML methods merge or more accurate models are constructed. In addition, the module can train on its own private dataset, which is not part of the *database* module, and then encrypt its model parameters $\mathcal{M}$. Alternatively, it can establish a protocol with the *database* module to train on the private dataset without encryption for a wider range of algorithms and then encrypt the model parameters to preserve the dataset's privacy (Cases 01x in Table 3.1).

Client **(ts)** is where the testing data, which is probably sensitive and confidential, resides and needs analytics. The owner of this data can opt to encrypt it, and this encryption can be provided via simple software components installed on the *client* side available via communication with the *server*. Next, the encrypted testing data is sent to the *server* for prediction. Finally, the encrypted predicted scores are received back. The *client* should be responsible for setting the threshold on the scores for the final hard decision or classification. This is to achieve a required level of aggressiveness to control the per-class sensitivity, such as in the case of the binary classification problem, in which the threshold provides the trade-off between the sensitivity and specificity and thus controls the operating point on the receiver operating characteristic (ROC) curve.

Server is the cloud engine for prediction. On the one hand, it interfaces with the *client* to receive the encrypted dataset for prediction and sends back encrypted predictions, and on the other hand, it interfaces with *ML construction* to receive a particular predictive model. Based on the underlying encryption scenario (Table 3.1), the *server* receives the appropriate public key from these two modules. In addition, for the C-class classification problem and for greater privacy preservation for the model and/or the dataset ($\mathbf{tr}$), the server can optionally multiply the scores $s_c(x), c = 1, \ldots, C$, where $x \in \mathbf{ts}$, by a random integer. This keeps the relative C scores unaffected. However, this disallows the *client* from inferring information about the model weights ($\mathcal{M}$) by sending pseudo-feature vectors in the form $x = (0, \ldots, 1, 0, \ldots)$ (only one feature is 1; the others are zeros).

To illustrate the utility of proposed architecture, we demonstrate how scenario 1 can be implemented in a very practical setup. When ML training is based on public data ($\mathbf{tr}$), the weights of the trained model $\mathcal{M}$ are deployed on the *server* in unencrypted form, while the queries ($\mathbf{ts}$) must be encrypted for security sensitivity. Under a hospital's public key, many parties may also be eligible to upload data (e.g., doctors and patients). The *server* is used for deploying ML implementations $\mathcal{M}$. In this case, the hospital sends encrypted data to the *server*. In the *server*, many computations can be done on the encrypted data and the results sent back to the hospital. Only the hospital can decrypt the data because the private key is provided only on the hospital side.

3.3 Summary

This chapter discussed different scenarios for MLE and introduced our proposed system. We explained the proposed system in detail and introduced the major components.

Chapter 4

Secure Logging Architecture

In this chapter, we present a generic architecture for secure logging and extend the architecture by incorporating relevant modules for MLE.

4.1 General Architecture

The proposed Secure logging architecture consists of two main parties: data providers and a service provider. Data providers are hospitals and clinics that have patients' healthcare data. There can be many healthcare data providers and each data provider may consist of multiple hosts. The service provider offers precision healthcare services like disease detection. Since the service provider has access to data from multiple data providers, it can provide a much better prediction using machine learning models on a larger dataset. As an example, a health practitioner can initiate a request to know if a patient is prone to getting cancer or not. In this case, the user sends the patient's data in homomorphically encrypted format along with the algorithm (e.g., Neural Network) and the data model that is going to be used (e.g., MSK-IMPACT). The service provider processes the request and runs the selected algorithm and sends back the score to the client. Finally, the client makes the decision about the cancer type based on the scores. As we mentioned in Chapter 3 data providers encrypt data by using homomorphic encryption before sending it to the service provider. Using homomorphic encryption allows computation to be performed directly on encrypted data. We define two types of audit logs, internal audit logs and external audit logs.

Internal audit logs store information related to the data provider organization when a host communicates with the cloud service provider.

External audit logs store information related to the actual query.
For example, internal logs store username and role of the person initiating a service call. On the other hand, external audit logs store fields like algorithm name, dataset, and patient info. We store an identifier in both type of logs to relate an internal audit log to its corresponding external audit log. The reason for creating two logs is that we do not want the cloud service provider to view the information related to the data provider. It only stores a hash of the internal log to ensure integrity.

We use a permissioned blockchain system called Hyperledger Fabric [5] that can ensure both confidentiality and integrity of audit logs. We designed two frameworks consisting of four pivotal components which we describe below.

Private Channel. Since data providers do not want other participating organizations to view their logs, transactions between a data provider and the service provider should be kept private. One solution is to use separate private channels for each data provider organization and the service provider.

Public Channel. To preserve the privacy of logs associated with a data provider, we can use a public channel as well. In this case, we need to handle privacy by applying some policies to private collections.

Private data collection. The aim of using private data collection (introduced in V1.2 of Hyperledger Fabric [29]) is to prevent the cloud service provider from viewing internal audit logs. With private collections, the hash of the internal log is kept on the ledger, and actual (internal) logs are held under the control of the data provider.

Smart Contract. A Smart contract determines the system's business logic and how transactions change the state of the ledger. Since transactions need to be signed before committing to the channel, it requires multiple endorsements from different organizations. This multiple endorsement policy can ensure one organization on a network could not tamper the ledger and use business logic that was not agreed to by all channel members. This signing process requires each organization to invoke and execute the smart contract, which then signs the transaction's output. If enough organizations sign the transaction and the outcome is consistent, this transaction can be considered a valid transaction and committed to the ledger. Endorsement policy specifies the organizations on the channel that need to execute the smart contract. Deployed smart contracts in our prototype handle the logic of creating audit logs (both internal and external) and reading logs based on the unique identifier. We can consider two approaches for designing an architecture for secure logging framework. For the first approach there are private channels and private data. This means each data provider has their own channel and the service provider is a member of all channels. In this architecture, we have one internal log collection in each channel, which is private to a data provider and one external audit log, accessible to both the data provider and the service provider. Figure 4.1 shows this architecture. In this architecture, given that we might have more than one data provider, many channels in the network can lead to more complicated maintenance. Handling many smart contracts for each channel also can be very cumbersome.

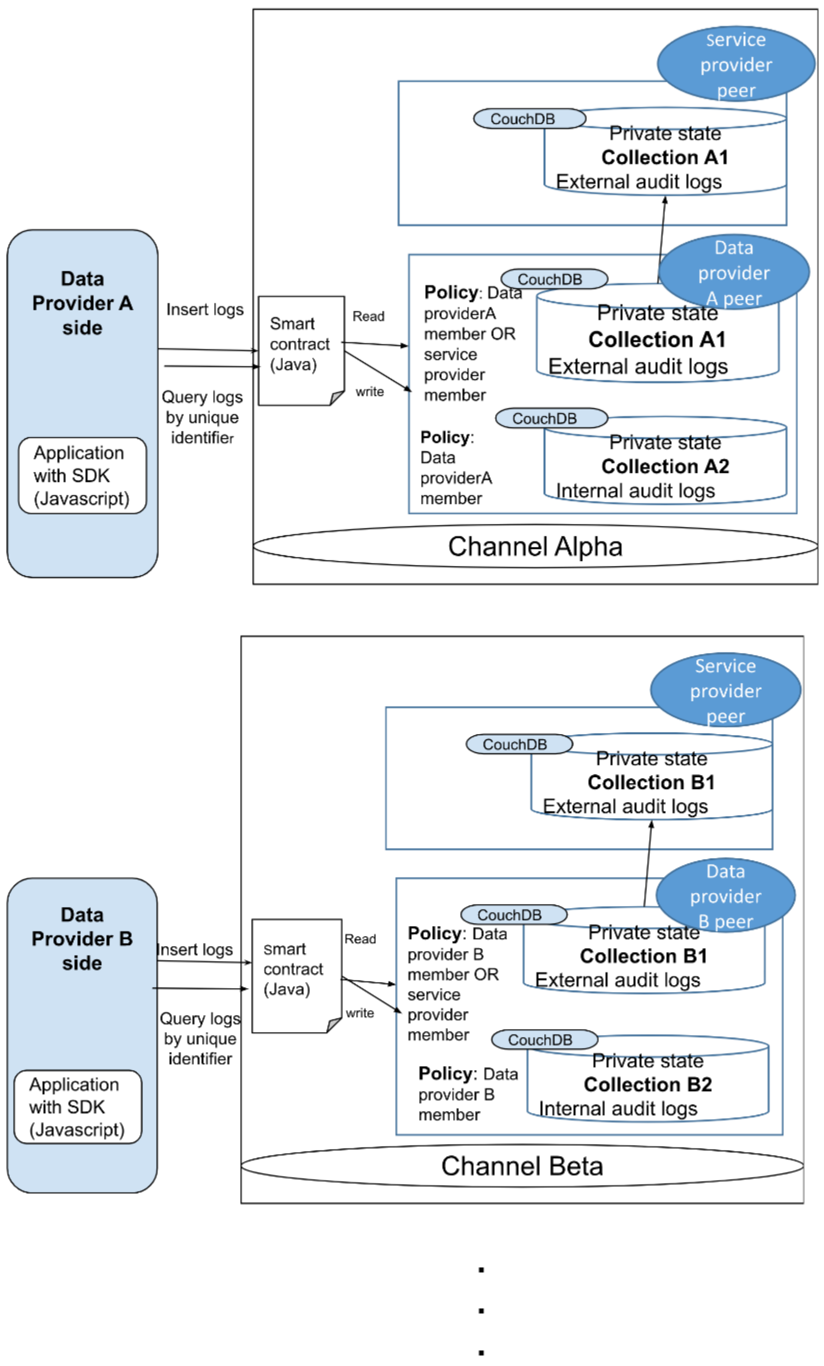
Service provider peer
CouchDB
Private state Collection A1 External audit logs
Data Provider A side
Application with SDK (Javascript)
Insert logs
Query logs by unique identifier
Smart contract (Java)
Read
write
Data provider A peer
CouchDB
Policy: Data providerA member OR service provider member
Private state Collection A1 External audit logs
CouchDB
Policy: Data providerA member
Private state Collection A2 Internal audit logs
Channel Alpha
Service provider peer
CouchDB
Private state Collection B1 External audit logs
Data Provider B side
Application with SDK (Javascript)
Insert logs
Query logs by unique identifier
smart contract (Java)
Read
write
Data provider B peer
CouchDB
Policy: Data provider B member OR service provider member
Private state Collection B1 External audit logs
CouchDB
Policy: Data provider B member
Private state Collection B2 Internal audit logs
Channel Beta

Figure 4.1: A secure logging architecture in a cloud-based precision healthcare service with private channels

For the second approach we can consider having a public channel and private data. This means all data providers and the service provider are members of one

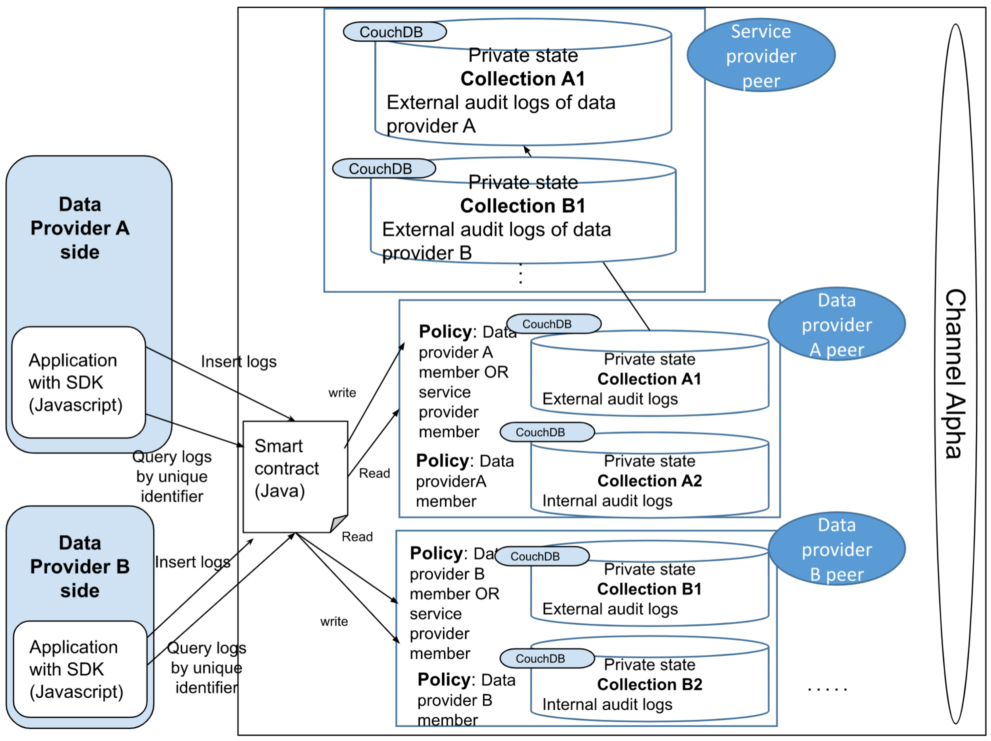

Figure 4.2: A secure logging architecture in a cloud-based precision healthcare service with a public channel

channel. Each data provider has an internal log collection accessible only to the data provider and an external log collection accessible to both the data provider and the service provider. Figure 4.2 shows this architecture. Since a private collection is a combination of the actual private data and a hash of the data, all channel members need to have an endorsed and ordered version of the hashes written on the ledgers. Accordingly, because each peer needs to store the hash version, we need more memory space.

4.2 Application to MLE Framework

The proposed secure logging framework can be used for the precision healthcare predictor system described in chapter 3. Figure 4.3 shows the transaction flow from the data provider to the Hyperledger network.

There are 6 main entities in this architecture as follows:

- **Data Provider Application**: This entity resides in the client side in the MLE block diagram in figure 3.1. An application needs to be deployed in the owner of sensitive data trusted servers. Each user (health practitioner) can install a client application or use a web application to comunicate with the system.

- **Service Provider**: This is the server side application which is responsible for providing the machine learning analysis.

- **Application with SDK**: This application is responsible to interact with with the blockchain ledger through chaincode. This application can be embedded in the data provider application and can be a standalone application that is only responsible for interacting with the ledger.

- **Endorsing Peers**: Endorsing peers are part of the authorized organizations of the collection and disseminate the private data to all the authorized peers (this should be based on the collection policy).

- **Authorized Peers**: Authorized peers are able to query private data. They will be determined based on the collection policy.

- **Ordering Service**: Ordering service is responsible for receiving submitted transaction from the application with the SDK and broadcasting a hash of the private data.

A user (healthcare practitioner) initiates a request to get service from the service provider. To relate an internal log to the corresponding external log, an identifier is added to this interaction. Users submitting a transaction need to have a valid private key to verify their identity to the network.

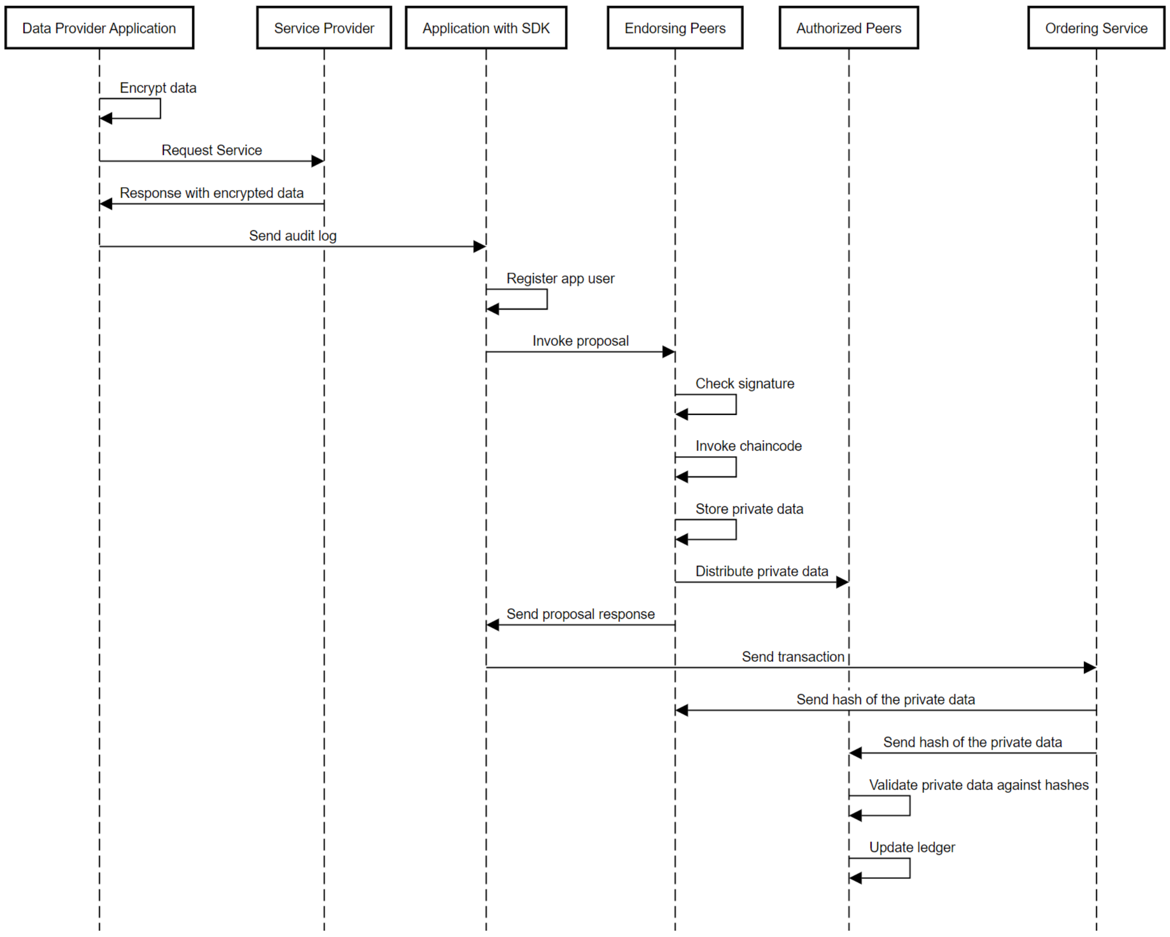

Figure 4.3: Sequence diagram of the system when inserting an audit log

There are different policies for internal and external log submission. For internal logs, an endorsement is needed from data provider peers and for external logs, an approval is needed from both data and service provider peers. To insert a new log, a transaction proposal is constructed to invoke the chaincode function to create logs. The user signs the proposal and endorsement peers will check it to make sure it is new and the signature is valid. Afterward, they will send the log to authorized peers.

In our scenario, authorized peers for the internal audit logs are from the data provider organization. For external logs, both data provider and service provider peers are authorized. For internal logs, after the application validates proposal responses, it sends a message with the channel id to the ordering service. The ordering service sends only a hash of the data to all authorized peers. In this way, the confidentiality of audit logs is preserved. For external audit logs, actual logs are sent to all authorized peers for storage.

4.3 Summary

This chapter we provided a general secure logging framework for healthcare system and presented a secure logging framework that could be used in such a precision healthcare system. We presented two approaches for this secure logging framework. In the next chapter, we present experiments that we conducted based on the selected approach.

Chapter 5

Implementation and Experiments

In this chapter, we describe the dataset, explain how to build the predictive model, demonstrate computational aspects of the encryption process for MLE and also build a secure logging mechanism for proposed system and finally testing our secure logging framework with a large amount of log data to examine security and performance.

5.1 MLE Implementation and Model Settings

In this section, we illustrate a cloud-based precision healthcare service as part of MLE framework and then apply our secure logging approach to the system.

5.1.1 MSK-IMPACT Dataset

To conduct our experiments, we use the MSK-IMPACT dataset. MSK-IMPACT, a clinical sequencing cohort dataset [48], comprises genomic patient records extracted from tumor-tissue samples taken from 10,336 patients. Since tumors are usually the results of many mutations, there are more than 100,000 discovered mutations. The dataset consists of 11 files linked together with `sample_ID` and `Patient_ID` and contains various information about the somatic mutations within the genomic samples, including mutation signature, copy number alternation, and gene fusion data files. '*With maturing clinical annotation of treatment response and disease-specific outcome"*, according to [48], "this dataset will prove a transformative resource for identifying novel biomarkers to inform prognosis and predict response and resistance to therapy. Tumor molecular profiling is a fundamental component of precision oncology, enabling the identification of genomic alterations in genes and pathways that can be targeted therapeutically".

The authors of the dataset tried to associate "biomarkers" with a particular type of cancer using simple methods of association, such as relative frequency. Then, to illustrate the usefulness of their DNA-sequence approach, they leveraged the Oncology Knowledge Base [13] to see how many of the mutations they detected (stratified by cancer type) were known to be actionable, that is, have an associated treatment or gene therapy.

5.1.2 Building the Model

In addition to the predictive power required for any ML model, the objective of privacy preservation via FHE requires the final ML model be FHE-friendly, that is, based only on addition and multiplication operations. Some ML models cannot satisfy both of these objectives. For example, a random forest (RF) has binary decision splits that are not FHE-friendly. However, although linear models (LM), logistic regression (LR), support vector machines (SVM), and many others are all HE-friendly, they may not perform well on a particular dataset. In this thesis, we used C-class LR which is HE-friendly.

5.1.3 Input Data and Needed Operations

For the experiment, we extracted separately 7791 feature vectors, each consisting of 5599 columns or features. The coefficients of the classifier were also provided in a separate file which has 22 rows and 5560 columns. The operation that was studied is as follows:

$$\Pr(G = c|X = x) = \frac{\exp(s_c)}{1 + \sum_{l=1}^{C-1} \exp(s_l)}, \tag{5.1a}$$

$$s_c = w_{c0} + w_c' x, \ \ c = 1, \dots, C, \tag{5.1b}$$

where, $C = 22$ types of cancer, the patient feature vector is $x \in \mathbf{R}^p, p = 5,599$; and the testing dataset **ts** to be encrypted has $N = 7,791$ patient records. By construction, the MLE framework requires sending the encrypted score of each testing observation to the client rather than the final hard decision for trading off the types of error. In addition, from (5.1), the numerator is a monotonic exponential function and the denominator is only for scaling, so probabilities sum to 1. Therefore, it is sufficient to encrypt the linear term s_c and treat it as the final score sent to the *client*.

5.1.4 Fully Homomorphic Encryption Libraries

Some of the homomorphic encryption schemes have implementations which, are publicly available. One of the most famous libraries which supports the Brakerski-Gentry-Vaikuntanathan (BGV) and Cheon-Kim-Kim-Song (CKKS) schemes is HElib [24]. HElib comes with many optimizations to make homomorphic evaluation run faster by mainly focusing on effective ciphertext packing techniques and optimizations [24]. Many related works used HElib [27, 37] and this library is the most important and widely utilized. It is designed using low-level programming, which deals with the hardware constraints and components of the computer without using the functions and commands of a programming language and hence is defined as "assembly language for HE" one [1]. Another notable library is SEAL [46]. SEAL is an open-source HE library developed by the cryptography and privacy research group at Microsoft. The library is written in C++ and can run in many environments. SEAL allows addition and multiplication to be performed on numbers. Other operations, such as encrypted comparison, sorting, and regular expressions, are in most cases not feasible on encrypted data using this technology. SEAL supports two FHE schemes: the Brakerski/Fan-Vercauteren (BFV) scheme, which allows modular arithmetic to be performed on encrypted integers, and the CKKS scheme, which allows addition and multiplication on encrypted real or complex numbers, but this latter scheme yields only approximate results. Other libraries like Fastest Homomorphic Encryption in the West (FHEW) [20], libScarab [39], Fast Fully Homomorphic Encryption Library over the Torus (TFHE) [17] are also major implementations. The computational performance of multiplicative homomorphic operations could be considered a factor in choosing between different libraries. Melchor et al. [2] conducted an experiment using large plaintext moduli with three different FHE libraries in order to evaluate their respective capabilities and performance. For this purpose, they used SEAL, Helib and FV-NFLlib to perform their experiments. They benchmarked for both libraries, 1 bit, 64 bits, 256 bits, and 2048 bits plaintexts. Based on different ciphertext modulus sizes, it turned out that for $\log p = 1$ where p is plain text length, SEAL v2.3 is the best choice up to a depth of around 12. Then SEAL v2.3 and HElib have similar performance until a depth of about 25, and above HElib outperforms all the other libraries. For $\log p = 60$, FV-NFLlib and SEAL v2.3 both beat HElib for all practical values (up to a depth slightly above 40). Based on these experiments, the authors believe that given that FV-NFLlib and SEAL result in similar performance and that SEAL is more actively developed and more user friendly, in practice, the

Table 5.1: Comparison between SEAL and HElib fully homomorphic encryption libraries

Feature	HElib	SEAL
Language Support	C++, Python	C#, C++, Python
Scheme Support	BGV, CKKS	BFV, CKKS
Operations Support	Addition, Multiplication, Bitwise operations, Square, Negation	Addition, Multiplication, Bitwise operations, Square, Negation

natural choice is SEAL v2.3.

Table 5.1 shows comparison between famous fully homomorphic libraries of SEAL and Helib. In this thesis, we decided to use Microsoft SEAL with BFV scheme to perform FHE operations. SEAL offers .NET library and is the only one that supports the C# language. SEAL is well documented and is easy to use when comparing with other libraries; there is just a challenging task to select encryption parameters which is a critical job. In some kinds of analysis like machine learning processes, using approximate results could impact the accuracy. That is the reason why we used the BFV scheme.

5.1.5 Homomorphic Operations On The Data

We applied the BFV scheme implementation of SEAL to perform this weighted summation term. The encryption operations are explained as follows. (1) Encrypt the feature list. (2) Multiply encrypted features by plain text weights and sum encrypted values. (3) Decrypt the results and repeat step 2 for each set of coefficients, i.e. each class.

Selecting parameters in SEAL library is very important because it can really impact the security level and performance of the implementation. Three important parameters are considered pivotal encryption parameters.

PolyModulusDegree: This parameter is the degree of the polynomial modulus. Selecting larger values for this parameter makes ciphertext larger and impacts on the speed of computations, on the other hand allows more complicated computations on ciphertext.

CoefModulus: This parameter is associated to noise budget, which means larger values for CoefModulus implies larger noise budget. The maximum value for this parameter is determined by the PolyModulusDegree.

PlainModulus: This parameter corresponds to the size of the plaintext data and noise budget consumption in multiplications.

In SEAL library (BFV scheme) every ciphertext has a quota by the name of noise budget. Due to the fact that there is a restriction in the number of operations on plaintext, arbitrary computations on encrypted data is not feasible. In this way, for each hmomorphic computation, this budget decreases. When the noise budget of a ciphertext reaches to zero, it makes it too corrupted to be decrypted. Finding optimum encryption parameters helps to prevent such situations.

5.1.6 Encryption and Parameters Selection

We tested different encryption parameters to compare computational time. Table 5.2 illustrates the computational time required as a function of a subset of the parameter space. Rows 3 and 4, caused the ciphertext noise budget to reach zero. This noise budget is determined by the encryption parameters, and once the noise budget of a ciphertext reaches zero, it becomes too corrupted to be decrypted. Thus, it is essential to choose parameters large enough to support the desired computations; otherwise, the correct result is impossible to obtain, even with the secret key. The values in row five give the best average per sample prediction time after testing on the entire dataset (7791 records), which spanned over seven days of computations on an i7core–2.5GHz–16G machine. From Eq. (5.1b), this time is obviously $T = NC\big((p+1)(E+M+A)+D\big)$, where N, C ,E, M, A, and D are the number of samples and the number of classes, encryption, multiplication, addition, and decryption times, respectively. Computation costs in terms of word size integer arithmetic for M and A are $O(n \log n)$ and $O(n)$ respectively. The BFV algorithm is manipulated on a polynomial ring $R = \mathbb{Z}[X]/(X^n + 1)$. In this ring, all the polynomials have degree at most $n - 1$. For modular reduction X^n, it equals to -1 in this ring, namely $X \equiv -1$, while X^{2n} equals to 1. The number of features (p) also impacts the computation cost since we need to run M in $p + 1$ times. During this experiment, `IntegerEncoder` was used to encode integers to BFV plain-text polynomials. `IntegerEncoder` is easy to understand and uses simple computations; however, there are more efficient approaches such as BatchEncoder, which can be investigated in future works.

Table 5.2: Effect of encryption parameters on encryption time. SEAL supports automatic selection of `CoeffModulus`. NA indicates noise ~ 0

#	polyModulusDegree	PlainModulus	Time in Sec
1	8192	2048	34560
2	2048	1024	960
3	1024	512	NA
4	1024	1024	NA
5	2048	1024	865
6	2048	512	875
7	2048	1499	939
8	2048	786433	894

During the first attempt, we encountered some issues with memory (Ram) utilization. Using parameter values in the first row required more than 16 Gb of Ram, which was not feasible based on the current system configuration. The solution to this issue was releasing the memory after each iteration manually. Releasing the memory ensures that everything is returned to the memory pool at the latest before running the next iteration. In rows 3 and 4, we observed that during the computations the ciphertext noise budget reaches zero. This noise budget is determined by the encryption parameters. We chose the parameter values in row 5 and experimented with all the feature list (7791 records). This experiment took 190 hours.

5.1.7 Time Complexity Analysis

5.1.8 Encryption Result

Based on our observation, using HE in a cloud-based precision healthcare system is entirely feasible because we could test our architecture on a real precision healthcare dataset and the response time was acceptable. To ensure security there are just some critical items like the power of the system, processing time and parameter selections that needs to be considered.

5.2 Secure Logging Framework Implementation

To implement our secure logging framework, we use Hyperledger Fabric, which is a permissioned Blockchain framework. In Section 3.2 we discussed two different architectures for secure logging. To employ our secure logging framework, we need to know what data are exchanged between client and server applications.

The client application calls the server API to get the prediction service. In this call, the client needs to send patient-related data input in a homomorphically encrypted format. Alongside patient data, the client sends type of algorithm that must be applied to the data (e.g. Neural network, Logistic regression, or any other machine learning algorithm). The client also needs to send which data set will be used. Currently, our MLE framework only supports logistic regression and MSK-IMPACT dataset. In response, the server application sends back the predicted scores in a homomorphically encrypted format, and an identifier assigned to this transaction. This identifier is useful when we want to store our audit logs. At the end, the client makes the decision based on scores.

As discussed in section 4.1, internal logs store information related to the data provider organization. This kind of data should be private to the data provider since they are related to its internal system such as the information about users of the data provider applications and their roles in the system. Below is a sample of the internal audit log that can be stored in Hyperledger fabric.

```
{
  "assetID": "0006dc82-9d6b-481a-8b89-40ea330e67c2",
  "name": "MARTIN BROWN",
  "role": "DR"
}
```

AssetID refers to the unique identifier that client app (data provider) sends to the ML service provider and ML service provider returns in response of the API call. The name refers to the person who initiated the request to call the service, and the role can be doctor, practitioner, nurse or any other roles defined in the service provider system. On the other hand we have the concept of external audit log which is the data that is shared between each data provider and the service provider. As an example below is an instance in JSON format of external audit log for data provider and the service provider.

```
{
  "assetID": "0006dc82-9d6b-481a-8b89-40ea330e67c2",
  "algorithm": "NN",
  "dataset": "MSK",
  "objectType": "AuditLog",
  "owner": "x509::CN=appUser1, OU=client + OU=org1 + OU=
      department1::CN=ca.org1.example.com, O=org1.example.
      com, L=Durham, ST=North Carolina, C=US",
}
```

5.2.1 Implementing secure logging with a public channel

As we described in chapter 3, our framework has three main components. In the following sections, we provide details of the implementation of these components.

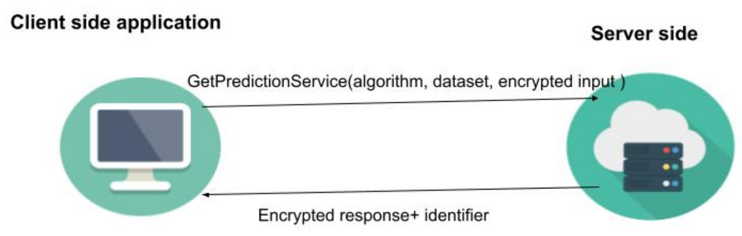
Client side application
Server side
GetPredictionService(algorithm, dataset, encrypted input)
Encrypted response+ identifier

Figure 5.1: Interaction between client application and the server side application

Private Data Collection

Private data collection can be configured to share with only a portion of authorized organizations, while public data can be shared with all organizations on a channel. Channel members approve collections, and they deploy when the chaincode definition is committed to the channel. All channel members need to have the same version of the collection file. A collection file is a JSON file containing an array of all channel collections with their properties.

Smart Contract Implementation

Before data providers and the service provider can communicate, they must agree on a contract. This contract can be composed of common terms, rules, definitions and processes. We can consider this a business model that governs all of the interactions between the service provider and data providers. A smart contract is an executable code that data providers and the service provider can invoke to generate transactions. In this thesis, we use Java language to develop our smart contract. There are some logics that we enforced by executing the smart contract. The current logic includes these operations:

- **ReadAsset (By assetId)** handles reading logs based on the unique identifier.
- **CreateAsset** handles the logic of creating audit logs (both internal and external)

Applications run by members of the network can invoke smart contracts to create assets on the ledger. There is an endorsement policy that specifies the set of peers on a channel that must execute smart contract and endorse the execution results in order for the transaction to be considered valid.

For private data collections, we can also specify an endorsement policy at the private data collection level, which would override the chaincode level endorsement policy for any keys in the private data collection, thereby further restricting which organizations can write to a private data collection.

Private channel

In our experiment one data provider (organization A) and the ML service provider (organization B) are part of a private channel.

5.2.2 Implementing Secure Logging With a Private Channel

To prevent storing extra hash on each peer need, we decided to choose a public channel to perform secure logging. For this reason, we implemented a Hyperledger Fabric with 2 organizations. Organization A which is a data provider and organization B which is a ML service provider. The channel comprised these two organizations. Since in our scenario, we want to make internal audit logs to be private to organization A, we need to have one private data collection to store organization A's internal audit log. We implemented this architecture and observed its behaviour in practice.

5.3 Framework Evaluation

In this section, we evaluate our framework in terms of security and performance.

Security. In the proposed framework, data and service providers are not able to delete/tamper logs since only read and write methods are implemented in the smart contract. If any party wants to add functions like delete or edit, all participating organizations need to agree on that to be approved. The current policy requires that a majority of channel members approve a new chaincode. As a result, an attacker needs

Table 5.3: Time to insert and query logs in our framework with different number of logs

Number of log records	insert (ms)	query one record(ms)
100	236453.43	37.34
500	1183543.31	61.47
10000	25437930.40	101.93

to compromise a majority of hosts from both data and service provider organizations to be able to modify logs which is very difficult.

Different data providers interact with the service provider through different channels which maintains the confidentiality of the logs. As mentioned before, based on the defined policy, the service provider can not view data providers' internal audit logs since only the hash is stored there.

To implement access control, we can easily restrict access of different service and data provider hosts using chaincode.

Performance. We tested the framework on a Ubuntu 20.04 virtual machine on a Windows host with Core i5 (1.10 GHz) processor and 8GB RAM. Table 5.3 shows the results of the experiment. We started by inserting 100 audit log entries and then we increased the number of log entries to 10000. In a cloud-based precision healthcare service scenario, it is unlikely that a data provider organization will request more than a hundred services in a quick succession. As a result, the insertion time is reasonable. We also present the time to query one record based on the identifier inserted. When the blockchain stores 10,000 logs, the time is 101.93 ms.

In Table 5.4, we show the disk usage on the data provider side. The service provider does not store the actual internal audit log data and stores only the hash of the internal audit logs; therefore, it needs less space than the data provider. As we can see from the table, disk requirement is minimal if we store text-only logs.

Table 5.4: Disk usage on the data provider

Number of log records	External log collection size (KB)	Internal log collection size (KB)	Hashed internal log collection size (KB)	Hashed external log collection size (KB)
100	50.8	34.4	42.8	42.8
500	248.2	166.1	209.3	209.3
1000	495	330.6	412.1	412.1
10000	4800	3200	4000	4000

Chapter 6

Conclusion and future work

6.1 Contribution Summary

We proposed a machine learning with encryption (MLE) framework that enables performing analysis on healthcare data while preserving its privacy. To ensure data security in the proposed framework we used homomorphic encryption. The proposed framework consists of four components: database which is a reservoir for both publicly available genetic datasets and private datasets, ML Construction which is the engine that constructs models, client which is where sensitive healthcare information resides and the server which is the cloud engine for prediction. In this framework, a hospital sends encrypted data to the server. In the server, many computations can be done on the encrypted data and the results are sent back to the hospital. Only the hospital can decrypt the data. Based on our observation, using HE in MLE is entirely feasible but it can be slow when comparing to other types of encryption especially when we need to perform multiplication on two ciphertexts. In our selected scenario in which only testing data set needs to be encrypted, for each request HE, computation takes 15 minutes which is acceptable in this system. We conducted different experiments for encryption parameter selection to ensure security and time efficiency but we observed some sluggishness in the encryption phase of the framework.

Furthermore, we designed a blockchain-based logging framework for precision healthcare services. We developed this framework for cloud-based precision healthcare services employing Hyperledger Fabric to provide a secure and auditable forensic-ready logging mechanism. We defined two types of audit logs, internal audit logs and external audit logs. To prevent storing extra hash when using private data collections, we decided to choose a public channel to perform secure logging. The channel comprised a data provider and the service provider. Since internal audit logs have to be private to the data provider, we considered one private data collection to store the data provider's internal audit log. The proposed framework is tamper-resistant and can ensure the security, integrity and confidentiality of healthcare audit logs. We tested the framework by inserting up to 10000 log entries. For this experiment, we observed insertion time, query on a record time and disk usage. In our preliminary experiments, the framework showed reasonable performance in terms of query time and disk utilization.

6.2 Future Work

Currently, MLE only works only with MSK-IMPACT dataset, accordingly the current secure logging implementation is developed with one data provider. One of the areas of interest for our future work is to test our secure logging framework with multiple data providers. These data providers will have different private data and might need different ML services. In this case, we would have more organizations and we can observe the performance of the framework when dealing with large amounts of log entries that are produced with multiple data providers.

Another area of interest is to implement the framework with a public channel. We expected to need more disk space but instead we need less complicated configurations for Hyperledger Fabric which can be a good positive point.

Based on the results, the framework has enormous potential to be used in practice and further research is necessary to improve its performance.

Bibliography

[1] Abbas Acar, Hidayet Aksu, A. Selcuk Uluagac, and Mauro Conti. A survey on homomorphic encryption schemes: Theory and implementation. *ACM Comput. Surv.*, 51(4):1–35, jul 2018.

[2] Carlos Aguilar Melchor, Marc-Olivier Kilijian, Cédric Lefebvre, and Thomas Ricosset. A comparison of the homomorphic encryption libraries HElib, SEAL and FV-NFLlib. In Jean-Louis Lanet and Cristian Toma, editors, *Innovative Security Solutions for Information Technology and Communications*, pages 425–442, Cham, 2019. Springer International Publishing.

[3] Ashar Ahmad, Muhammad Saad, Mostafa Bassiouni, and Aziz Mohaisen. Towards blockchain-driven, secure and transparent audit logs. In *ACM International Conference Proceeding Series*, pages 443–448. Association for Computing Machinery, nov 2018.

[4] Jessie Anderson and Sean Smith. *Securing, standardizing, and simplifying electronic health record audit logs through permissions blockchain technology.* PhD thesis, Dartmouth College, 2018.

[5] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. Hyperledger fabric: a distributed operating

system for permissioned blockchains. In *Proceedings of the thirteenth EuroSys conference*, pages 1–15, 2018.

[6] Louis J M Aslett, Pedro M Esperan ç a, and Chris C Holmes. A review of homomorphic encryption and software tools for encrypted statistical machine learning. 2015.

[7] Louis J M Aslett, Pedro M Esperan ç a, and Chris C Holmes. Encrypted statistical machine learning: new privacy preserving methods. *arXiv:1508.06845v1*, 2015.

[8] Rafael Belchior, Miguel Correia, and André Vasconcelos. JusticeChain: Using blockchain to protect justice logs. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 11877 LNCS, pages 318–325, 2019.

[9] F. Benhamouda, S. Halevi, and T. Halevi. Supporting private data on Hyperledger Fabric with secure multiparty computation. *IBM Journal of Research and Development*, 63(2), mar 2019.

[10] Guillaume Bonnoron, Caroline Fontaine, Guy Gogniat, Vincent Herbert, Vianney Lapôtre, Vincent Migliore, and Adeline Roux-Langlois. Somewhat/fully homomorphic encryption: Implementation progresses and challenges. In *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, volume 10194 LNCS, pages 68–82, 2017.

[11] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine Learning Classification over Encrypted Data. 2015.

[12] Alycia Carey. On the explanation and implementation of three open-source fully homomorphic encryption libraries. Undergraduate Thesis, University of Arkansas, Fayetteville, 2020.

[13] Debyani Chakravarty, Jianjiong Gao, Sarah Phillips, Ritika Kundra, Hongxin Zhang, Jiaojiao Wang, Julia E. Rudolph, Rona Yaeger, Tara Soumerai, Moriah H. Nissan, Matthew T. Chang, Sarat Chandarlapaty, Tiffany A. Traina, Paul K. Paik, Alan L. Ho, Feras M. Hantash, Andrew Grupe, Shrujal S. Baxi, Margaret K. Callahan, Alexandra Snyder, Ping Chi, Daniel C. Danila, Mrinal Gounder, James J. Harding, Matthew D. Hellmann, Gopa Iyer, Yelena Y. Janjigian, Thomas Kaley, Douglas A. Levine, Maeve Lowery, Antonio Omuro, Michael A. Postow, Dana Rathkopf, Alexander N. Shoushtari, Neerav Shukla, Martin H. Voss, Ederlinda Paraiso, Ahmet Zehir, Michael F. Berger, Barry S. Taylor, Leonard B. Saltz, Gregory J. Riely, Marc Ladanyi, David M. Hyman, José Baselga, Paul Sabbatini, David B. Solit, and Nikolaus Schultz. Oncokb: A precision oncology knowledge base. *JCO Precision Oncology*, (1):1–16, 2017.

[14] Hao Chen, Kyoohyung Han, Zhicong Huang, Amir Jalali, and Kim Laine. Simple Encrypted Arithmetic Library v3.2.0. *Tech. Rep.*, 2018.

[15] Shekha Chenthara, Khandakar Ahmed, Hua Wang, and Frank Whittaker. A Novel Blockchain Based Smart Contract System for eReferral in Healthcare: HealthChain. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 12435 LNCS, pages 91–102. Springer Science and Business Media Deutschland GmbH, 2020.

[16] Maxim Chernyshev, Sherali Zeadally, and Zubair Baig. Healthcare Data Breaches: Implications for Digital Forensic Readiness. *Journal of Medical Sys-*

tems, 43(1), 2019.

[17] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption library, August 2016. https://tfhe.github.io/tfhe/(accessed August 2021).

[18] Erikson Júlio De Aguiar, Bruno S Faiçal, Bhaskar Krishnamachari, and Jó Ueyama. A Survey of Blockchain-Based Strategies for Healthcare. 53(2), 2020.

[19] Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *33rd Int. Conf. Mach. Learn. ICML 2016*, volume 1, pages 342–351, 2016.

[20] Lèo Ducas and Daniele Micciancio. FHEW: A fully homomorphic encryption library, 2014. https://github.com/lducas/FHEW(accessed August 2021).

[21] Craig Gentry. *Fully Homomorphic Encryption Scheme.* PhD thesis, Stanford University, 2009.

[22] T. Graepel, K. Lauter, and M. Naehrig. Ml confidential: machine learning on encrypted data. In T. Kwon, M.-K. Lee, and D. Kwon, editors, *Information Security and Cryptology - ICISC 2012 (15th International Conference, Seoul, Korea, November 28-30, 2012, Revised Selected Papers)*, Lecture Notes in Computer Science, pages 1–21, Germany, 2013. Springer.

[23] Thore Graepel, Kristin Lauter, and Michael Naehrig. ML confidential: Machine learning on encrypted data. In *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, volume 7839 LNCS, pages 1–21, 2013.

[24] Shai Halevi and Victor Shoup. HElib (release 2.1.0). https://github.com/homenc/HElib(accessed August 2021), 2013. IBM Research.

[25] Taylor Hardin and David Kotz. Blockchain in Health Data Systems: A Survey. In *2019 6th International Conference on Internet of Things: Systems, Management and Security, IOTSMS 2019*, pages 490–497. Institute of Electrical and Electronics Engineers Inc., oct 2019.

[26] Ryan Henry, Amir Herzberg, and Aniket Kate. Blockchain access privacy: Challenges and directions. *IEEE Security and Privacy*, 16(4):38–45, jul 2018.

[27] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. CryptoDL: Deep Neural Networks over Encrypted Data. *arXiv:1711.05189v1*, 2017.

[28] Hyperledger. Introduction on hyperledger fabric [online]. https://hyperledger-fabric.readthedocs.io/en/release-2.2/whatis.html, 2021. Accessed: 2021-06-15.

[29] Hyperledger. Using private data in fabric [online]. https://hyperledger-fabric.readthedocs.io/en/release-1.2/private-data/private-data.html, 2021. Accessed: 2021-06-06.

[30] Adam Richard Jones. Homomorphic Encryption Within The NHS Using Paillier, 2018.

[31] Miran Kim and Kristin Lauter. Private genome analysis through homomorphic encryption. Technical report, 2015.

[32] Yikuan Li, Shishir Rao, José Roberto Ayala Solares, Abdelaali Hassaine, Rema Ramakrishnan, Dexter Canoy, Yajie Zhu, Kazem Rahimi, and Gholamreza Salimi-Khorshidi. Behrt: transformer for electronic health records. *Scientific reports*, 10(1):1–12, 2020.

[33] Ning Lu, Yongxin Zhang, Wenbo Shi, Saru Kumari, and Kim Kwang Raymond Choo. A secure and scalable data integrity auditing scheme based on hyperledger fabric. *Computers and Security*, 92, may 2020.

[34] Shuaicheng Ma, Yang Cao, and Li Xiong. Efficient logging and querying for blockchain-based cross-site genomic dataset access audit. *BMC Medical Genomics*, 13(S7):91, jul 2020.

[35] Ahmet Can Mert, Erdinc Ozturk, and Erkay Savas. Design and Implementation of Encryption/Decryption Architectures for BFV Homomorphic Encryption Scheme. *IEEE Trans. Very Large Scale Integr. Syst.*, 28(2):353–362, feb 2020.

[36] Microsoft. GitHub - microsoft/SEAL: Microsoft SEAL is an easy-to-use and powerful homomorphic encryption library., 2019.

[37] Karthik Nandakumar. Towards Deep Neural Network Training on Encrypted Data. *IEEE Conf. Comput. Vis. Pattern Recognit. Work.*, 2019.

[38] Silvia Paddock, Hamed Abedtash, Jacqueline Zummo, and Samuel Thomas. Proof-of-concept study: Homomorphically encrypted data can support real-time learning in personalized cancer medicine. *BMC Med. Inform. Decis. Mak.*, 19(1), 2019.

[39] Henning Perl, Michael Brenner, and Matthew Smith. libScarab: An implementation of the fully homomorphic smart- vercauteren cryptosystem, December 2013. https://github.com/hcrypt-project/libScarab(accessed August 2021).

[40] Benedikt Putz, Florian Menges, and Günther Pernul. A secure and auditable logging infrastructure based on a permissioned blockchain. *Computers and Security*, 87, nov 2019.

[41] Mohammad Saidur Rahman, Ibrahim Khalil, Abdulatif Alabdulatif, and Xun Yi. Privacy preserving service selection using fully homomorphic encryption scheme on untrusted cloud service platform. *Knowledge-Based Syst.*, 180:104–115, sep 2019.

[42] Sagar Rane and Arati Dixit. Blockslaas: Blockchain assisted secure logging-as-a-service for cloud forensics. In Sukumar Nandi, Devesh Jinwala, Virendra Singh, Vijay Laxmi, Manoj Singh Gaur, and Parvez Faruki, editors, *Security and Privacy*, pages 77–88, Singapore, 2019. Springer Singapore.

[43] Ronald Rivest, Len Adleman, and Michael Dertouzos. On Data Banks And Privacy Homomorphism. Technical report, Massachusetts Institute of Technology, 1978.

[44] Sai Sri Sathya, Praneeth Vepakomma, Ramesh Raskar, Ranjan Ramachandra, and Santanu Bhattacharya. A Review of Homomorphic Encryption Libraries for Secure Computation. *arXiv:1812.02428v2*, 2018.

[45] Stefan Schorradt, Edita Bajramovic, and Felix Freiling. On the feasibility of secure logging for industrial control systems using blockchain. In *Proceedings of the Third Central European Cybersecurity Conference*, CECC 2019, New York, NY, USA, 2019. Association for Computing Machinery.

[46] Microsoft SEAL (release 3.3). https://github.com/Microsoft/SEAL (accessed August 2021), June 2019. Microsoft Research, Redmond, WA.

[47] Abdul Wahab and Waqas Mehmood. Survey of consensus protocols, 2018.

[48] Ahmet Zehir, Ryma Benayed, Ronak H Shah, Aijazuddin Syed, Sumit Middha, Hyunjae R Kim, Preethi Srinivasan, Jianjiong Gao, Debyani Chakravarty, Sean M Devlin, et al. Mutational landscape of metastatic cancer revealed from

prospective clinical sequencing of 10,000 patients. *Nature medicine*, 23(6):703, 2017.