

FlexFHE:  
A System for Homomorphically Encrypting DNA and Operating on  
Encrypted Data Securely in Untrusted Environments

Lior Attias  
Department of Computer Science  
Columbia University  
lra2135@columbia.edu

June 22, 2023

## ABSTRACT

DNA data contains sensitive health information and personally identifiable data. Currently, even if DNA data is stored in encrypted databases, it must be decrypted for health professionals and researchers to analyze, which means that DNA data exists in plaintext on unsecured, untrusted servers and machines during analysis. This thesis describes a complete system for homomorphically encrypting DNA data in a trusted context and then running analytic operations on the encrypted DNA data in an untrusted context, thus allowing healthcare professionals and researchers to run both high volume analytics on many individuals' sequenced DNA and run complex analytics on a single individual's sequenced DNA without ever handling plaintext data.

Symmetric encryption is used as a mechanism for controlling which queries are made on the data. The threat model addressed by this system allows an authorized party to run only authorized queries on a genome, while restricting any additional access.

The system implemented achieves substring search, substring search with wildcards representing mutations, and percent match between two nucleotide sequences by converting genomic data into one-hot binary matrixes and encrypting each bit individually using OpenFHE's LWE Encryption implemented using the CGGI scheme. While runtime for each operation is  $O(nm)$ , each operation is maximally parallelized using OpenMP, thus allowing for accelerated performance on machines with multiple CPUs without the need for batching.

**Keywords:** Fully Homomorphic Encryption, homomorphic substring search, OpenFHE, LWE Encryption, Genomic privacy, parallelized encrypted search on sequenced DNA, CGGI

# TABLE OF CONTENTS

<b>ABSTRACT .....</b>	<b>2</b>
KEYWORDS .....	2
<b>1 INTRODUCTION .....</b>	<b>6</b>
1.1 INTRODUCTION TO DNA PRIVACY CONCERNS .....	6
1.1.1 <i>Genomic data has expanded with better sequencing technology</i> .....	6
1.1.2 <i>A need for anonymizing data and controlling partial database access</i> .....	6
1.1.3 <i>Genomic data is badly protected by existing regulations</i> .....	8
1.1.4 <i>Summary of goals</i> .....	8
1.2 SYSTEM OVERVIEW AND USE CASES.....	8
1.2.1 <i>Use case: trusted and untrusted environments</i> .....	9
1.2.2 <i>Use case: real world examples</i> .....	9
1.3 THREAT MODEL .....	9
1.3.1 <i>What are is being protecting from whom, and how can the threat be realized?</i> .....	9
1.3.2 <i>External threats to the system</i> .....	10
1.3.3 <i>Who has access to what assets, and why?</i> .....	11
1.3.4 <i>Restrictions on the threat model</i> .....	12
1.3.5 <i>Total Leakage of the System</i> .....	12
1.3.6 <i>Relaxation of the security of the system for increased functionality of the operational model</i> ....	12
1.3.7 <i>Security Policy</i> .....	13
1.3.8 <i>Use cases</i> .....	13
1.3.9 <i>Fewer parties = fewer links in the chain</i> .....	13
1.3.10 <i>Trusted vs. Untrusted environments</i> .....	15
1.4 THREAT MODEL: PARTIES.....	16
1.5 THREAT MODEL: ASSETS .....	16
1.5.1 <i>Central security objective</i> .....	16
1.5.2 <i>Assets in the trusted system</i> .....	16
1.5.3 <i>Assets in the untrusted system</i> .....	17
1.6 OPERATIONAL MODEL.....	18
1.6.1 <i>Functionalities supported by the system</i> .....	18
1.6.2 <i>Allowed queries</i> .....	18
1.6.2.1 <i>Substring Search</i> .....	18
1.6.2.2 <i>Percent Match</i> .....	18
1.6.2.3 <i>Homolog Search</i> .....	19
1.6.3 <i>Disallowed queries</i> .....	19
1.6.4 <i>Access control is managed via symmetric encryption</i> .....	19
1.6.5 <i>Key management</i> .....	19
1.6.6 <i>Information flow sequence</i> .....	20
1.6.7 <i>Security violation definition</i> .....	20
1.6.8 <i>Internal database representation</i> .....	20
<b>2 PRIVACY .....</b>	<b>22</b>
2.1 PERSONAL PRIVACY .....	22
2.1.1 <i>Keeping data private from other individuals</i> .....	25
2.1.2 <i>Limitations of the system</i> .....	26
2.1.3 <i>Challenges in restricting data usage to one party</i> .....	26
2.2 THE NEED FOR PRIVACY DESPITE EXISTING LEGAL FRAMEWORKS .....	26
<b>3 RELATED WORK .....</b>	<b>27</b>
3.1 BACKGROUND ON HOMOMORPHIC ENCRYPTION .....	27
3.1.1 <i>Homomorphic encryption in insecure environments</i> .....	27
3.1.2 <i>Using Fully Homomorphic Encryption to ensure privacy: technical solutions to address criticisms</i> 28	
3.1.2.1 <i>Modern cryptographic schemes should be post-quantum secure</i> .....	28

3.1.2.1	Systems leveraging homomorphic cryptographic security must speed up encryption and operation runtime	28
3.1.2.1	Parallelization without synchronization allows for maximal sub-threading	29
3.1.2.1	Cryptographic systems should support memory-constrained environments	31
3.2	BACKGROUND ON EXISTING SECURITY SYSTEMS FOR DNA	31
3.2.1	<i>Security in DNA databases should be inherent within the database, and not guaranteed by external factors</i>	31
3.2.2	<i>Background on advances in private information retrieval</i>	32
3.3	ENCRYPTED GENOMIC SEARCH RELATED WORK, AND HOW FLEXFHE DIFFERS	33
<b>4</b>	<b>SYSTEM ARCHITECTURE</b>	<b>35</b>
4.1	ADDRESSING THE OPERATIONAL MODEL IN THE CONTEXT OF THE THREAT MODEL	35
4.1.1	<i>Trusted environment</i>	35
4.1.2	<i>Untrusted environment</i>	36
4.1.3	<i>Untrusted environment: analysis of components</i>	36
4.1.3.1	Re-encryption to querier's key: tradeoff between engineering efficiency and query confidentiality	36
4.1.3.1.1	<b>Re-encryption strategies fail in the untrusted environment</b>	36
4.1.3.1.2	<b>Why not utilize a PKE scheme with built in re-encryption</b>	37
4.1.3.1.3	<b>Implemented solution: Alice performs proxy re-encryption</b>	38
4.1.3.2	Running functional operations on the database	39
4.1.4	<i>Key Management</i>	40
<b>5</b>	<b>DATA FLOW</b>	<b>41</b>
5.1	TRUSTED ENVIRONMENT	41
5.1.1	<i>Data Flow Overview in the Trusted Environment</i>	41
5.1.2	<i>Binary Encoding Discussion</i>	42
5.1.3	<i>Step By Step functionality in the Trusted Environment</i>	45
5.1.3.1	Driver Code	45
5.1.3.2	Read From File	45
5.1.3.3	One Hot Encode	45
5.1.3.4	Serialization	46
5.1.3.5	Encryption	46
5.2	UNTRUSTED ENVIRONMENT	47
5.2.1	<i>Data Flow Overview in the Untrusted Environment</i>	47
5.2.2	<i>Step by Step functionality in the Untrusted Environment</i>	48
5.2.2.1	Driver Code	48
5.2.2.2	Encrypted Substring Search	48
5.2.2.2.1	<b>Encrypted T/F is pattern found</b>	49
5.2.2.3	Percent Match	50
5.2.2.3.1	<b>Raw Match</b>	50
5.2.2.1	Homolog Search	51
5.2.2.1.1	<b>Encrypted Substring Search for Homolog</b>	51
5.2.2.1.2	<b>Encrypted Percent Match for Homolog</b>	52
5.3	RE-ENCRYPTION PROTOCOL	52
5.4	DECRYPTION OPERATIONS	52
5.4.1	<i>Decryption for Substring Search</i>	52
5.4.2	<i>Decryption for Percent Match</i>	52
5.4.3	<i>Decryption for homolog search</i>	52
<b>6</b>	<b>UNDERLYING ALGORITHMS</b>	<b>54</b>
6.1	OPENFHE	54
6.1.1	<i>BinaryFHE and LWE (Learning With Errors)</i>	54
6.1.2	<i>Invoked CryptoContext functions</i>	56
6.1.3	<i>Chosen Security Parameters</i>	56
6.1.4	<i>Private, refreshing, and re-encryption keys</i>	57
6.2	PROXY RE-ENCRYPTION AND KEY SWITCHING	57
6.2.1	<i>Noise reduction via bootstrapping</i>	59
<b>7</b>	<b>RESULTS AND DISCUSSION</b>	<b>61</b>

<b>8</b>	<b>FUTURE WORK .....</b>	<b>64</b>
8.1	IMPLEMENT IND-CCA1 LWE SCHEME .....	64
8.2	IMPLEMENT GPU THREADING .....	64
8.1	MODIFICATION FOR RNA VARIANTS .....	64
<b>9</b>	<b>APPENDIX.....</b>	<b>65</b>
A	USER MANUAL: REPRODUCE PERFORMANCE AND TIMING RESULTS .....	65
A.1	GETTING STARTED.....	65
A.2	EXECUTING TIMING AND PERFORMANCE DATA ONLY .....	67
A.3	USAGE STEPS OVERVIEW .....	67
A.3	STRUCTURING YOUR GENOME FILES .....	68
B	USER MANUAL: EXECUTE FLEXFHE AS A USER, SECURELY.....	69
<b>10</b>	<b>REFERENCES .....</b>	<b>70</b>
<b>11</b>	<b>ACKNOWLEDGEMENTS.....</b>	<b>77</b>

# 1 Introduction

In the past decade, DNA sequencing technology has dramatically advanced, allowing individuals to get their entire DNA sequenced for a manageable cost [69]. While sequencing technology has advanced dramatically, the storage and management of DNA data has remained largely the same. This means that even though individuals can get their entire DNA sequence, a feat, this sensitive data is stored in the same way as all general health data. Healthcare and research analysis is done on plaintext DNA, and the data is protected only by the trust in those handling DNA.

## 1.1 Introduction to DNA privacy concerns

### 1.1.1 Genomic data has expanded with better sequencing technology

The sequencing of the human genome has rapidly improved over the past decade. In 2003, the first human genome was sequenced. The project was funded nationally by several organizations, most notably the National Human Genome Research Institute under the National Institute of Health under the Human Genome Project. The first genome took about 13 years to sequence and cost an estimated 3 billion dollars at project completion [89]. In 2008, the cost of sequencing one individual's genome was reduced to about 1 million dollars via utilization of massively parallel DNA sequencing and took about two months to complete [103]. In 2009, technology to sequence a genome was estimated to cost \$100,000 [89]. Today, development of technology to sequence the entire 6 gigabyte human genome for under \$1000 is underway, a project funded by National Human Genome Research Institute [15].

The impact of full genome sequencing cannot be understated. The COVID-19 pandemic brought with it the ability to efficiently mass produce mRNA vaccines at low cost. While the COVID-19 mRNA vaccines wrapped mRNA into a lipid-nanoparticle shell [85], companies like Moderna and Pfizer have demonstrated that the same delivery method can carry DNA fragments, opening the door for treatments that target DNA directly.

Databases containing DNA are being used for more functionality than ever before [25] [28]. In their 2009 paper for the International Journal of Law, Crime, and Justice, Dahl and Sætnan discuss the concept of "function creep" of DNA databases, which describes the increasing utilization of DNA databases, beyond their intended original use [28]. Some examples of function creep for DNA database, they describe, include the usage of databases containing either complete or partial patient DNA sequences, originally created for healthcare purposes, for forensic purposes such as identifying family members and identifying suspects in criminal cases.

With ever increasing utilization of DNA data in modern healthcare, and the utilization of DNA databases for functionality beyond their intended healthcare purposes, it is ever more important to have secure systems in place to protect the privacy, integrity, and confidentiality of patient DNA data [77].

### 1.1.2 A need for anonymizing data and controlling partial database access

Healthcare data is some of the most private, personal, and regulated forms of information. There are a myriad of laws and regulations to protect this information from being used by bad actors.

When a research lab requests access to medical records, the systems in place to prevent the misuse of the data are policy based. Researchers follow laws and regulations such as the Health Insurance Portability and Accountability Act, standards set by the Institutional Review Board, and the policies of their direct institutions regarding how they access, store, and use sensitive healthcare data. For example, for healthcare providers working with electronic patient medical records, common requirements include using an encryption service to encrypt the hard drive of the machine where data is stored on, restrictions on storing data in cloud-based services such as Google Docs and AWS, and restrictions on the individuals that can access the data [58] .

In research labs, there is often a hierarchy of roles. For example, in a research study involving electronic medical records, the Primary Investigator may need access to a wider set of data than the research assistants using the data. Some people involved in the research may not even need access to the entire set of data the lab has access to complete their tasks. For example, in a lab using untrained volunteers to codify patient symptoms, the set of the data containing the patient symptoms often contains other personally identifiable information including names, phone numbers, identification numbers, addresses, and other data that although included in the database, is irrelevant for the current task at hand [12] .

For labs that request a wide set of patient data, there is often no effective mechanisms for removing irrelevant personally identifiable health data from the shared databases amongst their team[12] . Many labs store this data in simple Excel sheets, and anonymizing the information requires a manual modification of the entire database to physically remove data that should not be shared with others.

However, this approach has many risks. First, the primary investigator could fail to remove irrelevant personally identifiable information or could inadvertently modify the database in doing the manual operation. More pressingly however, this approach does not allow for a simple way of hiding information in the database for some team members, but not all. Team members that do not need all the information in the database to complete their analysis still receive the entire set of data to look at. This inappropriately wide access puts sensitive patient data in the hands of unseasoned and often poorly vetted researchers who may use the data inappropriately, and who may not follow stringent policy requirements for use of the data, such as, for example, putting the data into Google Docs, talking about personally identifiable health data with others, or even posting inappropriate details of their work on social media.

For researchers using an Excel-based data system, an obvious solution is to simply obfuscate and deobfuscate patient data as needed [12] . A simple solution is to encrypt the data sections that should be hidden, and simply unencrypt when access to the full database is needed. However, medical researchers often do not have the software background to parse, encrypt, and decrypt specific sections via a program such as a python script.

Today, there are many APIs that provide out of the box solutions for hiding partial Excel data via encryption and obfuscation methodologies, such as PyCryptodome [60] , Google's Fully Homomorphic Encryption Library [92] , Microsoft's SEAL library [67] , and many others [104] . Using these APIs is unfeasible for professionals outside the software field and hiring a software engineer onto a team to make custom tools is cost-prohibitive and introduces an additional party poorly versed in regulations meant to secure sensitive health data.

### **1.1.3 Genomic data is badly protected by existing regulations**

Existing regulations on healthcare data set by the IRB, HIPAA laws, and institutional-specific protocols rely on trust of human actors in the system.

In organizations that have such regulations in place, a non-malicious actor that fails to follow protocols threatens the privacy of genomic data to which they have access.

DNA data is unique from general healthcare data [77] . In a post published by NIST, genomic data is outlined as unique from other healthcare data, and thus requiring additional regulatory and cybersecurity-based protections, due to several attributes unique to DNA [77] . First, genomic data is immutable; once collected, DNA data does not change over the lifetime of the patient. Additionally, NIST recognizes the uniquely impactful role of genomic data in personalized healthcare treatments; the risk of genomic data being used to discriminate against patients or their family members; the risk of genomic data being used to impact financial aspects of patients' life such as insurance costs; and the risk of biological weapon creation [77] .

The United States' Genomic Privacy Act was created to apply additional protections to DNA data, regulating the usage of the data more stringently than other healthcare data [1] . The act "protects individual privacy while permitting medical uses of genetic analysis, legitimate research in genetics, and genetic analysis for identification purposes"[3] . Additionally, the United States' Genetic Information Nondiscrimination Act of 2008, or "GINA", was created to protect against discrimination of individuals by insurance companies based on sequenced genetic information [39] . Split into two sections, or Titles, "Title I of GINA prohibits discrimination based on genetic information in health coverage. Title II of GINA prohibits discrimination based on genetic information in employment." [39]

However, even databases that follow HIPAA regulations are susceptible to de-identification attacks using cross references from publicly available databases [58] [97] . In a 2013 study by the Harvard School of Engineering and Applied Sciences, Sweeny et al showed that it was possible to identify the complete name of all participants in an anonymized genetic database using cross references to other publicly available databases [97] .

Finally, bad actors can maliciously access healthcare databases directly, an inappropriately view private health data, including specific databases containing genomic data linked to personally identifiable information such as names, addresses, and identification numbers [90]

### **1.1.4 Summary of goals**

The goals of this project are therefore to: create a system, usable by professionals without software development experience, that provides partial encryption and decryption of a database, provides homomorphic encryption of the data in the database, exports the encrypted database, is able to decrypt an encrypted database, allows for relevant operations on the encrypted data in an untrusted environment, and allows for decryption of the final data after relevant transformations have been applied.

## **1.2 System overview and use cases**



### **1.2.1 Use case: trusted and untrusted environments**

The trusted environment exists offline; the untrusted environment exists online.

The central use case of this system is that a trusted party has access to a DNA database and needs to give an untrusted or partially trusted party access to the DNA data. The goal of the trusted party is to restrict access to the data for untrusted parties.

Some examples of this use case include having a locally secure DNA database in a clinical setting but wanting to use cloud computing technology to operate on the data. In this case, the clinical setting represents the trusted environment, and the cloud setting represents the untrusted environment.

Another use case includes a trusted party having access to a DNA database in a clinical setting but needing to give untrusted or partially trusted parties access to some, but not all, of the data. For example, a manager of a database might want to allow employees to query the database but wants to strongly control the ability of the corruptible parties to view the data. In this scenario, the corruptible parties need to be able to make legitimate queries on the data for research purposes, but they should not be able to access more data than they are authorized for via their queries.

### **1.2.2 Use case: real world examples**

Some real-world examples of the use cases described in section 1.2.2 include police needing to search a DNA database for forensic purposes. The clinical setting should cooperate with the police but should assure that the police only access the data they need to access within the database to protect confidentiality of patients in the database.

Another example of these use cases includes the case of corruptible insiders. To protect against corruptible insiders, we wish to restrict access to the database to the minimal required data subset.

## **1.3 Threat Model**

The central asset in the system is the privacy of individuals to whom the genomes in the dataset belong. Alice's challenge is to allow Bob to query the dataset while protecting the privacy of these genomes.

The parties in the system are database owner Alice and querier Bob. The other assets are Alice's DNA database and Alice's secret key used to encrypt the dataset. The primary threat is Bob running analysis on Alice's DNA beyond what she allows. The secondary threat is Bob gaining access to Alice's secret key and decrypting the encrypted dataset.

### **1.3.1 What are is being protecting from whom, and how can the threat be realized?**

The threat in this system is unauthorized access to the dataset and unauthorized querying of the dataset. The threat is realized through the vulnerabilities of the system, which is the communication between the querier (Bob) to the database, and the vulnerable storage of the database. The assets in system are the genomic dataset and metadata about the dataset.

Honest Alice wants to protect against unauthorized access to her genomic database from an honest but curious Bob, a compromised Bob, or a malicious Bob.

In all cases, Bob needs to make some queries on the database. However, Alice only wants Bob to make queries on the database that she allows.

An honest querier will only query the dataset with authorized queries and will seek to glean the minimum possible information he requires from the dataset. For example, an honest querier who is authorized only to look up the incidence of gene X in a population will only attempt to execute that query, and only gain that information from the dataset.

The threat of unauthorized access to the database can be realized in several different ways. First, the querier could directly access the dataset, thus gaining complete access to the dataset beyond his authorized level. Second, the querier could make unauthorized queries to the database, and gain more information than allowed.

The threat of unauthorized access can be realized through corruption of the querier. In the case the querier is honest but has complete access to the dataset, the querier may be hacked, bribed, coerced, or otherwise corrupted; in this case, the querier would leak the entire dataset to a malicious adversary.

An honest but curious querier threatens the system by accessing information beyond his authorized level, which violates the privacy of the individuals to whom the genomes in the database belong. The people who have their genomic data in Alice's database consent to a specific set of querying to be performed. While honest but curious Bob may only be trying to get additional data from the dataset, he is violating the privacy of the individuals in the dataset which Alice has promised to protect.

A malicious querier threatens the system by attempting to break the core security of the system, which relies on the security of Alice's secret key.

### **1.3.2 External threats to the system**

External threats to the system exist beyond Alice and Bob. In the system, external threats are malicious third-party actors that attempt to gain access to some or all of the plaintext genome dataset.

Motivations for a third-party attacker to violate the privacy of the system range from curiosity to monetary gain. If this system is implemented into consumer DNA services (such as 23AndMe) an attacker would be motivated to breach the system in order to run their unauthorized queries on the individuals in the dataset, find a familial association between a police-held genome and the genomes in the dataset (as in the Golden State Killer case); identify genes in an inappositely acquired genome for persecution, monetary gain, or curiosity (for example, a fan purchasing Madonna's personal assets containing her DNA or police swabbing discarded items for prosecution).

This threat can be realized in several different ways.

A third-party attacker can attempt to gain access to the plaintext dataset by impersonating, coercing, or otherwise corrupting Alice and the Trusted Environment to expose the plaintext

dataset. In this scenario, there is no cryptographic defense to protect Alice. Instead, external authentication systems (such as multi-factor authentication or usage of an HSM) are expected to be used. The attacker can also attempt to gain access to Alice's secret key to decrypt the encrypted dataset in the Untrusted Environment. Thus, we expect Alice to store her secret key using an HSM.

A third-party attacker can also attempt to gain access to the parts of the dataset by corrupting Bob. If Bob is corrupted, the third-party gets access to the data that Bob has, which is the encrypted dataset, and the result of the query. In our threat model, we assume that Bob is honest but curious and therefore a malicious third-party impersonating Bob would corrupt the privacy of the system. We assume that common computer-security authorization defenses have been placed to prevent a corrupted Bob or a corrupted Untrusted Environment from launching a CCA1 or CCA2 attack to recover Alice's secret key. Such defenses include authorization level assignment and management during login to the system or login with client-side certificates issued by a trusted certificate authority, both of which can occur in an offline setting to decrease vulnerability to hacking.

We expect the system described in this thesis to be run within a reasonably secure infrastructure to minimize these failure points.

We also expect Alice and Bob to communicate through secure channels, using technology such as TLS to create a channel that preserves the integrity, confidentiality, integrity, and availability of Bob's queries that Bob sends to Alice, and the encrypted genome that Alice sends to Bob along with, if Bob is honest but curious, the re-encrypted query. It should be noted that while Alice has a symmetric secret key, Bob has a public/private key pair generated by PKE.

### **1.3.3 Who has access to what assets, and why?**

Alice has access to the database containing genomes, her secret key, Bob's public key, and the patterns (nucleotide sequences) that Bob would like to use as queries on the database. Alice uploads all this information to the Trusted Environment.

The trusted environment must access Bob's public key to re-encrypt the ciphertext query result such that Bob can decrypt it with his secret key.

Bob has access to the homomorphically encrypted genome, which is encrypted with Alice's secret key. If Bob is honest but curious, he also has access to the re-encrypted result of his query from the Untrusted System.

If Bob is malicious or corrupted, he gets the plaintext result of the query from the Untrusted System.

In other words, Bob is more trusted (honest but curious, not malicious), the Untrusted Environment will not see the plaintext result of the query. If Bob is less trusted (malicious or corrupted) the Untrusted System will decrypt the result of the query into plaintext to export it to Bob. The trust levels between Bob and the Untrusted Environment are inversely correlated due to the nature of homomorphic encryption being only IND-CPA, or semantically, secure.

### **1.3.4 Restrictions on the threat model**

In the system, we assume Alice is honest and incorruptible, and that Bob cannot coerce, bribe, or otherwise corrupt Alice into giving up the plaintext genomic database or her secret key used to encrypt the database.

We assume the Trusted Environment is incorruptible. This can be realized in the real world via the use of a Hardware Secure Module (HSM) to store Alice's secret key.

We assume that the Untrusted Environment will not launch a regular or adaptive chosen ciphertext attack on the encrypted database. Most modern ciphers are immune to such attacks and exhibit IND-CCA2 security. However, by definition of IND-CCA2, these ciphers do not exhibit homomorphism, and thus systems that utilize such encryptions require the decryption of data before querying the system. In the system implemented in this thesis, we relax the requirement CCA2 security down to CPA security, the maximum security level guaranteed by FHE schemes. Although we downgrade the security level of system, we relieve the vulnerability of having data in plaintext in an untrusted environment.

If Bob is honest but curious, we assume that Bob will not launch a chosen ciphertext attack to recover Alice's secret key.

If Bob is malicious or corrupted, we assume that Bob will not influence the Untrusted Environment to launch a CCA1 or CCA2 attack.

### **1.3.5 Total Leakage of the System**

If Alice is corrupted, or if the Untrusted System launches a chosen ciphertext attack, the security of the system is broken. Alice has populated the dataset, and thus no cryptographic primitive will prevent her from knowing the data in the dataset. The Untrusted System homomorphically operates on the encrypted database, operations which are only secure against chosen plaintext attacks; or in other words, protect against the semantic security of the system.

The most secure system that protects the privacy of the database completely is one in which Bob does not query the dataset. In this thesis, we address the case in which the querier needs some access to the dataset to reflect the real-world use cases of genomic data; thus, we relax some security guarantees.

### **1.3.6 Relaxation of the security of the system for increased functionality of the operational model**

Because the dataset is symmetrically encrypted, the security of the system depends on the security of Alice's secret key. Because the dataset is encrypted using FHE, only IND-CPA security is guaranteed, under the assumption that Learning With Errors (LWE) [100] is secure.

This system was created to allow restricted access to a person's genome. Therefore, part of the operational model requires that some information about the genome will be leaked during normal function; namely, the result of the query must be decrypted at some point.

The system does not attempt to, and cannot, restrict all access to the private genome; instead, this system minimizes the amount of information exposed to a querier, or in other words, reveals the minimum set of information required to answer a query.

In short, the system lets you ask questions without revealing the source.

It allows for only consented queries; for example, letting researchers identify the incidence of diabetes as consented to by the population, but not of schizophrenia (as in the case of ASU researchers violating the privacy of the Havasupai population's genomes).

### **1.3.7 Security Policy**

The security policy is twofold. First, only authorized users shall run authorized queries on the database. Second, only the owner of the database shall be able to see the plaintext values in the dataset.

As in traditional large-scale security systems, a separate certificate authority shall authenticate Bob before letting him access the system.

The underlying mathematical properties of homomorphic encryption will allow only authorized queries to be run on the database and will only allow the owner of the database to access the plaintext values of the dataset.

### **1.3.8 Use cases**

One example of a use case in this model is as follows: Alice is the principal investigator of a research lab. She has an employee, Bob. Bob is not malicious, but he is badly educated on rules as to how to handle sensitive data. Alice wants to allow Bob to perform some analysis on her DNA database—specifically, she wants Bob to identify the number of individuals in the database with gene X. She wants Bob to only perform the specific query that she allows for; in other words, Bob should not be able to glean any more information about the genome such as reconstruction of genome or search for additional genes.

Another use case that fits this model: Alice is the PI in the above example wanting Bob to identify how many individuals in her database have gene X. One day, Bob is hacked, and an unauthorized attacker tries to read the genomes that Alice sent Bob.

Another use case: Alice writes a research article discussing the prevalence of gene X in a population's DNA, and Bob is a malicious reviewer at the journal. Bob must verify Alice's claims about the prevalence of the gene in her population, but Bob also wants to glean additional information about the population (for example, if they have gene Y). The system is created so that Bob can only run the analysis that Alice allows.

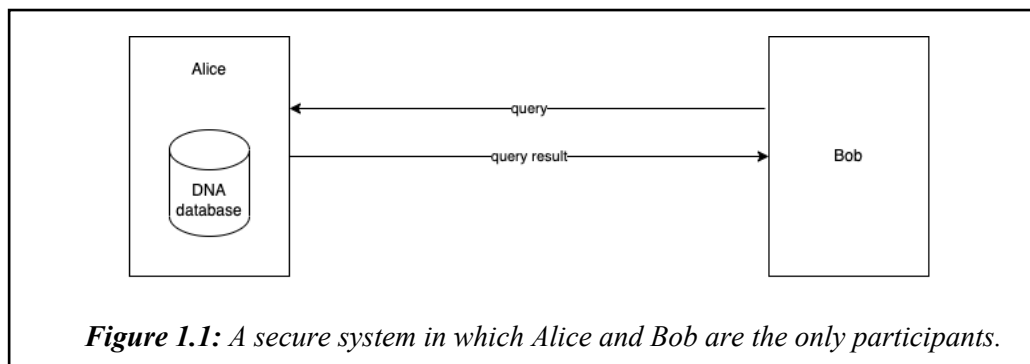
### **1.3.9 Fewer parties = fewer links in the chain**

Alice wants to allow Bob to run allowed queries directly on her database, without allowing for a compromised impersonator to see the plaintext values of the dataset.

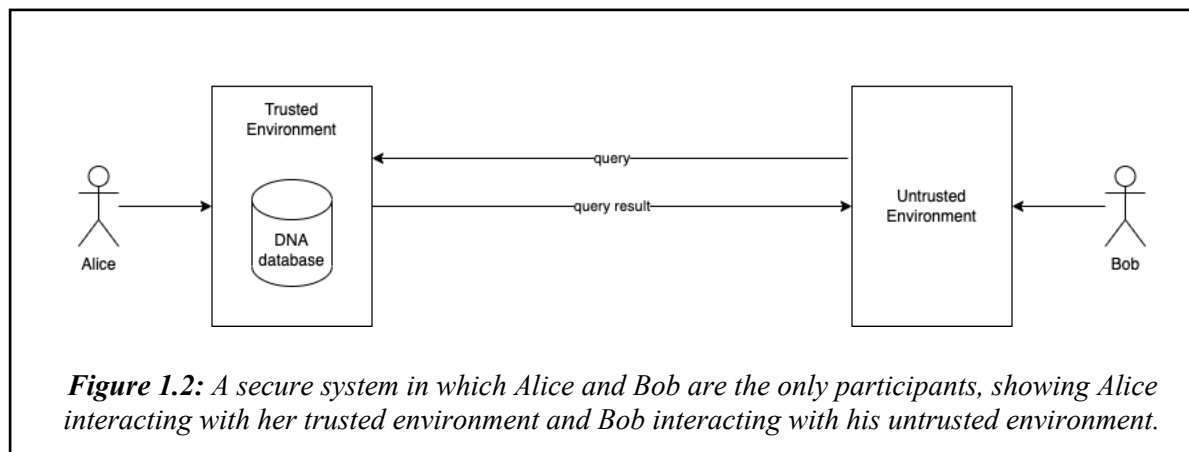
Traditionally, this use case is managed by a third-party server that controls Bob’s access to the database. The authentication server either contains direct access to the database or sends authorized query.

The Authentication Server not only injects additional trust requirements into the system, it becomes the weak link—if the authentication server falls, the DNA database is exposed. While Bob can only be compromised directly, the authentication server can be compromised both directly through a hack and indirectly by a compromised Bob.

This can be visualized as follows:

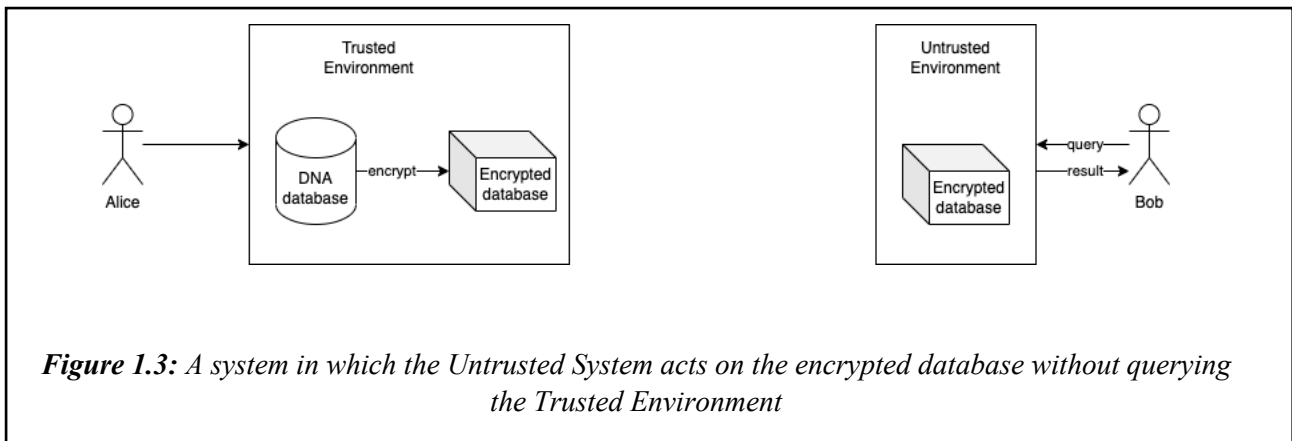


In reality, Alice does not communicate with Bob directly. Alice interacts with her computer (which we shall call the Trusted Environment). Bob interacts with his computer (which we shall call the Untrusted Environment). Alice and Bob communicate through these environments. This can be visualized as follows:



In this model, there are two parties, and only one can be corrupted. However, communication still exists between the Trusted Environment and the Untrusted environment. If Bob is corrupted, he can maliciously communicate with the Trusted Environment.

Therefore, we desire to further separate the communication between Alice and Bob. The figure below shows the final system, in which Alice’s trusted environment is separated from Bob’s untrusted environment.



**Figure 1.3:** A system in which the Untrusted System acts on the encrypted database without querying the Trusted Environment

### 1.3.10 Trusted vs. Untrusted environments

The system consists of a trusted environment and an untrusted environment. The trusted environment has access to the database and performs the initial homomorphic encryption of the entire database. This operation occurs with a single master encryption and decryption key and is stored within the trusted system. Section 4 describes this system in detail.

The trusted environment generates a new master decryption key each time a new database is encrypted. This prevents misuse of leaked keys affecting many databases.

The trusted environment does not store any database information, neither in encrypted nor plaintext format, other than the master encrypt and decrypt key. Instead, it only exports the homomorphically encrypted database out. The trusted system is a minimal module meant to perform the initial encryption of the database. Additionally, the system is designed such that it supports incremental updates to the database.

Additionally, the trusted environment exports a re-encryption key for each query made, which defined the range of data a query is authorized to access. This re-encryption key is linked to a query, a querier's ID, and is symmetrically encrypted and digitally signed.

The untrusted environment interacts with the homomorphically encrypted database and the query.

In the untrusted environment, the query is corruptible. In other words, the query can be maliciously modified to attempt to access a greater range of data than is authorized. The purpose of the re-encryption key is to identify when a query is attempting to access a greater amount of data than is appropriate.

The untrusted environment contains the system to access the database and perform logical operations on the database. Although this system is in a corruptible location, unauthorized access is prevented in several ways. First, the data in the database is encrypted. Second, the relationship between the queries and the portioned key are validated, assuring that no queries try to badly access the encrypted data.

The untrusted environment outputs a single data value for each query in encrypted format. The key to decrypt this result lays in the trusted environment only.

## **1.4 Threat model: parties**

The threat model consists of a database, database operator, querier, and queries. Access control is built into the system via the use of homomorphic encryption.

The Database contains sensitive data; authorized parties should only perform authorized actions on the database.

The database operator is honest but curious. The database operator converts the queries made from the querier to actions on the database and can see the queries they receive.

The database operator is the entity that interacts most closely with the database. The database operator interacts with encrypted data only. Even though the database operator is corruptible, the data the operator cannot see the plaintext data it interacts with. This assures that in the case of an adversarial operator, the data in the database remains secret.

The querier can be corrupted.

The querier makes requests to get certain information from the database. To constrain the scope of the system, we assume that the queries made by the querier cannot be modified after they are made. The querier sometimes is honest, but sometimes can be a corrupted entity trying to access the database beyond their authorized access level. Therefore, queries should be checked for legitimacy within the system.

The encryption of the database, and the created functional interface into the database, will only allow authorized parties to decrypt the results of queries on the database.

## **1.5 Threat Model: assets**

A threat is a combination of a vulnerability and a motivated adversary [9]. In the threat model, we will identify who the adversary is and the capabilities they are assumed to have.

The following section defines the threat model of the system with the purpose of narrowing the scope of this project to focus on specific cybersecurity applications [65].

### **1.5.1 Central security objective**

The central security objective is to prevent individuals from gaining unauthorized access to any subset of data in a database containing DNA data.

### **1.5.2 Assets in the trusted system**

The trusted system contains two central assets: the master key and the plaintext dataset. Secondary assets include the querier's public key and the generated re-encryption key, and the generated public/private key pair.



In this thesis, the goal is to protect both the master key and the plaintext dataset from leaking into a malicious party.

We constrain the definition of a malicious party to be an insider threat or an eavesdropper in a cloud-based environment. The system is not built to protect against parties with very high computing power and nearly unlimited technical resources, such as threats posed by nation states.

Some examples of real world threats to genomic datasets are: researchers in clinical settings with access to the dataset that do not abide cryptographic and security regulations to protect patients' data; insider threats selling or sharing patient data; bad-actor researchers with access to the dataset that share the dataset with other laboratories, performing research beyond the patients' agreed scope, especially research that is culturally and religiously sensitive such as in the case of the usage of indigenous people's genomic data in ways that violated their religious and cultural beliefs [43] ; and cloud based providers that have a security breach of their own, leaking data of their customers, leading to financial penalties on entities holding regulated health data.

If a master key is corrupted, the database may be completely unencrypted to exist in plaintext only. Therefore, the highest priority asset to protect is the master key.

The second asset, the plaintext dataset, is equally as important. The plaintext dataset contains sensitive genomic data and should not be seen by entities without authorization.

Finally, the trusted system contains the protocol to homomorphically encrypt the dataset. This asset is open source and does not need to be protected—knowledge of the encryption protocol does not leak information about the data to be encrypted.

### **1.5.3 Assets in the untrusted system**

The untrusted system contains as assets: the homomorphically encrypted dataset, the homomorphically encrypted result of the query, and the query protocols.

The first asset is the homomorphically encrypted dataset. The encrypted dataset is unprotected as it does not leak meaningful information about the private genomic dataset. The second asset is the query which exists in ciphertext form. In this scenario, the query that has been encrypted is not protected but certain security requirements are defined. The ciphertext query should not be forged, and the query should be bound to the querier.

This system does not protect the query, but instead protects the integrity and non-repudiation of the query against untrusted environments and insider threats. Finally, the system protects the access to the dataset. Each query must only access the minimal subset of the dataset required to answer the query.

These assets are all protected against an insider threat wishing to analyze the dataset beyond what is necessary for them to complete their task. For example, a research assistant with access to patient DNA data should not be able to access portions of the genome beyond what is needed for their analysis. In short, these informational assets should be protected against the curiosity of an individual without true need to access the data.

## **1.6 Operational model**

The following section defines the operational model of the system with the purpose of defining the supported functionalities of the system, and the allowed and disallowed functions that build up the definition of a breach of security [65] .

The purpose of the operational model is to constrain the querier to access the minimum required data they need to perform their analysis.

### **1.6.1 Functionalities supported by the system**

Using the system, the following functionalities are supported on the database, where 'loci' refers to a specific location in the genome.

For one individual, run analysis on DNA via direct DNA access. This includes looking up specific loci, looking up loci range, and searching for an allele in the DNA. It is also possible to access personally identifiable information in the database, phenotypic information, and other data held in the database.

Using the system, it is possible to run a comparative analysis on many individuals in a database and run statistical operations. It is possible to access the DNA strings of many individuals, a specific locus, or a loci range for uses of comparative and statical analysis.

By using the query operations exposed to the user, the system can answer questions such as: does an individual have a gene; what the frequency of a gene in the system is or in a subset of the individuals in the system; what genetic commonalities between individuals or between a demographic of individuals are; and what mutations are present in a gene for an individual.

### **1.6.2 Allowed queries**

#### **1.6.2.1 Substring Search**

Substring Search does a pattern matching operation to identify if a substring is located in a string. In practice, Substring Search can be used to identify if an individual has a gene, and at which locus the gene is located at.

#### **1.6.2.2 Percent Match**

Percent Match returns the percentage of matching nucleotides between two genomes, or between a genome and a sequence of nucleotides (a gene or a gene with a mutation). When two genomes are known, Percent Match can be used to identify familial relationships between two individuals. For example, 23 and Me detects related individuals based on the percent of DNA shared between two individuals, and the amount of contiguous matching segments that are at least 700 SNPs long.

### **1.6.2.3 Homolog Search**

Homolog Search allows for searching for mutation through the use of wildcards. The input to the function is a sequence of nucleotides with wildcards, and the function returns the sequence in the genome. For example, given genome “aaaaggcgaaaagtgcg” and wildcard pattern “agXcg”, homolog search returns two matching sequences: “aggcg” and “agtgcg”.

### **1.6.3 Disallowed queries**

The querier shall not be able to modify the database via a read or write operation, or perform a query beyond which has been authorized by the database owner.

Given data that resides in the database, the querier shall not be able to request additional attributes located in either the row or column of that data. In other words, the querier shall not access additional data on the individual, even if they have one or more data on the individual. For example, given a phenotype, the querier shall not be able to query the DNA string, and given the DNA string or substring, shall not be able to query other attributes such as ID, name, or phenotype.

For all queries, given a value in a row in the database, the query shall not return the entire row, or another value in the row. Similarly, given a value in a column, the query shall not return related row values.

### **1.6.4 Access control is managed via symmetric encryption**

Because the genomic dataset is symmetrically encrypted, queries must operate on patterns that are also symmetrically encrypted. In the system, Bob must send his patterns to Alice who will use the trusted system to symmetrically encrypt the patterns.

This gives Alice total control on the subsequences that are used to query a pattern-match request on the genome.

In the real-world example, this would correspond to a participant in a research study encrypting nucleotide sequences that the researcher has requested to study. Because the owner of the dataset must encrypt queries on the dataset symmetrically, the owner of the dataset controls which nucleotide sequences may be used to pattern match on the genome.

Access control is approved through the owner of the dataset and enforced through homomorphic encryption; by definition of symmetric homomorphism, it is not possible to query the dataset with strings that have not been encrypted by the dataset owner.

### **1.6.5 Key management**

Given a secure key to the database and a querier’s public key, a new key is generated called a “re-encryption key” that is used to re-encrypt the ciphertext result of the dataset so that the querier can decrypt it with their private key.

In the system, re-encryption occurs in the trusted environment. The re-encryption protocol decrypts the Learning With Errors ciphertext result from the untrusted environment and encrypts it with BFV, a scheme that supports PKE and therefore proxy re-encryption (more

details are in section 4.1.3.1.3. Thus, during re-encryption the plaintext result of the query exists in the trusted environment, which Alice has access to. In order to reduce the overall required trust in the system, this re-encryption protocol should be run as a separate service on a secure HSM so that Alice is not able to see the result of Bob’s query.

### 1.6.6 Information flow sequence

A querier (Bob) passes their desired patterns to query on to the database owner (Alice). Alice passes the patterns, her secret key, and her genomic dataset to the Trusted System.

The trusted system symmetrically encrypts the genomic dataset and the patterns to be used in querying with Alice’s secret key. The trusted system serializes the genome and the patterns into files and compresses the file set to download onto Alice’s machine.

Alice then gives the encrypted file set to Bob, who uploads it to the Untrusted System. The untrusted system listens to a query command from Bob, and then performs encrypted queries on the encrypted genomic dataset. Upon completion of the dataset, the untrusted system outputs the encrypted results.

Re-encryption of the result then occurs so that Bob can decrypt the query with his private key. Details on re-encryption are discussed in section 4.1.3.1

### 1.6.7 Security violation definition

A security violation is defined as an access violation in which a querier tries to access a greater subset of the database than they are allowed.

### 1.6.8 Internal database representation

The trusted system stored the genome sequence into a one-hot encoded representation internally. It stores the patterns to be compared against the genome in the same one-hot encoded representation.

Each item in the dataset must consist of only the characters *a t g c*.

Internally, each genome and pattern is converted into a 4 by <sequence length> array, where each row in the array is a one-hot encoded location of a (index 0), c (index 1), g (index 2), t (index 3).

Tables 1.1 and 1.2 below show the internal representation of data in plaintext form, where *n* is the length of the genome and *m* is the length of each pattern (the lengths of each pattern need not be the same).

**Table 1.1:** *Plaintext representation genomes in the system.*

Genome (nucleotide sequence)
$[[ \{0, 1\}^n ], [ \{0, 1\}^n ], [ \{0, 1\}^n ], [ \{0, 1\}^n ] ]$

**Table 1.2:** Plaintext representation of patterns in the system.

Patterns (nucleotide sequence)
$[[\{0, 1\}^m], [\{0, 1\}^m], [\{0, 1\}^m], [\{0, 1\}^m]]$
$[[\{0, 1\}^m], [\{0, 1\}^m], [\{0, 1\}^m], [\{0, 1\}^m]]$

In the untrusted system, all data exists in encrypted form. Tables 1.3 and 1.4 show this representation. In this system, *Enc* is the Learning With Errors binary encryption algorithm and *sk* is Alice's secret key.

**Table 1.3:** Encrypted representation of a genome in the system.

Genome (nucleotide sequence)
$[[\text{Enc}_{sk}(\{0, 1\}^n)], [\text{Enc}_{sk}(\{0, 1\}^n)], [\text{Enc}_{sk}(\{0, 1\}^n)], [\text{Enc}_{sk}(\{0, 1\}^n)]]$

**Table 1.2:** Encrypted representation of patterns in the system.

Patterns (nucleotide sequence)
$[[\text{Enc}_{sk}(\{0, 1\}^m)], [\text{Enc}_{sk}(\{0, 1\}^m)], [\text{Enc}_{sk}(\{0, 1\}^m)], [\text{Enc}_{sk}(\{0, 1\}^m)]]$
$[[\text{Enc}_{sk}(\{0, 1\}^m)], [\text{Enc}_{sk}(\{0, 1\}^m)], [\text{Enc}_{sk}(\{0, 1\}^m)], [\text{Enc}_{sk}(\{0, 1\}^m)]]$
...

## 2 Privacy

“It is outrageous and grossly offensive that my DNA could be auctioned for sale to the general public.” – Madonna

Information security is typically defined by three tenets: confidentiality, integrity, and privacy. In other words, cyber systems attempt to protect personal data by preventing unauthorized disclosure of information, preventing unauthorized modification of information, and preventing unauthorized withholding of information. This thesis focuses on building a system that secures the confidentiality of data.

In other words, the threat of disclosure of data is addressed by creating a system with minimized vulnerabilities to leakage attacks.

In this paper, the security policy is simple: unauthorized users shall not have access to any part of an individual’s genome located in the database. Authorization is defined by the owner of the data; in other words, only the owner of the data shall have access to the genomes of individuals in the database. Parties other than the original owner shall only be given the result of their queries on the database.

### 2.1 Personal Privacy

Privacy is a core tenant of security. When the privacy, or confidentiality, of data fails, this data can be misused. Misuse of an individual’s data can occur at the hands of other individuals and organizations.

Third parties often have legitimate reasons for needing to access some aspect of an individual’s personal data. However, rarely do third parties need to access the entire corpus of data belonging to an individual. For example, a third party may need to check the familial relationship of two individuals. To check relation, the party will have to identify percentage match between two genomes and identify contiguous stretches of genomes that match to a certain degree [51] .

Ideally, the party will only have access to the information it needs to answer its query. If a party needs to know the percentage of genome match, the length of contiguous matching genomes, or the percent match between areas of a genome, the party should only get responses to these questions and nothing more. The party should not be able to know other genes inside an individual’s genome, other mutations in an individual’s genome, or other relevant health data that can be extracted from a genome.

Why is it important to control the extent of the information a party can gain from a genome?

Genomic analysis is a powerful tool that can be used for “good” in many contexts. Within the general ethical rules of society, there are many appropriate contexts for genome analysis: emergency services may need to identify an individual in critical care; police may need to ascertain the relationship between a victim and an attacker; individuals may be curious as to their familial relationship to another individual; genes that indicate an increased risk to adverse phenotypes should be identified to allow for better preventative care.

However, indiscriminate access to an individual's genome easily leads to corruption of, at best, well-intentioned parties. Emergency services may pass genetic information to insurance companies, who may choose to illegally restrict coverage of an individual on the basis of their genotypic data. Police may use third party datasets to prosecute individuals based on information contained within these genome datasets. Third parties meant for genealogy exploration may release private health information contained an individual's genome for monetary gain. Well-intentioned healthcare practitioners with access to an individual's genome may pass the genome to parties that will use the data to perform research that the individual does not consent to.

While it may seem improbable that bad actors reside within these structures that are ingrained into society, real world examples of each of these abuses has occurred.

In 1989, Arizona State University approved a study to identify the incidence of diabetes within the Havasupai population, a Native American tribe native to and living in Arizona [87] [16]. Researchers John Martin and Teri Markow collected blood samples from tribal members and used genomic technology, which at the time was in its rudimentary stages, to assess the incidence of genomic markers for diabetes within the tribe.

In this complicated case, which included omission of informed consent from the participants, the researchers used the genomic samples to identify the incidence of other diseases that members of the tribe had not consented to, namely schizophrenia. Over several years, several other abuses of the data occurred, including the extraction and storage of cell lines from the blood cultures in 1991 without consent [87]. Furthermore, the genetic information was stored and shared with researchers at both the University of Arizona and the University of Chicago to public papers about "inbreeding", alcoholism, and origin of migration of the tribe from Asia [96]. Not only did the resultant research come from gross misuse of the tribe's genomic data, the research performed contracted the tribe's religious beliefs, specifically religious laws governing familial origin and links within the tribe [96] [48] [49]. In 2003 these abuses were discovered by the tribe, and they subsequently sued the Arizona Board of Regents in 2008 for invasion of personal, religious, and cultural privacy [49].

In recent years, police usage of genomic data has increased both with ease of collection of genomic information and with the increased number of commercial parties privy to an individual's genomic data according to the American Civil Liberties Union (ACLU) [94]. In 2015, police in Florida began implementing "Stop and Frisk", a program that allows police to collect genomic data "voluntarily" from individuals and minors, without any of the safeguards that protect personal privacy guaranteed by the 4<sup>th</sup> Amendment of the US Constitution [57], especially the requirement for police to have reasonable suspicion of a crime before requiring a search. Additionally, in the absence of a legally ordered warrant or subpoena, law enforcement may simply purchase genomic data from private organizations outright [27], which is outside the restrictions granted by the Fourth Amendment [27] [95].

A 2003 report by the ACLU outlined that despite discrimination laws currently in place, personal health data is routinely shared between healthcare providers and insurance providers. The report outlines the consequences for genomic data as discrimination by insurers, employment discrimination, and genetic spying, the improper analysis of high-profile genetic information such as that belonging to politicians or celebrities [95]. In a high profile 2018 case, Madonna famously lost her bid to stop the auction of personal items including those which contained her DNA, such as a hairbrush containing some of her hair

[56] . The case was tried in the state court in New York which ruled that since the items were stolen from her in 2004, the statute of limitations (three years in New York) prevented her from suing to get them back. In this case, the larger, and to genomic security researchers the more relevant, point regarding sale of genetic information was not ruled on by the court [50]

Since the 2018 case, the courts have upheld the sale of genomic data to law enforcement [44] , allowed civilians to identify the Golden State Killer by accessing publicly available genomic data and upheld a conviction based on this data [50] , allowed police to use genetic data collected from newborn genetic screening to search for suspects [44] , and upheld the ability to collect DNA at a private residence without a warrant to convict a suspect in South Dakota State v. Bentass [82] . Specifically, the court ruled that “Because defendant had no reasonable expectation of privacy in the items searched, the Fourth Amendment does not apply to the DNA testing performed on those items” [35] . In each of these cases, the defendants claimed that DNA data is too sensitive to be treated in the same manner as other items to which a person does not have an expectation of privacy and needs additional protection under the law. The courts did not agree.

The legal challenges of protecting DNA in law enforcement contexts cannot be understated. Should the Golden State killer be let free because the data came from a public database? Should Bentass, who was genetically linked to a baby found in a ditch [35] , escape conviction because her DNA was extracted without due process? Should doctors stop working with insurance because of the risks of discrimination?

Private companies like GEDMatch, a genealogical database, can and are used as tools for law enforcement [32] . In the Golden State Killer case, police uploaded a genetic profile generated from a crime-scene investigation to the GEDMatch website, and then used the tools on that website to identify the great-great-great-grandparents of the killer, and then reconstructed the family tree until they were able to identify the living relations of the individual [54] . Joseph James DeAngelo was arrested 44 years after he committed the initial crimes, including at least 12 murders, 50 rapes, and 100 burglaries, based on this evidence.

This case highlights that the privacy of an individual is directly related to the privacy of their direct relations [86] . In this case, the result was justice. The ACLU describes this privacy as “networked privacy” [32] , comparing the mechanisms of finding an individual similar to Facebook’s networking API, which describes members on the site in a connected traversable graph.

In the words of Eric Rescorola, CTO of Firefox, for publicly available genomic tools the “intended use case and the adversarial use case” are the same [86] . Therein lays the difficulty, and to some the impossibility [75] , of creating a legal system to protect the rights of individuals.

Genomic data is the most deeply identifying feature of an individual. It is the clearest identifier of the human body; of human lineage; and with the rising field of epigenetics, which studies how environmental factors after birth influence the gene expression of an both an individual and the descendants of that individual, of human actions [31] .



This thesis posits a solution to address the challenge of keeping an individual's genomic data private while still allowing (hopefully) honest but curious parties to extract meaningful data from it.

### **2.1.1 Keeping data private from other individuals**

Linda Avey, cofounder of 23andMe, said “it is a fallacy to think that genomic data can be fully anonymized” [98]. And in the words of Laura Lyman Rodriguez, the director of policy, communications and education at the National Human Genome Research Institute, “DNA is so unique, and there are so many data sources out there, that it is incredibly hard to fully anonymize — and more so to promise and provide any absolute guarantee that the data are anonymized” [98].

The system described and prototyped in this thesis does not address all possible genomic misuse scenarios. Instead, the system protects against honest but curious parties. It allows only that access to which the owner (in the crypto-game, Alice) permits. By leveraging homomorphic encryption, the system removes the requirement of a trusted third party to police allowed and disallowed queries. In this system, the only parties are Alice and her computer, and Bob and his computer.

For example, consider the scenario in which a patient (Alice) would like to know if she is an increased candidate for breast cancer. She does not feel comfortable allowing her physician to know other information about her genome. In this scenario, the physician can tell Alice which genes they would like to assess (for example, a mutation in her BRCA1/2 genes).

Perhaps it may seem contrived that Alice would not want to hide data from her physician; consider the case in which a consumer DNA company like 23AndMe wants to be able to check for certain traits, but some customers may not want the company to know other traits such as ethnic ancestry or markers for addiction likelihood. In this case, customers can control which data the company gets to run analytics on.

In the ASU-Havasupai case, the Havasupai population consented for one trait—diabetes—to be analyzed by ASU researchers. Using this system, Havasupai volunteers would be able to symmetrically encrypt only the genes requested by the original researchers to look for the incidence of diabetes. Without the active consent of the study participants, the researchers cannot look for other genes in the genome. Additionally, researchers cannot decrypt the genomes to make copies, storing the participant's genomic data for longer than originally agreed upon.

Genomic research is critical to advances in healthcare. However, no participant should have to give up their complete control of their genome to allow for targeted research to occur on their genetic material. The system proposed in this paper allows for the consented querying of genetic information, while preventing gross negligence of this private data. We see this system as being a step in protecting patients' rights.

Patients should be in control of their privacy; when they go to the doctor's office, and when they selflessly volunteer to further medical research.

In the absence of laws and policy, and especially in societies in which the legal system is both impacted by the political environment and impacts on healthcare regulations, privacy

preserving technology can assure that healthcare goals are met while allowing individuals to implement, rather than exercise, a right to privacy.

### **2.1.2 Limitations of the system**

The system described in this paper only works in cases in which one party holds the data, and the other party is querying the data. This system provides no guarantees of privacy in the case in which one party both owns and queries the databases, such as in the case of police genomic datasets.

### **2.1.3 Challenges in restricting data usage to one party**

One of the central use cases of this system is to restrict access to genomic information to one party. One party can encrypt their genome and queries on that genome such that only one other party can decrypt.

Thus, passing genomic data from one party to the next requires active participation from the party that owns the genomic data, since the owner must both encrypt the new party's patterns and generate re-encryption keys specific for the new party. This prevents scenarios witnessed in the ASU-Havasupai case in which the participants' genomes were freely passed from one researcher to the next without active consent from the participants in the study.

## **2.2 The need for privacy despite existing legal frameworks**

One of the challenges of creating legal frameworks to control authorized access to genomes is that the intended use case is the same as the adversarial use case; the trusted party's operations on the genome are the same as the untrusted party's operations on the genome. It is therefore difficult to create legislation that allows some parties access to genomes, especially when private consumer facing companies have access to so much individual genetic information and yet, as private companies, have fewer restrictions on to whom they can share or sell this data.

It has been noted by several legal scholars [25] that laws governing genomic data are not able to evolve as quickly as genomic analytics are advancing. While legal frameworks protecting privacy do exist such as the Genomic Privacy Act, Genetic Information Nondiscrimination Act (GINA) and HIPAA, these laws restrict only a subset of organizations, and have minimal impact on private companies that process genomic data [25]

The legality of the use of genomic data is often determined on a case-by-case basis. As seen in the cases mentioned in this chapter, courts often decrease restrictions on the privacy of genomic information in favor of allowing for relatively straightforward prosecution cases, such as in the prosecution of the Golden State Killer; however, the precedent set by these cases is then used to allow for broader access to genomic data in cases that are not strictly criminal, such as the case of prosecution of individuals based on discarded genomic material without judicial oversight or a warrant process.

### 3 Related work

While this thesis sits on the foundation set by past research in cryptography, DNA security, and security systems, the central differentiation between this thesis and past work, and the value add of this thesis, is the implementation of cryptographic research into a visual user interface useable by professionals without software development experience.

Many software libraries and APIs provide encryption and homomorphic encryption functions. However, the purpose of this thesis is to create a complete system with a visual user interface, usable by researchers with no software development or cryptography experience.

Currently, there is no existing product that allows a user completely or partially encrypt their database, export that database and be able to decrypt it, and then run operations on the encrypted database in an untrusted online environment.

#### 3.1 Background on Homomorphic Encryption

##### 3.1.1 Homomorphic encryption in insecure environments

Much of the literature on homomorphic encryption in healthcare settings focuses on the need to move sensitive healthcare data to cloud-based systems, which can allow for better sharing of data, harness more robust computational power, and allow for more computationally expensive analytics to be performed on healthcare data, especially when working with Electronic Health Records [100].

Vengadapurvaja et al (2017) discuss the use of homomorphic encryption to encrypt medical image data, allowing it to be uploaded to cloud based systems. Their solution builds a custom homomorphic encryption library with a focus on efficiency of encryption and decryption in a cloud system, and their FHE scheme supports multiplication and addition, allowing for basic image processing functionalities to be run on the encrypted data.

While their use case focused on the financial sector, Masters et al presented an application of the FHE scheme CKKS, an FHE scheme located in the HELib software library, to create a secure big data pipeline on financial data. Additionally, they showed that they were able to run basic machine learning algorithms on the CKKS encrypted data. Specifically, they ran a logistical regression on binary data encrypted with CKKS successfully [66].

Similarly, as the Masters et al study, FHE has been applied to patient data to support machine learning capabilities in an untrusted environment, specifically for cancer research applications. In their 2019 paper, Paddock et al encrypted patient identifiers in their database, while relevant tags of cancer type were kept in plaintext [72]. The researchers used the HomomorphicEncryption library native to R and showed that they were able to train and predict on this data set using basic machine learning models. Importantly, they discuss the need for a solution that will anonymize data without modifying the original database, stating that “past solutions to either completely anonymize data or restrict access through stringent data use agreements have limited the utility of abundant and valuable data [72].”

This study found that while learning on partially encrypted FHE data was time consuming, it was possible and successful in a cancer analytics study. The results of this study highlight the

need to encrypt databases to allow researchers to use sensitive data in less restricted computational environments.

### **3.1.2 Using Fully Homomorphic Encryption to ensure privacy: technical solutions to address criticisms**

Fully homomorphic encryption allows data to be encrypted while it is operated upon, thus assuring that even if the encrypted dataset is leaked the privacy of the data remains intact. Although FHE offers strong privacy guarantees, primary criticism of FHE systems is that they are slow and memory-intensive. Thus, design choices of data representation, scheme selection, and parallelization have been selected to reduce runtime and memory usage of each operation.

In this section, we explore the challenges of implementing a practical FHE system to explain the technical design decisions made in the system.

#### **3.1.2.1 Modern cryptographic schemes should be post-quantum secure**

One of the critiques of relying on cryptography to provide privacy guarantees is security properties of cryptographic schemes when operated upon a quantum computer.

Cryptographic schemes derive their security guarantees from mathematical promises of hardness of their primitives; if the underlying primitives are insecure in a post-quantum setting, so too is the scheme relying on these primitives.

This critique is legitimate; several schemes have been shown to break under post quantum conditions. Schemes that rely on the hardness of the discrete log problem, i.e., computing the discrete log in cyclic groups is hard, fail in a post-quantum world since the discrete log problem can be solved in polynomial time (efficiently) using a quantum computer as shown by Shor's algorithm for prime factorization [91]. This leads to the insecurity of popular PKE schemes such as the Diffie-Helman Key Exchange, RSA, or elliptic-curve cryptosystems.

In this thesis, the homomorphic protocol used derives its security from the hardness of the Learning with Errors problem [84]. Learning With Errors (LWE) is a Lattice-based cryptographic primitive; a rigorous reduction from the worst-case of lattice problems that are considered secure in post-quantum environments [19]. It is therefore currently believed that LWE is secure in post-quantum systems [19].

#### **3.1.2.1 Systems leveraging homomorphic cryptographic security must speed up encryption and operation runtime**

In cryptographic systems, fully homomorphic encryption has been regarded as too slow to be usable in practical settings [59]. This is due to a combination of long ciphertext size leading to large storage requirements, and lengthy compute time for mathematical operations arising from many Fully Homomorphic Encryption schemes such as BGV and CKKS [59]. The large ciphertext size arises from large security parameters, including a large plaintext modulus and large plaintext size.

Genomic DNA presents a unique challenge to analyze due to its large size and lack of logical tokens such as 'words' or 'prefixes'. It is difficult to know the start locations of each gene in the genome and much of DNA consists of non-coding regions, segments of nucleotides that

are not meant to be transcribed, and the variable length these non-coding segments as the DNA incurs damage and changes during repeated cell replication over the lifetime of the organism.

The cryptosystem implemented in this thesis utilizes a the Chillotti-Gama-Georgieva-Izabachene (CGGI) [21] scheme, which is built upon the LWE primitive.

As background, the Gentry-Sahai-Waters (GSW) homomorphic scheme combines LWE with linear algebra to encrypt messages as the eigenvalues of matrixes that have a common eigenvector.

Homomorphic encryption incurs noise both during initial encryption and increases the error during each homomorphic operation. Homomorphic schemes are said to be ‘leveled’, where the level of the ciphertext decreases for each multiplication operation; when the level hits zero the ciphertext is no longer decryptable. The result of this is that the number of operations on each homomorphic encryption is limited.

For this reason, homomorphic ciphertexts must be periodically refreshed to allow for correct decryption. Bootstrapping is a technique proposed by Gentry in his original paper on homomorphic encryption to refresh, or reduce the error term, of the homomorphic ciphertext. Bootstrapping is a traditionally expensive operation; a 2014 implementation of Halevi and Shoup’s bootstrapping algorithm in IBM’s HELib library required approximately six minutes per bootstrap operation [46] [47] . In 2015, Ducas and Micciancio created a scheme, referred to as DM or FHEW, to reduce bootstrapping time dramatically to “less than a second” [30] , but required that only one NAND operation is allowed between consecutive ciphertext refreshing operations [102] .

The CGGI scheme replaces the inner product of the GSW scheme with a simpler external product between a GSW ciphertext and an LWE ciphertext [21] . CGGI then expresses the DM/TFHE bootstrapping scheme in terms of the inner product to reduce bootstrapping time from 1 second to less than 0.1 seconds and reduce the bootstrapping key size from 1 GB to 24 MB without reducing security parameters. Finally, CGGI replaces the exact decomposition result with an approximate decomposition result to reduce noise reduction computation time.

The small message space and plaintext modulus required to encrypt one bit, combined with the bootstrapping optimizations in the CGGI scheme make the binary homomorphic learning with errors scheme perform fast encryption and fast binary operations.

### **3.1.2.1 Parallelization without synchronization allows for maximal sub-threading**

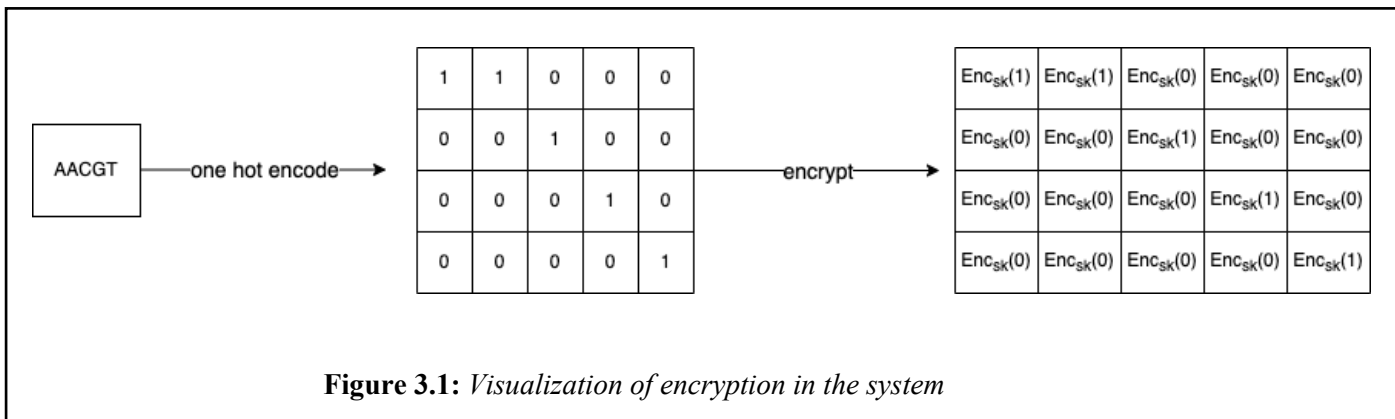
While the lightweight CGGI scheme provides faster encryption and binary operations, parallelization is used to further speed up computations. To be able to use the maximum number of sub-threads available on a machine, encryption, decryption, serialization, and homomorphic operations (homolog/mutation search, substring search, and pattern match) all occur without the need for any synchronization between threads. OpenMP, an open-source multi-threading library, allocates the maximum number of sub-threads to be handled by a system. This is typically 16 on an 8-core machine and 40 on a 20-core machine.

In order to allow threading to occur without the need for synchronization, the data must be represented without the need for ordering of operations.

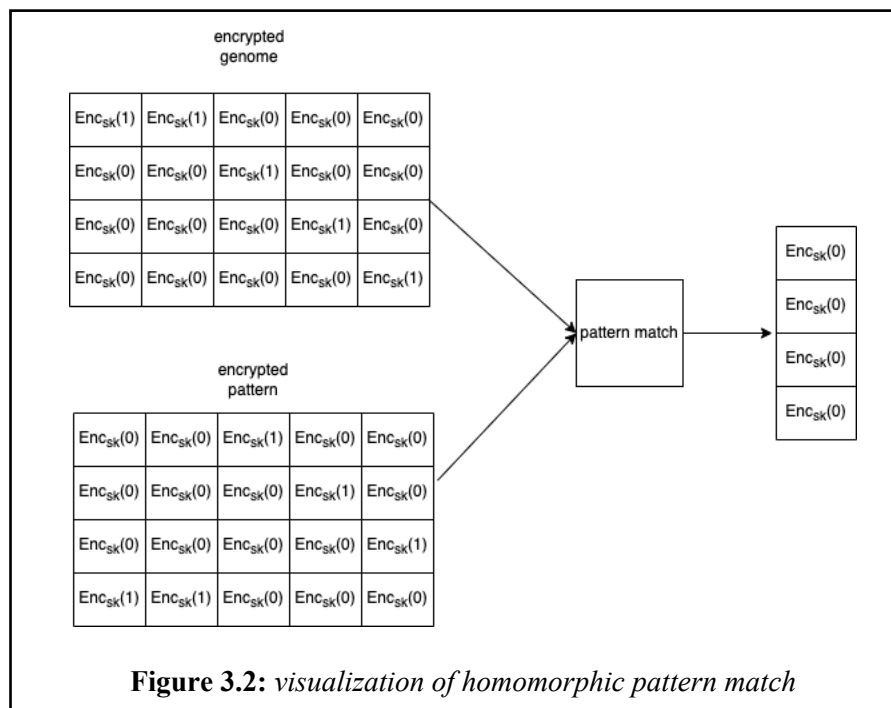
Therefore, the genome is one-hot encoded into a  $4 \times \langle \text{genome length} \rangle$  matrix, where each row in the matrix corresponds to the positions of each nucleotide in the genome. Each value in the matrix is either a 1 or 0 if the nucleotide exists at that position.

Each pattern is encoded in the same way, into a plaintext  $4 \times \langle \text{pattern length} \rangle$  matrix.

During encryption, the structure of the one-hot encoded matrix is preserved. Figure 3.1 below illustrates this:



This matrix representation allows for each encrypted bit to be operated on without need for synchronization between threads; an encrypted pattern bit and an encrypted genome bit can undergo an AND, OR, or XNOR operation independent of other encrypted bits in the system. In other words, each bit can be encrypted in parallel, and each encrypted bit can also be operated on in parallel. Figure 3.2 below shows a visualization of pattern match when a pattern is not found in the genome.



By using a matrix representation for the genomic data, all pattern match operations can occur without the need for synchronization between threads allows for the maximum number of CPU cores available in a system to be utilized at once to allow the maximum number of bits in the genome to be operated on at the same time.

### **3.1.2.1 Cryptographic systems should support memory-constrained environments**

During serialization, each encrypted bit is serialized to a binary file at a time in parallel. The size of each ciphertext is small, and therefore each file is approximately 4 KB in size.

The lean CGGI scheme operates on relatively short ciphertexts, thus reducing RAM usage as the system runs as well.

Finally, multithreading is controlled by OpenMP which not only automatically tailors the number of sub-threads created to the number of processors available but also utilizes a shared-memory model for lower memory bandwidth usage.

## **3.2 Background on existing security systems for DNA**

Security system for databases containing DNA often contain large scale systems that must be used as a single unit. However, there is a sizeable amount of research looking into security of DNA databases. The following section is an overview of a sampling of research in this field.

### **3.2.1 Security in DNA databases should be inherent within the database, and not guaranteed by external factors**

Research in the security of DNA databases tends to focus on forensic use cases. Much of this research is focused on the more urgent need to security on databases as the use case of data in DNA database expands [28] , as outlined in Dahl, J. and Saetnan, A's 2009 paper. Additionally, Guillen et al (2000) point out that DNA databases often hold genetic markers as well as raw DNA data and discuss the ethical ramifications of preserving the privacy of DNA databases for use in a criminal investigation [45] .

The need for security in DNA databases arises from the increased functionality of DNA. Ge, J (2020) notes the technical mechanism for searching relatives in DNA databases for forensic analysis and details the needs for robust security practices in forensic settings [38] .

In their 2000 paper, Bohannon et al outline a method of implementing forensic DNA database to only allow legitimate queries [11] . Specifically, they focus on the requiring prior genetic information about an individual to be passed in as input before a query, such as familial relation, about that individual can be made on the database.

To address the ability to share DNA data without violating the privacy of personally identifiable data in the database, Nassar et al (2019) outline a deterministic method for aggregate statistical queries on DNA data [68] .

In a survey of existing literature, Arshad et al (2021) provide outline privacy and security challenges of DNA databases and applications accessing those databases[8] . Particularly, they highlight the vulnerable software that DNA researchers use while accessing sensitive genomic databases.

Overall, the existing literature highlight the need to security within genomic databases that does not depend on external systems for security and privacy.

### 3.2.2 Background on advances in private information retrieval

Advances in encrypted genomic similarity operations have leveraged encryption protocols to privately compute across the allelic profiles of many patients. In a 2018 study by Asharov, Halevi, Lindel, and Rabin, a Similar Patient Query operation was privately run on remote genomic data [5]. This operation was used to find similar patients with similar genomic profiles across a large patient dataset while preserving the privacy of each patient's genomic profile from the server. Asharov et. al. demonstrated the private search compare operation via applying an approximation method on the compare operation to allow for private computation, and then optimization a two-party protocol to complete the operation. Their protocol presented low runtime on databases containing thousands of alleles for many patients, and showed linearly scaled performance with linearly increasing database size. Specifically, the showed the two-party protocol achieved a 1 second performance to identify the nearest 5 records to a query on a dataset of size 500 and length 3500 sequences. Asharov et al demonstrated that a 2 party protocol combined with private approximation techniques can yield approximately 3 times performance gain as compared to existing secure protocols that utilize existing edit distance algorithms. Asharov et. al. demonstrated that modified protocols can be privacy preserving and show performance gains by leveraging approximation algorithms. Their solution also yields performance gains by having the client and server pre-process their inputs into efficient data structures. The idea of preprocessing inputs to decrease runtime is borrowed in the FlexFHE system; the data is transformed into a data structure that can be exploited for performance gains on private operations.

In developing private search on database systems, several papers explore different threat models and suggest solutions tailored to different security requirements. In their 2015 paper titled Malicious-Client Security in Blind Seer: A Scalable Private DMBS, authors Fish, Vo, Krell, Kumarasubramanian, Kolesnikov, Malkin, and Bellovin describe a modified Blind Seer database management system that allows client query privacy and server data privacy by encrypting the client input with the query evaluation and policy check circuits provided by Blind Seer, and encrypting the database record result with a key that is obtained only when the query is successfully answered and the privacy policy are found to be followed [36]. In this way, they were able to solve a threat model that allows a client and server to communicate privately when the client is malicious. Their solution relied on the utilization of bloom filters to counteract malicious attempts to modify query results on the client end, an improved garbled policy circuit, and an improved modification of the query circuit described in Blind Seer [73].

Exploring private data retrieval under a slightly different threat model in which the server is honest but curious and the multiple clients are malicious, 2013 paper by Jarecki, Jutla, Krawczyk, Rosu and Steiner explored symmetric private information retrieval [53] by showing a protocol that allowed a client to learn only the information from a database to which they were given authorization, while ensuring the privacy of the results of the client's query. Their solution provides for the ability to enforce authorization rules without the database owner learning the privacy policy. As part of their solution, they utilized a homomorphic signature scheme to sign trapdoors given to the client. The client then homomorphically transformed the signatures on their search tokens. Jarecki et al showed that



by leveraging homomorphic signatures, forgery of the tokens or signatures is not possible by malicious clients, thus preserving the integrity of the system.

These works show different approaches to privacy preserving database search. All leverage unique architectures and specific cryptographic protocols to achieve security against specific threat models. Similarly in this work, the architecture is built to support a specific threat model as well, and data privacy is provided and client authorization is enforced by the underlying homomorphic encryption of the data.

### 3.3 Encrypted genomic search related work, and how FlexFHE differs

Much of the related work on encrypted genomic search utilizes a SNP table, encrypting it into a tree structure, and doing an encrypted logical traversal to identify string match. For example, Mahdi et al. (2021) [63] build a generalize suffix tree from a haplotype SNP representation . Figure 3.3, taken from the 2021 paper, shows a sample of the SNP table representation used in this paper.

Sample haplotype SNP data representation.						
#	$SNP_1$	$SNP_2$	$SNP_3$	$SNP_4$	$SNP_5$	$SNP_6$
1	1	0	0	0	1	0
2	1	1	1	0	1	0
⋮						
$n$	0	1	0	1	0	1

Figure 3.3 Haplotype SNP representation used in Mahdi et al.  
Taken from source: Mahdi et al 2021[63]

Mahdi et al use a set maximal search to perform a pattern match between two SNP tables as well as identify differences between a sequence and the genotype.

Instead of using a haplotype SNP data representation, the system described in this thesis performs search on a sequenced DNA segment, without the need for a SNP table. While the operations performed are similar, the system implemented in this thesis is able to perform secure substring search, find the percent match between to sequences, and identify mutations via simple wildcard notation on sequences which do not easily have a SNP table representation such as plasmids. Additionally, the system is easily extensible to support RNA variants using only the RNA sequence, and no additional data.

Another way of approaching encrypted substring search on a string is presented by Bonte and Iliashenko in their 2015 paper [13] . In this paper, they split the string into vectors of fixed length, and use homomorphic subtraction to perform substring search with a constant multiplicative depth. While this system is effective in reducing search time by reducing multiplicative depth of homomorphic operations, it utilizes a costly Homomorphic equality protocol which utilizes homomorphic AND, Addition, subtraction, and a non-zero check . In this protocol, the length of the substring must be known ahead of time as it is used to partition and encrypt the larger text, and the text must be re-encrypted for queries of varying sizes.

The protocol described in this thesis is able to encrypt and serialize one bit at a time, allowing both for parallelization and for support of memory constrained systems. Additionally,

homomorphic operations only work on one bit a time, and equality checking is done via an XNOR operation which is relatively fast. Most importantly, the length of the query is irrelevant in the encryption of the larger text (genome). The genome may be encrypted once and operated on with ciphertexts of any length thereafter.

## 4 System Architecture

This section provides a technical description of the architecture of the system.

### 4.1 Addressing the operational model in the context of the threat model

The purpose of this project is to provide a system that can be used by professionals without a software engineering background to provide security to database systems existing in untrusted environments.

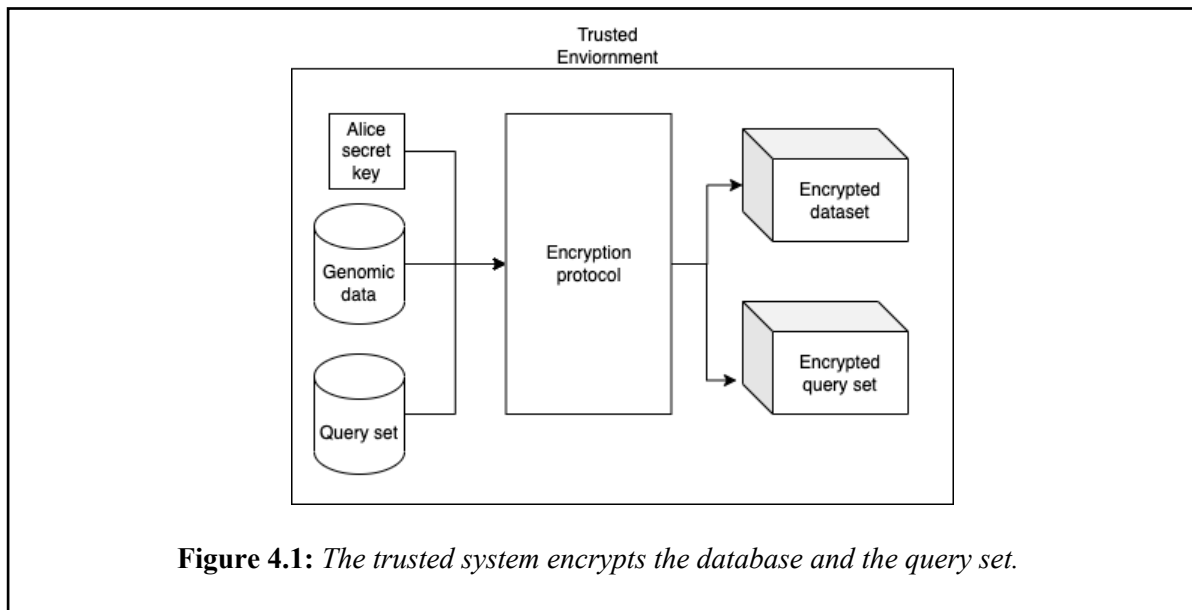
One of the known difficulties of working with data encrypted with fully homomorphic encryption is the lengthy runtime of running operations, making the system inefficient. The success of the system comes not from leveraging fully homomorphic encryption, but rather by leveraging a partially homomorphic encryption of the database based on the desired operations.

Overall, this software is split into two parts; one which represents an untrusted environment and the other which represents a trusted environment.

#### 4.1.1 Trusted environment

The trusted environment contains the system and software to partially or completely homomorphically encrypt a database containing DNA.

The user passes in their database into the software and can then export their partially or completely homomorphically encrypted database. The trusted environment stores the master key for decryption of the data. Figure 4.1 below represents the encryption of the database in the trusted environment. Please note that while not included in the picture, encryption and decryption is done via a Master Key stored in the Trusted System.



**Figure 4.1:** *The trusted system encrypts the database and the query set.*

### 4.1.2 Untrusted environment

The untrusted environment contains all the operations to run on the encrypted database, such as direct analysis of one individual's DNA or aggregate statistical analysis on many individuals' DNA. Figure 4.2 below represents operations run on the encrypted data in the untrusted environment.

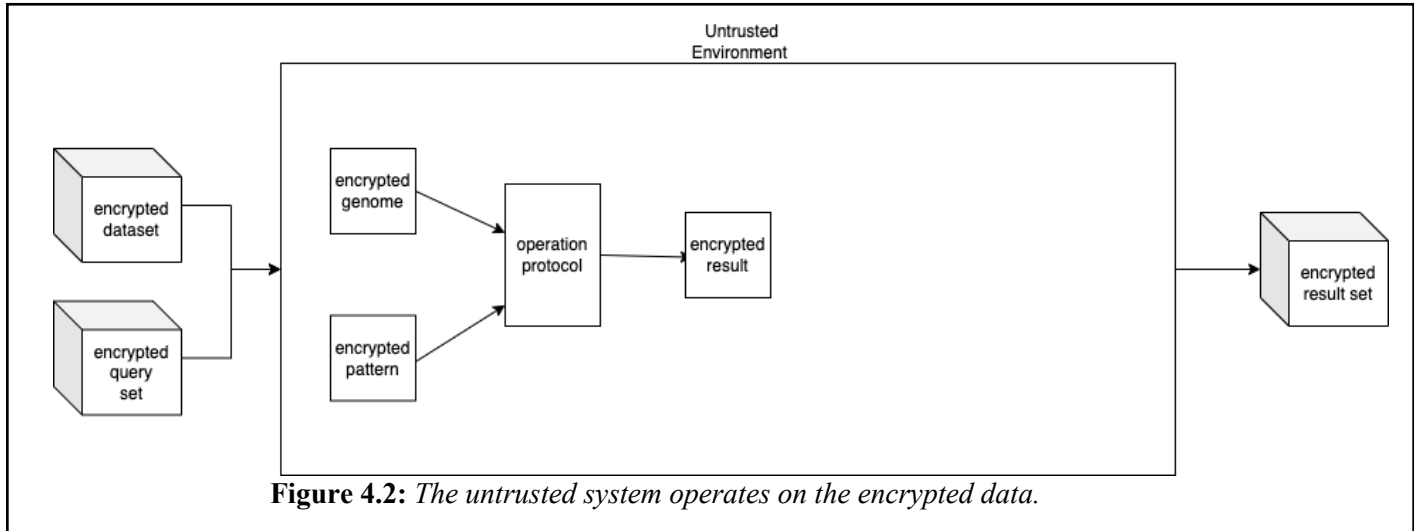


Figure 4.2: The untrusted system operates on the encrypted data.

### 4.1.3 Untrusted environment: analysis of components

The untrusted environment is meant to model an environment, in which an adversary has access to all components in the environment. The following section describes an analysis of the assets in the untrusted environment, which are inherently corruptible.

#### 4.1.3.1 Re-encryption to querier's key: tradeoff between engineering efficiency and query confidentiality

In the untrusted environment, data in the database exists only in encrypted form. The data is homomorphically encrypted to the master key. In other words, at the completion of a functional operation, the result must be decrypted with the master key.

In the operational model, the querier may be corrupted. This is meant to represent both the scenario of internal threats, in which a person with access to the database cannot be trusted, and the scenario of cloud-based computations, in which computations run on external servers can be eavesdropped upon and intercepted.

For this reason, it is undesirable for the master key to be used in the untrusted environment, or for the querier to have access to the master key for decryption. And it is desirable for the owner of the database (the holder of the master key) to be able to retract access to the database when needed.

##### 4.1.3.1.1 Re-encryption strategies fail in the untrusted environment

Ideally, re-encryption to the querier's key occurs in the untrusted environment. Re-encryption would occur via Gentry's Recrypt protocol, via TFHE's Key Switching protocol, or via OpenFHE's scheme switching protocol.

Gentry's Recrypt protocol utilizes a PKE public key and secret key pair to generate a re-encryption key. In the system described in this thesis, symmetric encryption was selected to allow Alice to control the queries that Bob runs on her dataset without the use of an additional authorization mechanism, therefore this cannot be used in the untrusted system.

TFHE/CGGI defines a key switching protocol in which allows a ciphertext symmetrically encrypted under one key to be modified such that it will decrypt with another key. In CGGI, each ciphertext is a large vector and key switching is achieved via a homomorphic multiplication operation. In their original paper, CGGI defines a key switching protocol for a Ring Learning With Error based scheme. Key switching occurs by mathematically canceling the secret key of the original ciphertext and re-encrypting under a new secret key.

At the end of the key switching operation, the secret key has switched but the message is constant. However, to do this process the untrusted system must have access to a re-encryption key, which is created with both Bob and Alice's secret keys. If Alice is to give a key-switching key to the untrusted environment, she would require access to Bob's secret key, injecting an additional level of trust within the system. Bob can be corrupted to give a malicious party's secret key to Alice, and the untrusted system can be corrupted to re-encrypt the entire dataset with the key-switching key, thereby allowing a malicious party to decrypt the dataset.

Additionally, in practice, the symmetric key switching operation must be accompanied by a noise flooding operation that adds random noise to the prior error which reveals information about the original secret key. Noise flooding is an operation that adds noise bits to a ciphertext. As background, the error in ciphertexts is always sampled from a distribution. Where the error distribution of the base scheme is bounded from  $[-B, B]$ ,  $\lambda$  is the security parameter, the error distribution for the additional error added to the ciphertext (the noise flooding) should approximately be  $[-B2^{\lambda}, B2^{\lambda}]$  [4] to allow for a sufficiently large standard deviation in the noise distribution [4] [70], in practice however, noise flooding is often set as a constant value deemed to be large enough which provides slightly less than 128 bits of security, even if it is acceptable

The current implementation of key switching in OpenFHE only supports switching from a RLWE ciphertext to a LWE ciphertext, without noise flooding. Adding the noise flooding operation would therefore not only increase the complexity of the code base but also cause a slowdown in key switching and subsequent decryption. As a note to the user, CKKS has implemented noise flooding to 128 bits of security, but only for the CKKS scheme and with slowdown [70].

#### **4.1.3.1.2 Why not utilize a PKE scheme with built in re-encryption**

A PKE protocol would allow for simple Proxy Re-Encryption per Gentry's Recrypt protocol. However, utilizing a PKE protocol would allow anyone with Alice's public key to query the dataset; in the threat model, Alice desires to restrict the possible queries on her dataset and enforces the restriction by symmetrically encrypting only the patterns she approves.

Additionally, PKE schemes like CKKS, BGV, and BFV use larger security parameters and larger ciphertext size than CGGI. Only CGGI can achieve a ciphertext size of less than 1MB per plaintext and is able to run on less than 2GB of RAM [88]. Meanwhile, CKKS, BGV, and BFV are all vectorized schemes, encrypting an entire vector of data into a single

ciphertext. In practice, the data in each vector is only a subset of the text, which in this case is a genome. Encrypt time of an entire genome into a single ciphertext is extremely high and is therefore unsuitable for RAM-constrained machines.

The encryption of vectorized ciphertexts is also difficult to parallelize, since each vector must be encrypted all together in a synchronized manner.

Finally, substring search or pattern matching is slow on vectorized ciphertexts. Encrypted search algorithms on vectorized ciphertexts rely either on hashing or on tree traversal. Algorithms like Rabin-Karp use hashing to find patterns in strings. Encrypted Rabin-Karp hashes each vector and uses a homomorphic subtraction operation to identify the substring in a text. The drawback to this algorithm is that although it runs in time  $O(n + m)$  in the best case and  $O(nm)$  in the worst case where  $m$  is the pattern size and  $n$  is the text size, each homomorphic “compare” operation requires several multiplication, addition, and subtraction operations each of which take significant time; especially when compared to a binary XNOR operation.

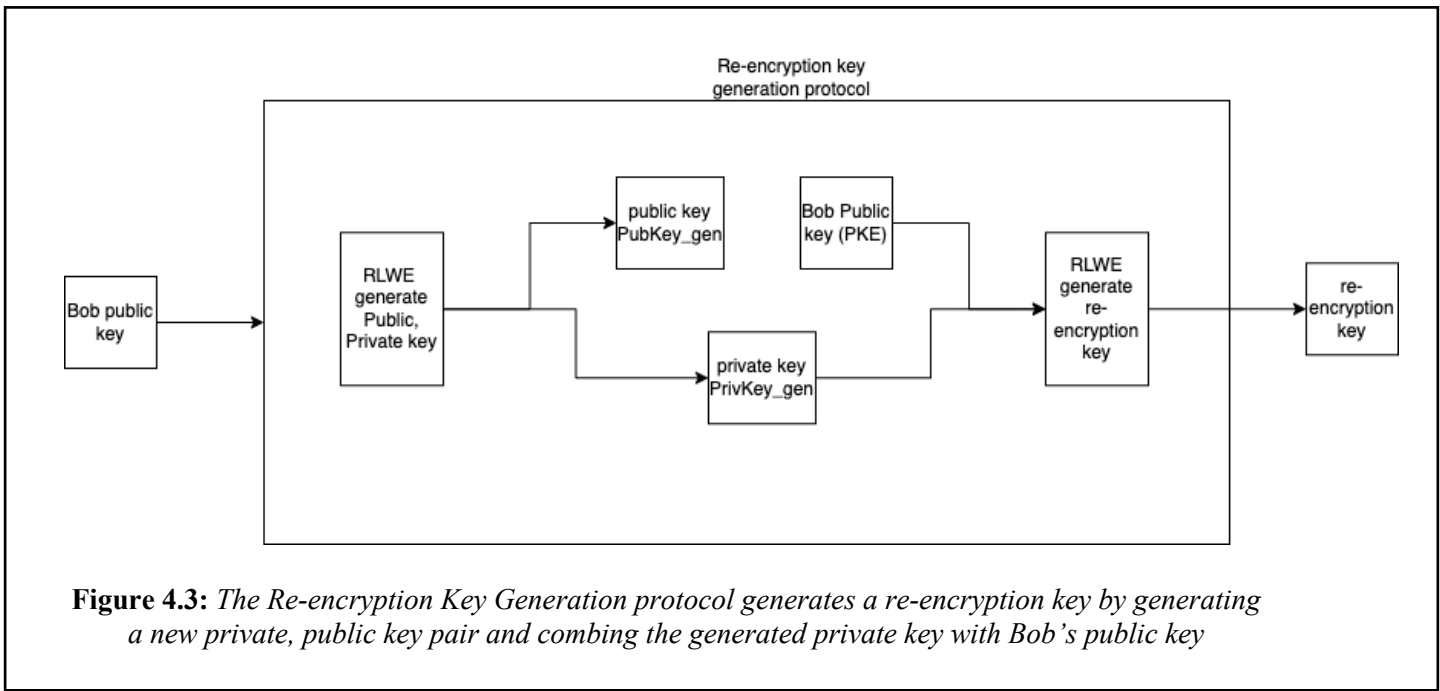
#### **4.1.3.1.3 Implemented solution: Alice performs proxy re-encryption**

The implemented solution was selected with the goal of minimizing RAM and disk memory requirements and minimizing pattern match operation runtime and encryption time on the large genomes, while still adhering to the operational model informed by the threat model.

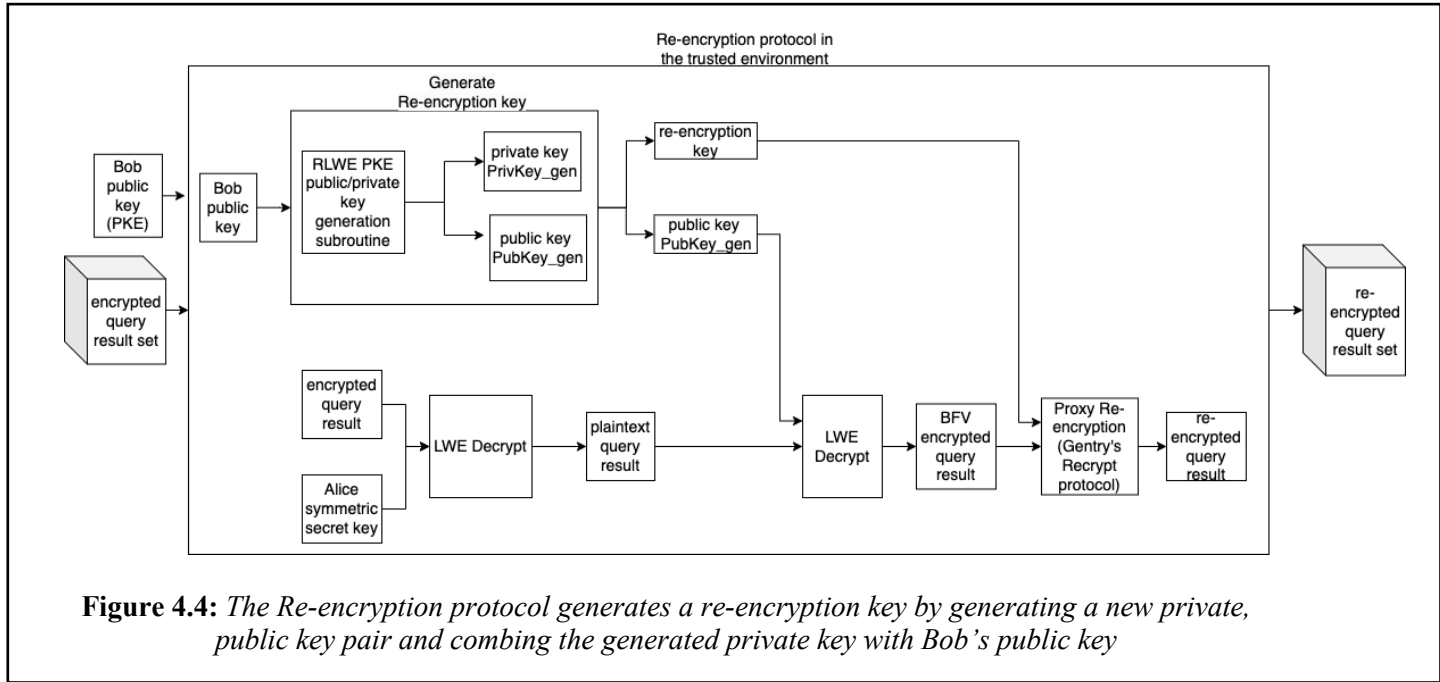
In the threat model, Alice is the owner of the dataset and Bob is assumed to be honest but curious, while Bob’s Untrusted Environment is malicious. The corresponding operational model allows Bob to use the untrusted environment to query the dataset. In the model, Alice has access to the queries, the dataset, and Bob’s public key; therefore, our threat model allows Alice to query the dataset herself. In other words, our threat model supports Alice seeing the result of Bob’s queries.

Therefore, upon completion of the query, the Untrusted Environment will output the ciphertext result to Bob. Bob will send the ciphertext to Alice, who will decrypt the ciphertext, re-encrypt the ciphertext with a BFV scheme, and perform a proxy-re-encryption operation using Bob’s public key. She then sends the PRE encrypted cyphertext to Bob, who decrypts with his secret key.

A visualization of how an encrypted query result from the untrusted environment is re-encrypted to be decrypted with the querier’s private key is shown in Figures 4.3 and 4.4 bellow. Figure 4.3 shows the re-encryption key generation protocol in detail, while Figure 4.4 shows the trusted system utilizing the re-encryption key generation protocol as a subroutine in the re-encryption of the ciphertext.



**Figure 4.3:** The Re-encryption Key Generation protocol generates a re-encryption key by generating a new private, public key pair and combining the generated private key with Bob’s public key



**Figure 4.4:** The Re-encryption protocol generates a re-encryption key by generating a new private, public key pair and combining the generated private key with Bob’s public key

**4.1.3.2 Running functional operations on the database**

In the untrusted environment, the database exists in encrypted form. While the untrusted environment has access to the queries on the database, the untrusted environment does not have access to the result of those queries.

Privacy preserving queries on databases is a strong branch of cybersecurity and cryptographic research. It is also well known that queries can leak at least some data about the database [99]. Regarding queries that answer only aggregate questions, Cho et al. (2020) note that even aggregate query answers can leak data about the dataset [24]. Cho et al finds a solution in the

usage of differential privacy techniques to differentially modify the dataset, and Vaidya et al (2013) bound the attack query inference to statistical dataset, describing solutions that audit information leaked by queries, utilizing differential privacy techniques to modify the queries, and using differential privacy techniques to modify the dataset.

In this system, we constrain the scope of the definition of security to that of the output of the queries, and not of the queries themselves. In the trade-off of security and usability, we allow the queries in the untrusted environment to operate on the encrypted dataset directly, instead relying on the cryptographic security of the output of these queries.

#### **4.1.4 Key Management**

During re-encryption time, the trusted environment generates a new private, public key pair. It decrypts the ciphertext using the owner's secret key and re-encrypts the ciphertext using Brakerski/Fan-Vercauteren scheme (BFV), which is a leveled FHE protocol utilizing private/public keys from public key encryption that obtains its security from the hardness of ring learning with errors (RLWE). RLWE is a generalization of LWE to polynomial rings over a finite field. RLWE supports PKE while LWE encryption supports only symmetric encryption. For this reason, systems that are built on top of RLWE encryption can more readily implement Gentry's Recrypt protocol, which utilizes the private key of one party and the public/private key pair of another party to create a third re-encryption key.

The untrusted environment uses a key generation protocol which takes as input the private key from the public key/private key pair that was newly generated and Bob's public key.

As a note, while the trusted environment could simply encrypt the result of the query symmetrically using a LWE scheme, doing so would require Bob to give his secret key to Alice.



## 5 Data Flow

This section describes the flow of information throughout the system. The Trusted Environment and the Untrusted Environment represent Alice and Bob, respectively. Alice is trusted, and Bob is either untrusted, or honest but curious. The security of the system is adaptable to the trustworthiness of Bob, or the trustworthiness of the Untrusted System.

### 5.1 Trusted Environment

We will first describe the overall data flow through the trusted environment, and then expand on each operation taken in the trusted environment.

#### 5.1.1 Data Flow Overview in the Trusted Environment

For context, we say that Alice interacts with the trusted environment, and Bob interacts with the untrusted environment.

Begin in the trusted environment. This environment is one that is protected by external systems and organizations. One example of a trusted environment is an “air gapped” computer that does not have access to an external internet source. Another example is a computer located within an organization that provides reasonably strong security guarantees, or an organization within which plaintext analysis of a genome is allowed such as a secure lab within a hospital.

The trusted environment script will run in the trusted environment and will access the genome file within the trusted environment.

First, the genome will be one-hot-encoded. Four vectors one-hot vectors will be created, one representing the positions of A within the genome, one representing the indexes of C within the genome, one representing the indexes of T, and one representing the indexes of G within the genome. The four vectors are appended to another vector, creating a  $4 \times \langle \text{length of genome} \rangle$  representation of the genome.

The first one-hot vector contains a 1 if the genome has an A nucleotide at the index, and a 0 otherwise. The is applied to the one-hot vectors representing C, G, and T. An example of a one-hot encoding of the nucleotide sequence ‘AATT’ is shown below:

1100
0000
0000
0011

**Figure 8.2:** A one-hot encoding of the nucleotide sequence AATT. The first row represents the positions of the A nucleotide, the second row the positions of the C nucleotides, the third row the positions of the G nucleotides, and the fourth row the positions of the T nucleotides.

Alice will also encrypt the patterns that Bob would like to match against the genome. The patterns are also one-hot encoded. The resultant matrixes will be of size  $4 \times \langle \text{length of pattern} \rangle$ . The patterns can be of any length.

The Trusted System will encrypt each one-hot encoded using a Learning With Errors (LWE) fully homomorphic scheme that encrypts only binary values.

A binary LWE FHE is chosen because it drastically decreases the encryption time. Additionally, it allows the serialization of the ciphertext to occur one bit at a time, allowing for systems with limited RAM and memory to serialize the genome in a reasonable amount of time (please see chapter 6 for performance metrics).

Additionally, a Binary LWE FHE scheme creates very small ciphertexts as only one bit is encrypted. Ciphertexts created by probabilistic encryption schemes are highly random. Compression protocols like ZIP or TAR rely on patterns and uniformity within a file to compress the data within a file to create a compressed file of decreased file. Because compression protocols are largely ineffective at compressing serialized ciphertexts, the ciphertexts must be small to allow Alice to download her genome locally.

A  $4 \times \langle \text{length of sequence} \rangle$  vector will be created, where each index represents an encrypted 1 or 0.

Because LWE introduces random error during encryption, two identical bits will be encrypted to different ciphertexts.

Bob has a PKE public, private key pair while Alice has a symmetric key. If Bob is honest but curious, during re-encryption time, the trusted environment will use Bob's public key to transform the ciphertext encrypted under Alice's secret key into that may be decrypted by Bob's private key.

Fully Homomorphic Schemes is only IND-CPA; thus, it will leak the Alice's secret key in non-adaptive chosen ciphertext attacks.

If Bob is untrusted, then Bob should not receive any ciphertexts. In this setting, Bob receives only the plaintext result of his query.

However, consider the threat model in which Bob is honest but curious and is not expected to launch a chosen ciphertext attack. In this case, the Untrusted Environment (Bob's computer) may remain completely insecure while Bob receives a ciphertext only he can decrypt.

The trusted environment will compress the switching key, encrypted patterns, and encrypted genome into a file system. Alice will then send this tar file to the Untrusted Environment.

### 5.1.2 Binary Encoding Discussion

Why convert the genome and patterns into a binary encoding?

One-hot encoding converts a sequence of nucleotides into a matrix of binary data. In code, this is represented in both 2-D array and a 2-D vector. A matrix representation was chosen because it is easy to transpose the matrix.

### **Why a matrix?**

By having data in matrix format, we can easily transpose the matrix from a  $4 \times \langle \text{sequence length} \rangle$  to a  $\langle \text{sequence length} \rangle \times 4$  matrix. When parallelizing the loop, the transposed matrix representation prevents systems with a small number of CPU cores from falling into the trap of only using 4 sub-threads, when more are potentially available.

OpenMP [17], an open-source implementation of multithreading that supports shared memory multiprocessing, is used to parallelize the encryption. Using openMP, the iteration on the rows and the iterations on the columns of the matrix is collapsed. OpenMP then uses the Fork-Join model to parallelize the loop, wherein the primary thread forks into several sub-threads and then divides the tasks amongst sub-threads. The runtime environment then allocates the sub-threads to different processors on the system to allow the sub-threads to run concurrently.

The one-hot model allows the matrix (2-d char array) to allocate memory dynamically on the heap before encryption begins, and column and rows are easily indexed. Thus, openMP can be directed to allocate the maximum number of sub-threads that the system can support, and no synchronization between the sub-threads is needed. In every encrypted operation, the pattern of allocating the matrix on the heap and then using maximizing the number of threads the system can support (i.e., 16 for an 8-core machine and 40 for a 20-core machine). This means that compared to the same code without threading, an 8-core machine experiences a 16-fold speedup in completing a full operation (“full operation” meaning completing a query, i.e., encrypting a complete genome, finding a pattern, finding a homolog, finding a percent match) and a 20-core machine experiences a 40-fold speedup in completing the operation.

### **Why binary data?**

Because the data exists in binary form, time to encrypt is very simple: one bit is encrypted at a time. This means that traditional approaches to speed up encryption, such as batching, are not used.

### **Why not rely on batching for speedup?**

Batching encrypts many messages into a one ciphertext, so that one homomorphic operation can act on many messages at once, as defined by the Chinese Remainder Theorem.

Plaintext modulus is a security parameter that impacts both correctness and performance. When the plaintext modulus is too small, overflow occurs, but a large plaintext modulus decreases performance of encryption. The plaintext modulus is 1 modulo the ring dimension. Typical default values of the plaintext modulus is  $2^{16} + 1$ , which is considered conservative; in an example provided by OpenFHE, encrypting Anna Karenina requires a plaintext modulus of 786433 [71].

In batching, the longer the plaintext modulus, the larger the security parameter required by the scheme. Theoretically, if the size of the security parameter is poly(size of the message) we can say the system is secure and efficient.

However, practically, when the message is very long and the security parameter is very large (even when it is polynomial in the size of the message), encryption of the message takes a very long time. Additionally, the encrypted data needs to be stored in RAM until it is serialized. In systems with constrained RAM (i.e., less than 16 GB RAM), this often causes the process to crash. Encoding even very small subsections of genomes of less than 10,000 base pairs requires substantial RAM. Finally, the entire ciphertext must be serialized all at once, writing a substantially large ciphertext file to disk. Finally, parallelization is restricted during encryption since the batch operations must occur serially.

### **Binary data simplifies serialization and deserialization.**

Encrypting one bit at a time allows the system to serialize each encrypted bit of the plaintext at a time. Each serialized encrypted bit is written to a single file with a label indicating its index (column indexing is defined by folder the file is saved to). Thus, serialization of the encrypted genome can occur in parallel and with the maximum number of sub-threads possible supported by the machine, managed by OpenMP and without any need to synchronize the threads or utilize a mutex lock.

### **Binary data has a small message space, quick encrypt time, and quick operations.**

Because each bit is encrypted one at a time, the plaintext modulus is 2. FHE schemes like BGV, BFV, and CKKS are all based on Learning With Errors encryption; so much so that OpenFHE defines a “Binary FHE” library which is a CGGI/GINX scheme, a wrapper on the Learning With Errors encryption scheme. Each higher-level scheme is built on top of the binary Learning With Errors scheme.

The Binary LWE scheme provides minimal operations (AND, OR, and NOT). Operations like XNOR and XOR, used for equality checking between two bits, are composed of combinations of these three binary operations.

Binary LWE allows small security parameters which leads to fast encryption/decryption and small key size, small ciphertext size (approx. 4 KB when serialized into a binary file), and fast rudimentary operations that are easily explained (i.e., an XNOR operation is composed of AND, OR, and NOT operations and thus takes three times as long as each of those).

Compared to a higher-level scheme like BFV, each encrypt and each binary operation is faster; and the system is highly parallelizable, allowing multi-threading to occur at the maximum capacity of the machine the code is running on.

Thus, even though the system described in this thesis only achieves an  $O(nm)$  runtime for each operation, the operations run much faster because they are highly parallelized. Encryption, substring search, percent match, homolog search, and decryption thus occur at fast rates and use less RAM.

### 5.1.3 Step By Step functionality in the Trusted Environment

#### 5.1.3.1 Driver Code

The driver code of the trusted environment works as follows:

The driver code defines a genome, in file name holding the input genome, vector to hold the one-hot genome, vector to hold the one-hot-pattern, vector to hold the encrypted one-hot-genome, and vector to hold the encrypted one-hot-pattern.

The driver code calls `read_from_file` to store the data in the genome file in a plaintext vector. The driver then calls `one_hot_encode` to encode the plaintext genome into a one-hot vector. The driver then calls `one_hot_encode` to encode the plaintext pattern into a one-hot-vector.

The driver code then creates a `CryptoContext` object for the LWE binary fully homomorphic scheme. This object allows the driver to access key generation, encryption, and decryption functions provided by OpenFHE.

The driver uses the `CryptoContext` object to create Alice's secret key and Alice's "switch key", which will be used later for re-encrypting the result of Bob's query before the ciphertext exists the untrusted environment.

The driver then calls `encrypt_me` which encrypts the one-hot pattern and the one-hot genome and stores the result into the vectors representing the ciphertexts of the pattern and the genome.

The driver then calls `serialize_keys_and_contexts` which serializes the encrypted genome and the encrypted plaintext.

#### 5.1.3.2 Read From File

`Read_from_file` takes in as parameters plaintext genome vector, which is passed by reference to the function. `Read_from_file` also takes in the string which holds the absolute path to the genome file on the system.

`Read_from_file` opens the file and populates the plaintext genome vector character by character with the data in the genome file and returns the number of characters in the genome file, or the size of the plaintext genome vector.

#### 5.1.3.3 One Hot Encode

`One_hot_encode` takes in as parameters a plaintext vector of characters which is the read-only input, and a 2-dimensional vector object which contains vectors of plaintext integers passed by reference which is the write-to output.

The function creates four temporary vectors, one that holds the one-hot encoding of the letter 'A', one for the one-hot-encoding of the letter 'C', one for the one-hot-encoding for the letter 'G', and one for the one-hot-encoding of the letter 'T'.

`One_hot_encode` reads through the input one character at time. For each character, it appends the integer 1 to temporary vector corresponding to the letter of that character and appends the integer 0 to all the other temporary vectors.

Upon reaching the end of the input string, it appends the four vectors to output vector, so that the 'A' one hot vector is located at index 0 of the output vector, 'C' vector is located at index 1 of the output vector, 'G' vector is located at index 2 of the output vector, and 'T' vector is located at index 3 of the output vector.

The final output vector is thus a 4 x <length of sequence> matrix represented by a 2-D vector.

#### **5.1.3.4 Serialization**

`Serialize_keys_and_context` takes in a Binary CryptoContext object, a secret key, absolute path as a string which represents the folder to write the serialization data to, and a Boolean value 'serialize' which indicates if the function should serialize the data or not.

If the 'serialize' is false, the function returns.

Otherwise, the function uses the CryptoContext to serialize the secret key, the refresh key, the key switching key, and the CryptoContext object itself into a binary file which is saved on the system.

Serialization of the encrypted ciphertext and patterns occurs in the Encrypt function.

Serialization occurs via OpenFHE's serialization library, which is a simple wrapper on Cereal, open-source software that serializes C++ objects into binary files[14] .

#### **5.1.3.5 Encryption**

Encryption takes in as input the `one_hot_genome` passed by value, the `one_hot_pattern` passed by value, the Binary CryptoContext object passed by value, the ciphertext 2-D vector to populate the encrypted genome into passed by reference, and the ciphertext 2-D vector to populate the encrypted pattern into passed by reference, Alice's secret key passed by value, and a 'serialize' Boolean which indicates if the data should be serialized (true) or not (false).

Encryption iterates through all four indexes of the input one-hot-encoded genome and the input one-hot-encoded pattern. It creates a temporary vector of ciphertexts for each index of the genome. Encryption iteratively encrypts each integer using the secret key and the Binary Crypto Context and appends the encrypted item to the temporary ciphertext vector. At the end of the vector at that index, Encryption appends the temporary ciphertext to the ciphertext genome and pattern variable passed by reference.

The resultant ciphertext genome is thus a 2-D vector that represents a 4 x <genome size> matrix, where each location in the matrix contains the encrypted bit at the same row and column of the one-hot genome.

The resultant ciphertext pattern is thus a 2-D vector that represents a 4 x <genome size> matrix, where each location in the matrix contains the encrypted bit at the same row and column of the one-hot pattern.

If serialization is selected, the Encryption function will also serialize the encrypted data. For the genome and pattern, it will populate four folders ('zero', 'one', 'two', and 'three'). Each folder will contain a 4 KB file which contains the binary serialization of the encrypted bit corresponding to the 2-D one-hot-encoded vector. The name of each file contains the index of each bit.

The example of pattern is shown below:

▼	pattern	Mar 31, 2023 at 9:37 PM	--	Folder
▼	three	Mar 31, 2023 at 9:53 PM	--	Folder
	pat_enc_4.txt	Mar 31, 2023 at 9:53 PM	4 KB	Plain Text
	pat_enc_3.txt	Mar 31, 2023 at 9:53 PM	4 KB	Plain Text
	pat_enc_2.txt	Mar 31, 2023 at 9:53 PM	4 KB	Plain Text
	pat_enc_1.txt	Mar 31, 2023 at 9:53 PM	4 KB	Plain Text
	pat_enc_0.txt	Mar 31, 2023 at 9:53 PM	4 KB	Plain Text
▼	two	Mar 31, 2023 at 9:53 PM	--	Folder
	pat_enc_4.txt	Mar 31, 2023 at 9:53 PM	4 KB	Plain Text
	pat_enc_3.txt	Mar 31, 2023 at 9:53 PM	4 KB	Plain Text
	pat_enc_2.txt	Mar 31, 2023 at 9:53 PM	4 KB	Plain Text
	pat_enc_1.txt	Mar 31, 2023 at 9:53 PM	4 KB	Plain Text
	pat_enc_0.txt	Mar 31, 2023 at 9:53 PM	4 KB	Plain Text
▼	one	Mar 31, 2023 at 9:53 PM	--	Folder
	pat_enc_4.txt	Mar 31, 2023 at 9:53 PM	4 KB	Plain Text
	pat_enc_3.txt	Mar 31, 2023 at 9:53 PM	4 KB	Plain Text
	pat_enc_2.txt	Mar 31, 2023 at 9:53 PM	4 KB	Plain Text
	pat_enc_1.txt	Mar 31, 2023 at 9:53 PM	4 KB	Plain Text
	pat_enc_0.txt	Mar 31, 2023 at 9:53 PM	4 KB	Plain Text
▼	zero	Mar 31, 2023 at 9:53 PM	--	Folder
	pat_enc_4.txt	Mar 31, 2023 at 9:53 PM	4 KB	Plain Text
	pat_enc_3.txt	Mar 31, 2023 at 9:53 PM	4 KB	Plain Text
	pat_enc_2.txt	Mar 31, 2023 at 9:53 PM	4 KB	Plain Text
	pat_enc_1.txt	Mar 31, 2023 at 9:53 PM	4 KB	Plain Text
	pat_enc_0.txt	Mar 31, 2023 at 9:53 PM	4 KB	Plain Text

**Figure 8.4:** A visual representation of the folder structure created during serialization of the encrypted pattern, with the index of the pattern being appended to each file name. In this case, the pattern encrypted of length 5. Each file contains the encryption of one bit of the one-hot-encoded pattern.

## 5.2 Untrusted Environment

### 5.2.1 Data Flow Overview in the Untrusted Environment

The Untrusted Environment will decompress the encrypted patterns and encrypted genome. When Bob sends the command to substring search, homolog search, or get percent patch between a pattern and a genome, the Untrusted Environment will begin the search process.

The Untrusted Environment will create a new LWE Binary FHE context object, which contains the algorithms to homomorphically operate on bits. It will use the homomorphic XNOR function to get the (encrypted result) of the equality of two bits, and the homomorphic AND function to aggregate the result of the equality of the bits. Finally, it will use the homomorphic OR function to identify if a pattern exists or not.

If Bob is honest but curious (i.e., Alice does not expect Bob to launch a CCA attack), the trusted environment will decrypt and re-encrypt the ciphertext result of the query using Bob's public key. The Untrusted Environment will then compress the ciphertext result of the query and download it to Bob's computer, who will send it to the Trusted Environment to get the re-encrypted ciphertext. Bob will then decrypt with his private key.

If Bob is untrusted and is expected to launch a CCA attack, the untrusted environment will simply decrypt the result of the query and return the plaintext query result to Bob.

To allow FHE to be used in the system, we must assume that the Untrusted Environment is also honest but curious and will not launch a CCA attack on the ciphertexts either.

## 5.2.2 Step by Step functionality in the Untrusted Environment

This system describes the driver code and each of the operations in greater detail.

### 5.2.2.1 Driver Code

First, Deserialize is called to convert the serialized files into a  $4 \times \langle \text{length of sequence} \rangle$  representation of the genome, as well as for each of the patterns encrypted.

The driver code operates an infinite loop, listening for each operation to be called.

Each operation is parallelized.

### 5.2.2.2 Encrypted Substring Search

Encrypted Substring Search transposes the binary genome and pattern matrixes and allocates a matrix of size  $\langle \text{genome length} - \text{pattern length} + 1 \rangle \times 4$  matrix on the heap. The parallelized function then performs a simple search operation as follows:

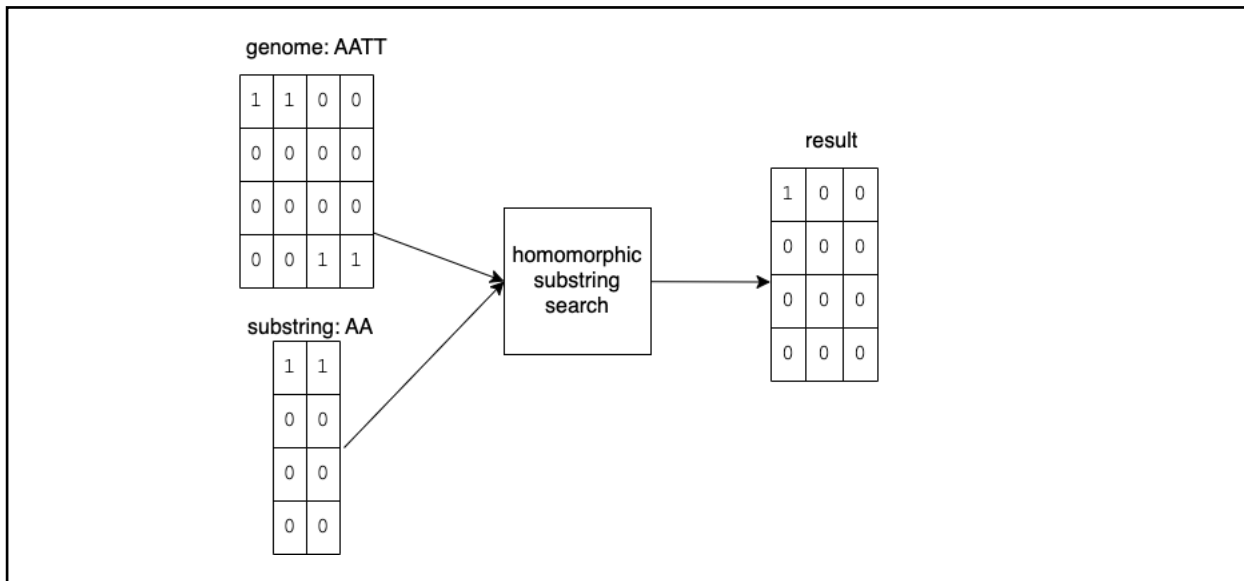
```
For each index in each row the genome:
  Ciphertext res ;
  For each index in each row of the pattern:
    Ciphertext temp;
    temp = Homomorphic XNOR (encrypted genome bit, encrypted pattern bit)
    res = Homomorphic AND (res, temp)

result_array[index of genome, row of genome] = res
```

It then transposes resultant matrix to be of dimensions  $4 \times \langle \text{genome length} - \text{pattern length} + 1 \rangle$ , where each index contains a 1 if the pattern begins at that index or a 0 if the pattern does not begin at that index, respective for the one-hot encoding of each nucleotide.



An example is shown below, where the substring “AA” is searched for in the genome “AATT”. In the function, the operation occurs entirely on encrypted data; the plaintext version is shown to clarify the logic of the function.



The resultant representation is stored to allow for the return of the index at which the pattern occurs, logic which is used in homolog search operation.

#### 5.2.2.2.1 Encrypted T/F is pattern found

This is a helper function for Encrypted Substring Search.

This parallelized function takes in the result of the Substring String function and performs a homomorphic AND operation on the four rows of each index. It then returns the homomorphic OR on the result of each AND operation. The pseudocode is show below:

```

Ciphertext res;
res = Homomorphic AND (substring_search_result[0][0] , substring_search_result[1][0])
res = Homomorphic AND (res , substring_search_result[2][0])
res = Homomorphic AND (res , substring_search_result[3][0])

For i in range(1, genome_len – patten_len +1)
  Ciphertext temp;
  temp = Homomorphic AND (substring_search_result[0][i], substring_search_result[1][i])
  temp = Homomorphic AND (temp, substring_search_result[2][i])
  temp = Homomorphic AND (temp, substring_search_result[3][i])

  res = homomorphic OR (res, temp)

return res

```

### 5.2.2.3 Percent Match

This function takes in Alice's encrypted genome and a pattern (which can be another genome) and returns the percent match between the two. It also takes in an offset, which allows a user to check the percent match between a nucleotide sequence and a starting position in the genome.

It take in the result of Raw Match, and performs the following operation:

```
Vector result_vector;
For i in range(1, genome_len - patten_len + 1)
  Ciphertext res;
  res = Homomorphic AND (substring_search_result[0][i],
  substring_search_result[1][i])
  res = Homomorphic AND (res, substring_search_result[2][i])
  res = Homomorphic AND (res, substring_search_result[3][i])

  result.push_back(res)

return result_vector
```

This function is fully parallelized, therefore the order in which each index appears in the final vector is not constant. The decryption to find the percent match only adds up the number of matchers, thus the order of the result vector does not matter.

#### 5.2.2.3.1 Raw Match

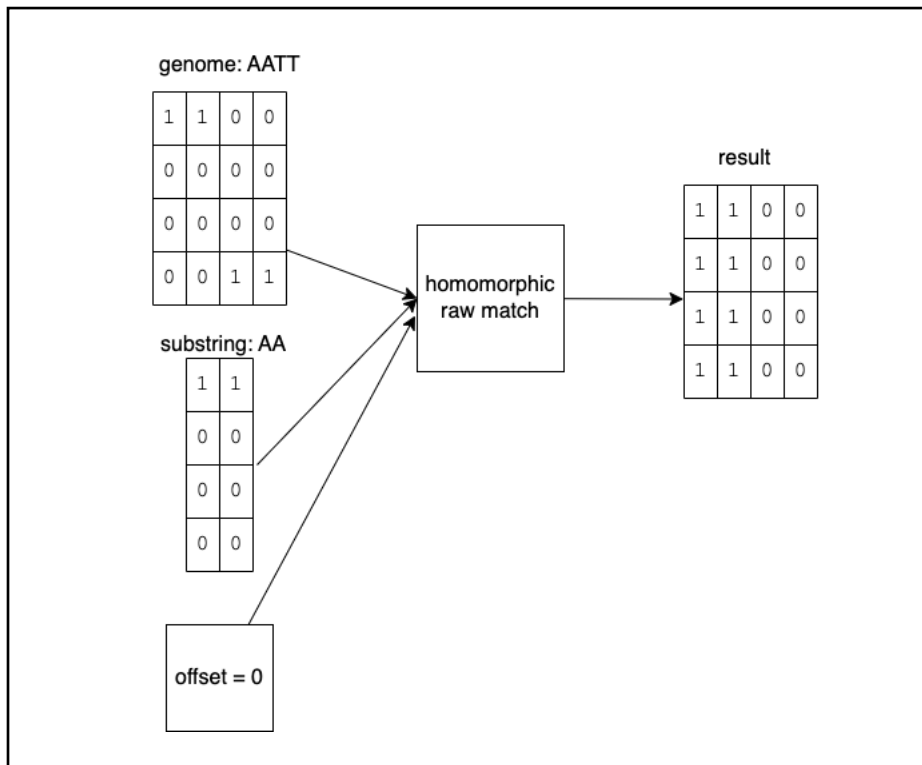
This function is used as a helper for Percent Match, which finds the percent similarity between two genomes, or between a genome and a pattern. An offset can be defined as the starting index at which to begin the match (for example, if the first X nucleotides of the genome should be ignored in the getting the Percent Match value)

This allocates a matrix of size  $4 \times \langle \text{genome\_len} - \text{pattern\_len} + 1 - \text{offset} \rangle$  matrix on the heap. The parallelized function then performs a match search operation as follows:

```
For each index in each row the genome:
  For each index in each row of the pattern:
    Ciphertext res;
    res = Homomorphic XNOR (encrypted genome bit, encrypted pattern
    bit)
    result_array[index of genome - offset, row of genome] = res

return result_array
```

An example of the operation is shown below; note that while in this function all values are encrypted, the plaintext is shown to explain the operation.



### 5.2.2.1 Homolog Search

Homolog search takes in a pattern with wildcards and returns an encryption of if the pattern was found. During decryption, the result of Homolog Search is used to return the actual value in the wildcard, thus simulating the presence of mutations in an allele in the genome.

X is used to indicate a wildcard. For example, in the genome `aaaaggcgaagtcg`, the pattern `agXcg` has homologs `aggcg` and `agtcg`.

#### 5.2.2.1.1 Encrypted Substring Search for Homolog

Encrypted Substring Search for Homolog operates similarly as Encrypted Substring Search for normal patterns, except that it will skip over the wildcard characters in the pattern as it does its comparison as shown below:

```

For each index in each row the genome:
  Ciphertext res ;
  For each index in each row of the pattern with wildcards:
    If found wildcard:
      skip
    Ciphertext temp;
    temp = Homomorphic XNOR (encrypted genome bit, encrypted pattern bit)
    res = Homomorphic AND (res, temp)

result_array[index of genome, row of genome] = res

```

#### 5.2.2.1.2 Encrypted Percent Match for Homolog

This function takes in the result of Homolog Substring Search and performs the same operations as the Percent Match for non-wildcard patterns.

### 5.3 Re-encryption protocol

The untrusted environment will output a tar file containing the encrypted results of Bob's queries. This tar file will be uploaded to the trusted environment, where it will be decrypted into plaintext, encrypted into a BFV, a FHE scheme that supports PKE, and finally using proxy re-encryption and Bob's public key will be re-encrypted such that Bob may decrypt with his secret key.

Because this operation would allow Alice to see the plaintext result of Bob's query, the re-encryption protocol should occur in an HSM that Alice does not have access to in order to reduce the overall trust of the system.

### 5.4 Decryption Operations

Once the result of each ciphertext is output from the Untrusted Environment, it must be decrypted.

#### 5.4.1 Decryption for Substring Search

Decrypting substring search requires the secret key and the result of the query, which is a single ciphertext. The ciphertext is decrypted into a 1 if the substring was found, and 0 otherwise.

#### 5.4.2 Decryption for Percent Match

The decryption operation will take in the ciphertext vector result from Percent Match, which is of dimensions  $1 \times \langle \text{genome\_len} - \text{pattern\_len} + 1 \rangle$ . The decryption will occur as follows:

```
int total = 0;
For each ciphertext in encrypted vector:
    Plaintext result = decrypt(ciphertext)
    Total += result

Return total / genome_len
```

#### 5.4.3 Decryption for homolog search

The ciphertext output of homolog search is a ciphertext vector of dimensions  $1 \times \langle \text{genome\_len} - \text{pattern\_len} + 1 - \text{num\_wildcards} \rangle$ . Decryption occurs as follows:

```
Vector<char*> result;

For i in range (0, ciphertext_vector.size() ):
    Plaintext homolog = decrypt(ciphertext[i])
    If (result):
        For j in range (0, wildcard_pattern.size()):
            a = genome_ciphertext[0][i+j];
            c = genome_ciphertext[1][i+j];
            g = genome_ciphertext[2][i+j];
            t = genome_ciphertext[3][i+j];

            if (a): homolog[j] = 'a'
            if (c): homolog[j] = 'c'
            if (g): homolog[j] = 'g'
            if (t): homolog[j] = 't'

        result.push_back(homolog)

for homolog in result:
    print (homolog)
```

## 6 Underlying Algorithms

### 6.1 OpenFHE

OpenFHE is an open-source lattice-based fully homomorphic encryption library [1]. OpenFHE wraps other open-source fully homomorphic encryption libraires including PALISADE, HELib, HEAAN, and provides an implementation of TFHE (The Fast Homomorphic Encryption over the Torus) library [22]. Thus, OpenFHE supports the following fully homomorphic encryption schemes: CKKS, BGV, BFV,DM, and CGGI.

This section describes the underlying algorithms developed by CGGI and implemented into OpenFHE.

#### 6.1.1 BinaryFHE and LWE (Learning With Errors)

The BinFHE library of OpenFHE, also called Boolean FHE is the basis for the encryption of the genome in this project. This library implements the CGGI encryption scheme—this is the same scheme implemented by TFHE.

BooleanFHE uses LWE ciphertexts, which means LWE is used to encrypt a message. The secret key in LWE is a short binary key in which each bit is generated randomly. The ciphertext is composed of  $k$  elements, where  $k - 1$  is the length of the secret key.

As described by Chillotti et al in their original paper[21] [22], and further explained on Chillotti's blog for Zama.ai [20], the first  $k - 1$  elements of the ciphertext are randomly chosen, and the last  $k^{\text{th}}$  bit  $B$  is the summation of each random element multiplied by each element of the secret key plus a gaussian error term, plus the message. Decryption occurs by multiplying the vector of the random elements by the vector of the secret key and subtracting this result to obtain the message plus the gaussian error [20].

Where  $p$  is the plaintext modulus and  $q$  is the ciphertext modulus, are two positive integers that are powers of 2, otherwise, implement a rounding operation in decryption.  $p \leq q$ , define the scaling factor  $\Delta = q/p$ . Message  $M \in \mathcal{R}_p$ .

The generalized LWE ciphertext encrypting  $M$  under secret key  $\vec{S} = (S_0, \dots, S_{k-1}) \in \mathcal{R}^k$  is a tuple:

$$(A_0, \dots, A_{k-1}, B) \in GLWE_{\vec{S}, \sigma}(\Delta M) \subseteq \mathcal{R}_q^{k+1}$$

where  $A_i$  for  $i \in [0, k-1]$  is sampled uniformly randomly from  $\mathcal{R}_q$

$$\text{and } B = \sum_{i=0}^{k-1} A_i \cdot S_i + \Delta M + E \in \mathcal{R}_q$$

and  $E \in \mathcal{R}_q$

The decryption protocol is canonical:

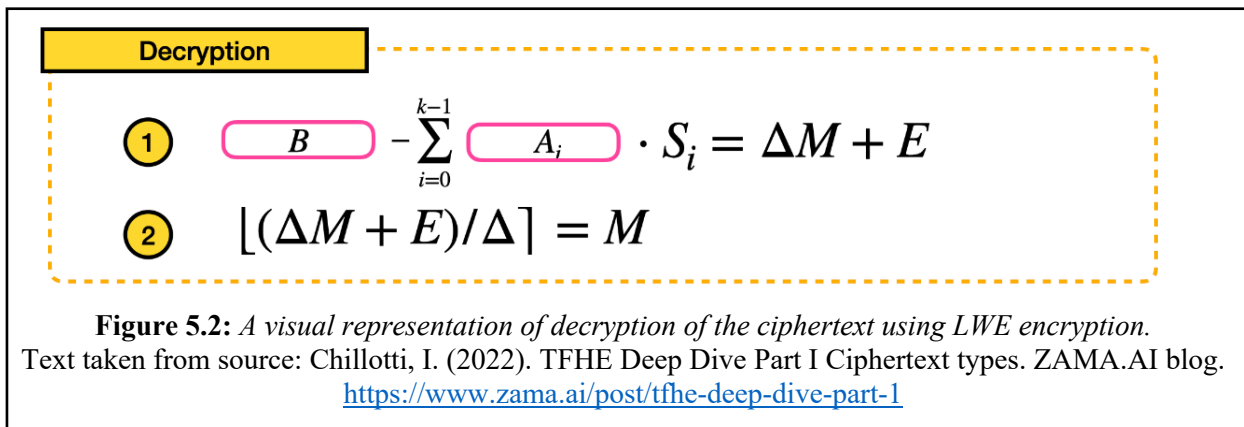
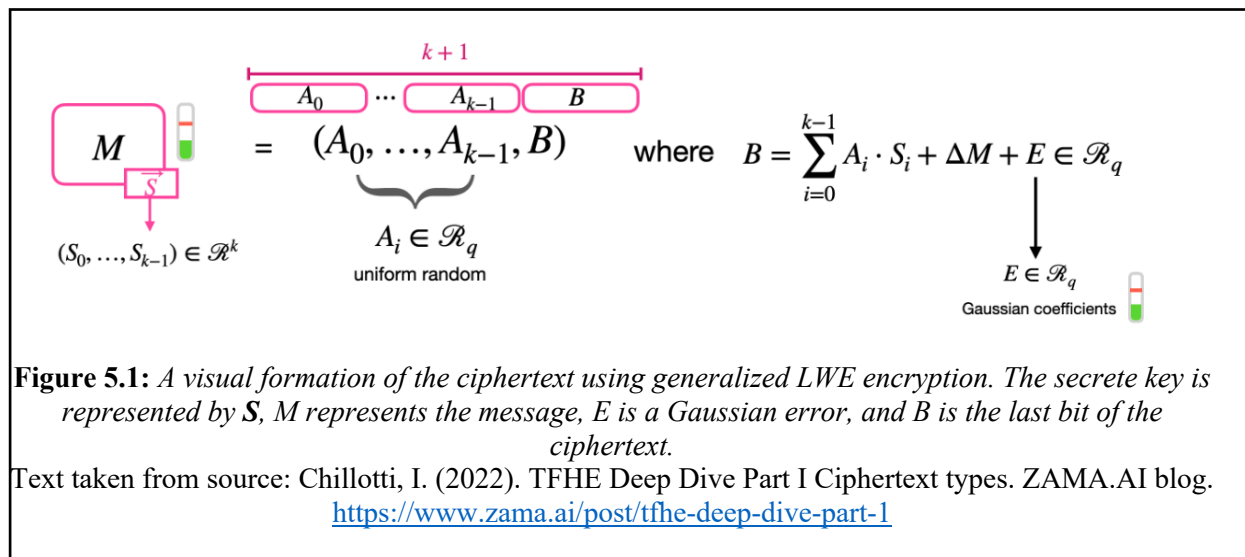
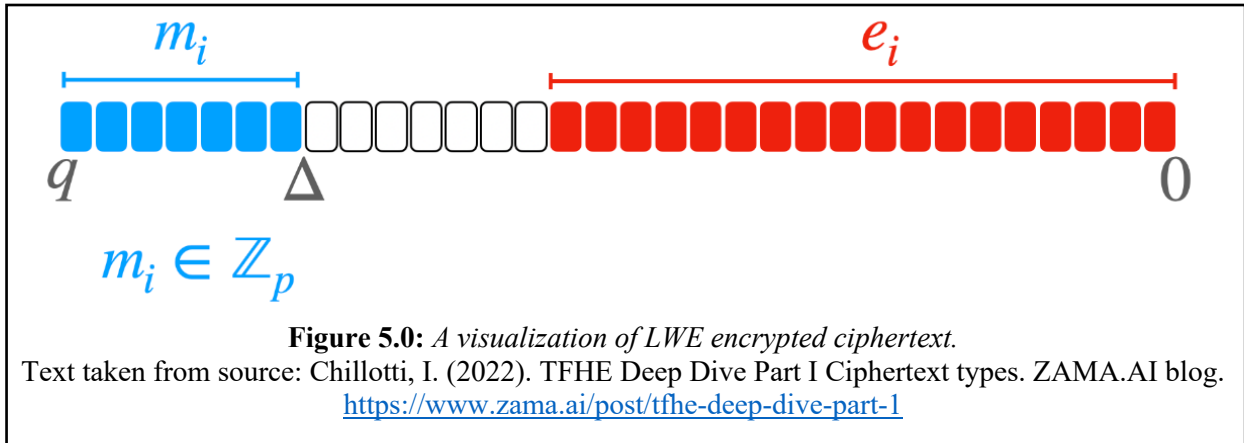
$$B - \sum_{i=0}^{k-1} A_i \cdot S_i = \Delta M + E \in \mathcal{R}_q$$

$$M = \lfloor \Delta M + E \rfloor / \Delta$$

Text taken from source: Chillotti, I. (2022). TFHE Deep Dive Part I Ciphertext types. ZAMA.AI blog. <https://www.zama.ai/post/tfhe-deep-dive-part-1>

Message  $M$  is stored in the MSB of  $\Delta M + E$ , and the error  $E$  is stored in the LSB. The following images show a visualization of a generalization of LWE encryption and decryption protocol. When the  $|E| < \Delta/2$ , in other words every coefficient  $e_i$  of  $E$  is  $|e_i| < \Delta/2$  to allow for correct decryption.

Figures 5.0, 5.1, and 5.2 below from Chillotti's blog post shows a visualization of the message  $m$ , the scaling factor  $\Delta$ , and the gaussian error  $e$  for (non-generalized) LWE. The visualization highlights that as homomorphic operations are applied to encrypted  $m$ , the gaussian error (noise) grows. This noise is reduced in bootstrapping operations.



For LWE encryption, the scheme used to encrypt the genome, instantiate  $k = n \in \mathbb{Z}$  and  $N = 1$ . When  $N = 1$ , where  $A_i$  for  $i \in [0, k-1]$  is chosen uniformly at random from  $\mathbb{Z}_q$ .

The LWE ciphertext encrypting  $m$  under secret key  $\vec{s} = (s_0, \dots, s_{n-1}) \in \{0,1\}^n$  is a tuple:

$$(a_0, \dots, a_{n-1}, b) \in LWE_{\vec{s}, \sigma}(\Delta M) \subseteq \mathbb{Z}_q^{n+1}$$

where  $a_i$  for  $i \in [0, n-1]$  is sampled uniformly randomly from  $\mathbb{Z}_q$

$$\text{and } b = \sum_{i=0}^{n-1} a_i \cdot s_i + \Delta m + e \in \mathbb{Z}_q$$

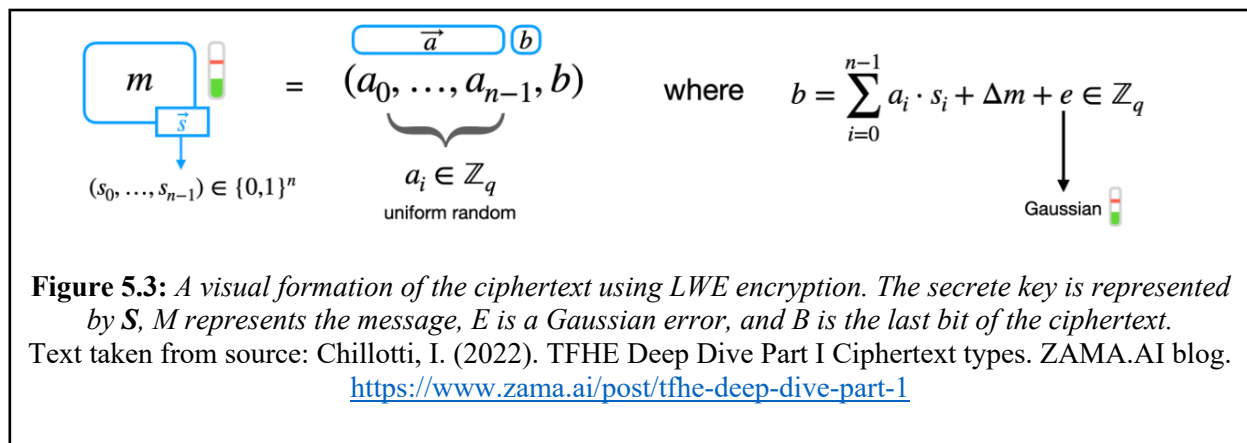
and  $e \in \mathbb{Z}_q$

Text taken from source: Chillotti, I. (2022). TFHE Deep Dive Part I Ciphertext types. ZAMA.AI blog.

<https://www.zama.ai/post/tfhe-deep-dive-part-1>

And decryption is canonical.

Figure 5.3 below shows a visualization of the LWE scheme.



### 6.1.2 Invoked CryptoContext functions

The invoked homomorphic encryption functions via the LWE Binary FHE library are XNOR and AND.

XNOR is used for equality checking. To illustrate,  $1 \text{ XNOR } 1 = 1$ ,  $0 \text{ XNOR } 0 = 0$ ,  $1 \text{ XNOR } 0 = 0$ ,  $0 \text{ XNOR } 1 = 0$ .

### 6.1.3 Chosen Security Parameters

The chosen security parameters on the LWE crypto context are the STD128 parameters, detailed in tables below. The specific scheme used is GINX, an implementation of the Chillotti-Gama-Georgieva-Izabachene (CGGI) FHE scheme [21]. This scheme is used to evaluate Boolean circuits [22]. CGGI is specifically used as the bootstrapping method.



**Table 5.1: STD128 security params**

Source: OpenFHE source code <https://github.com/openfheorg/openfhe-development/blob/main/src/binfhe/lib/binfhecontext.cpp>

numberBits	cycleOrder	latticeParam	mod	modKS	stdDev	baseKS	gadgetBase	baseRk
27	2048	512	1024	$1 \ll 14$	STD_DEV	$1 \ll 7$	$1 \ll 7$	32

**Table 5.2: Security Param Details**

Source: OpenFHE source code <https://github.com/openfheorg/openfhe-development/blob/main/src/binfhe/lib/binfhecontext.cpp>

variable	description
numberBits	Used for intermediate prime and RLWE (Ring GSW) used in bootstrapping
cycleOrder	Used for intermediate prime and RLWE (Ring GSW) used in bootstrapping
latticeParam	LWE crypto parameter
mod	Modulus for additive LWE
modKS	Modulus for Key Switching
stdDev	Preset to 2.19
baseKS	base for key switching
gadgetBase	gadget base used in the bootstrapping, used for Ring GSW + LWE
baseRk	base for the refreshing key, used for Ring GSW + LWE

### 6.1.4 Private, refreshing, and re-encryption keys

The private key is generated by OpenFHE’s LWE Binary FHE context object. The private key serves as a symmetric key.

The refreshing key is also generated by OpenFHE’s LWE Binary FHE context object. The rotation key is used to bootstrap the ciphertext.

The re-encryption key is generated by creating a new PKE key pair in the trusted environment, using a protocol that takes as input the new private key and Bob’s public key. The new re-encryption key is used proxy-re-encrypt a ciphertext encrypted by the generated private key to a ciphertext that can be decrypted with Bob’s private key. Re-encryption is described in greater detail in section 4.1.3.1.3.

## 6.2 Proxy Re-Encryption and Key Switching

Craig Gentry, in his PhD thesis, identified that the noise-reduction operation of bootstrapping could be used to transform a ciphertext asymmetrically encrypted by Alice into one that can be decrypted by Bob [40]. In his paper describing FHE using ideal lattices, which is the underlying construction of the RLWE FHE scheme, he defines a Recrypt function, the existence of which implies the existence of a one-way proxy re-encryption scheme [41].

Gentry defines the re-encryption operation by switching an asymmetrically encrypted ciphertext by Alice to one that can be decrypted with Bob’s private key [41] . Gentry’s Recrypt function is formally described below.


$\text{Recrypt}_{\mathcal{E}}(\text{pk}_2, D_{\mathcal{E}}, \langle \overline{\text{sk}_{1j}} \rangle, \psi_1).$

Set  $\overline{\psi_{1j}} \xleftarrow{R} \text{Encrypt}_{\mathcal{E}}(\text{pk}_2, \psi_{1j})$

Output  $\psi_2 \leftarrow \text{Evaluate}_{\mathcal{E}}(\text{pk}_2, D_{\mathcal{E}}, \langle \langle \overline{\text{sk}_{1j}} \rangle, \langle \overline{\psi_{1j}} \rangle \rangle)$

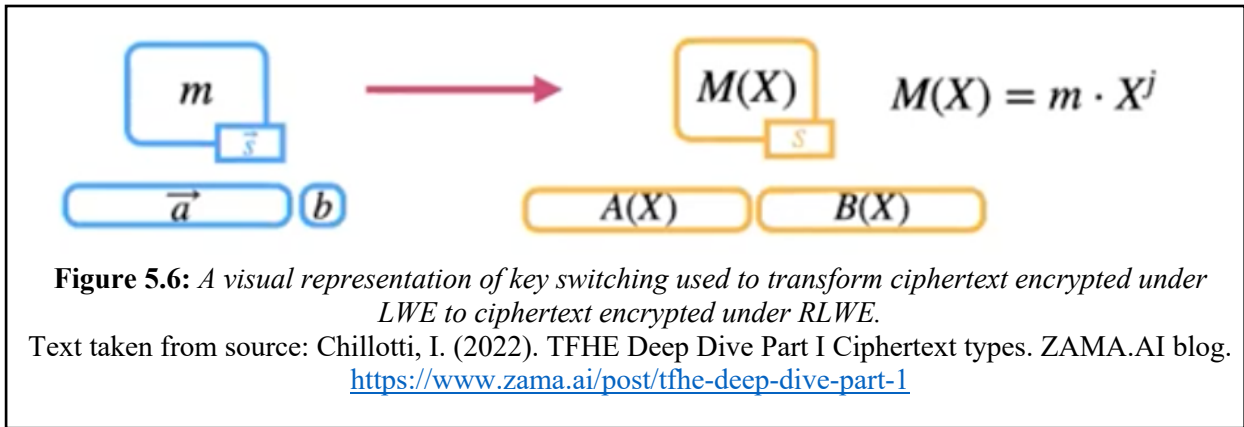
**Figure 5.4:** Gentry’s Recrypt function.  
Text taken from source: Gentry 2009 [41]

In the context of this project, the key switching operation transforms a message encrypted under LWE. The “switching key” is a generated public key. Key switching takes as input a message encrypted with secret key  $\vec{S}$ , and outputs the same message encrypted by secret key  $\vec{S}'$ . Key switching dramatically increase the noise generated on the ciphertext as compared to the amount of noise generated by a homomorphic operation or by encryption. A visual representation of key switching is show below:



**Figure 5.5:** A visual representation of switching the ciphertext of message  $M$  encrypted by key  $\vec{S}$ , to the ciphertext of message  $M$  encrypted by the Switch Key, key  $\vec{S}'$ .  
Text taken from source: Chillotti, I. (2022). TFHE Deep Dive Part I Ciphertext types. ZAMA.AI blog. <https://www.zama.ai/post/tfhe-deep-dive-part-1>

Key switching is also used to transform a ciphertext encrypted under LWE, which allows for only symmetric encryption, to a ciphertext encrypted under Ring LWE, or RLWE, which encrypts a message asymmetrically to allow for utilization of public key encryption on top of the fully homomorphic algorithm. The figure below shows a visual representation of converting a message encrypted under LWE (in blue) to a message encrypted under RLWE (yellow).



Finally, key switching can be used to modify the parameters of a ciphertext.

In key switching, the “switching key” is a secret key generated by the CryptoContext. However, in Proxy Re-Encryption, the “re-encryption” is a protocol matching Gentry’s Recrypt function.

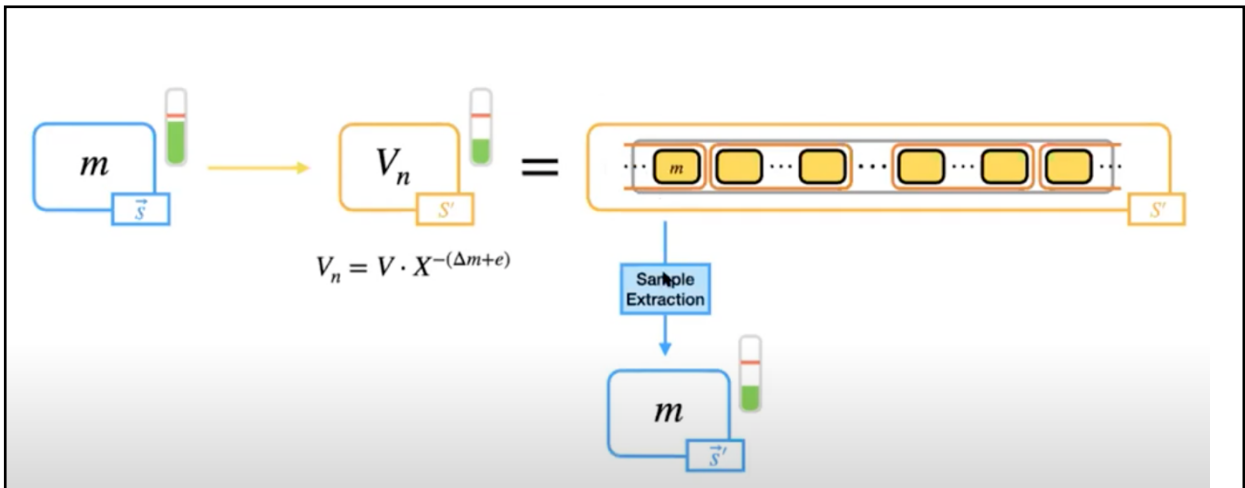
### 6.2.1 Noise reduction via bootstrapping

The input to the bootstrapping operation is an LWE ciphertext with noise, and the output is the same ciphertext with less noise to allow for correct decryption.

Bootstrapping occurs in many steps. The goal of bootstrapping is to isolate  $\Delta m + e$ .

Bootstrapping takes in as input the LWE ciphertext  $c = Enc(m, \vec{S})$  where  $m$  is the message and  $\vec{S}$  is the secret key. By redundantly sampling the LWE ciphertext into a vector, the LWE ciphertext  $c$  is converted into a Ring LWE ciphertext  $c'$  encrypted with new key  $\vec{S}'$ . A Blind Rotation operation occurs in which the most significant bits of  $c'$  are moved to the least significant position, flipping the ciphertext. Finally, a single coefficient (located at the most significant bit of the RLWE ciphertext) is extracted via a Sample Extraction protocol. This coefficient  $c''$  encodes  $m$ , but with less noise. Finally, a key switching operation converts  $c''$ , encrypted with  $\vec{S}'$ , back to ciphertext  $c$ , encrypted with  $\vec{S}$ .

This is visualized below, where blue represents an LWE ciphertext and yellow represents a RLWE ciphertext.



**Figure 5.7:** A visual representation of key switching used to transform ciphertext encrypted under LWE to ciphertext encrypted under RLWE.

Text taken from source: Chillotti, I. (2022). TFHE Deep Dive Part I Ciphertext types. ZAMA.AI blog. <https://www.zama.ai/post/tfhe-deep-dive-part-1>

## 7 Results and Discussion

Size of a serialized encrypted single bit under LWE encryption: 4 KB.

Time complexity to search, get percent similarity, and search with wildcard (homolog search), where  $n$  is the length of the genome and  $m$  is the length of the pattern:  $O(nm)$ .

Experimental results were run on two machines. The 8-CPU machine was a 2020 MacBook Air with Apple M1 chip and 8 GB of RAM running macOS Monterey. The 20-CPU machine had 34 GB RAM, Intel® Xeon® Silver 4210 CPU at 2.20 GHz, CPU MHz 2194.843, BogoMIPS 4389.68 running on a VMware hypervisor. Software for both machines identical and compiled for each environment.

The plaintext genome used was a plasmid of yeast bacterial (*S. cerevisiae* (budding yeast)), growth strain DH5alpha, that is resistant to Ampicillin [2].

The plasmid has been split into three test conditions, “small”, “medium”, and “large.” The small condition contained a segment of 16 nucleotides, the medium segment contained 3571 nucleotides, and the large continued the full plasmid which has 5574 nucleotides.

Encryption occurs bit by bit, parallelized such that each thread encrypts one bit and performs one write operation into a pre-allocated matrix. Therefore, runtime scales linearly with the size of the genome and improves as the number of available threads increases. We can approximate that encrypting one nucleotide takes approximately 103 ms per nucleotide on an 8-core machine, and approximately 76 ms per nucleotide on a 20-core machine. Therefore, significant hardware optimizations are necessary to encrypt and operate on an entire human genome. Borrowing from the ideas of modern machine learning, it is expected that modifying the software to run on GPUs is a necessary first step in bringing encrypted genome search to a runtime that can be viable in real-world scenarios.

Figures 7.1 and 7.2 below show a comparison of the runtime of different operations on machines of increasing CPU capacity. The machines running Apple Silicon consistently outperform the Linux machines, even when the Intel machines have greater CPU capacity. This performance can be attributed to the inclusion of four high performance CPU cores on the Apple Silicon Chips. We can therefore see that by parallelizing each operation, we can take advantage of both increasingly fast hardware, and traditional hardware that can handle a greater number of threads. Typically, each intel CPU can handle two threads per CPU, but true thread count varies depending on the background processes running on each machine.

The OpenML library was chosen due to its ability to adapt to both faster CPUs and a variable number of CPUs. The FlexFHE software exploits the ability of OpenML to split one central thread into multiple subthreads. OpenML calculates the maximum possible threads available on the machine that FlexFHE is running on, which is typically two threads per CPU. In the software, the matrix containing the one-hot encoded genome is maximally parallelized, meaning that each thread takes an  $i, j$  position, where  $i, j$  represent each row, column index in the matrix. Encryption, decryption, and binary operations (AND, OR, XOR, NOT) are all an operation handled by a single thread. The thread then write to a pre-allocated matrix indexed by the same  $i, j$  pair. Therefore, the relative position of the row, column index is irrelevant; each thread simply remembers its own  $i, j$  and single bit operation, and then performs a single

write operation into its own  $i, j$  in the new pre-allocated “result” matrix. Because each thread can operate and perform a write operation independently of the other threads, thread synchronization is not needed at all for encryption and decryption operations and for binary logic operations. For operations in which a search or search-like operation is requested, each thread will perform an AND aggregation of the XNOR’d (or an OR aggregation of the XOR’d) comparison between the substring (the allele or mutation) and the text (the genome).

Therefore, we expect to have the lowest runtime on operations that involve no aggregation—and indeed we see overall lowest runtime on encryption and decryption operations. As the substring increases in size linearly, we expect a linear increase in runtime across all machines; this too can be seen in figure 7.2, which shows the runtime of the initial encryption of the genome, the pattern match between a short substring and the long genome, a wildcard search which allows for identifying mutations between a wildcard string and the long genome, and a percent match operation which returns the percent of matching nucleotides in between the genome and a contiguous substring (in testing, the second genome utilized was of similar length to the original genome, but this is not a requirement of this operation; percent match works regardless of the relative sizes of the two genomes to be compared).

Figures 7.1 and 7.2 also show the performance of each machine, and show the performance of each machine across a long, medium length, and short genome segment. We can see that the runtime of all operations increases approximately linearly as the genome segment length increases. Since each operation occurs on a maximally-parallelized matrix, this is aligned with performance expectations.

Looking at figure 7.2, we see a consistent relationship between each machine and its relative runtime for each operation; the machine performance speed order is fairly consistent across all operations and all genome sizes. We see the fastest runtime is exhibited by the Apple (ARM) machine with 10 cores. Next fastest is the Apple Silicon M1 chip with 8 cores, 4 of which are high efficiency cores. Next is the Intel machine running Linux with 20 high efficiency cores. We see that the performance of the 10 core Apple Silicon machine and the 8 core Apple M1 chip machine tends to be similar across all operations and genome lengths. We then see the three Intel machines in a slightly slower consistent grouping. The 20 High Efficient Intel CPUs machine performs constantly faster than the remaining two machines without high efficiency cores, and for encryption operations the 20 HE CPUs machine performs similarly to the Apple 10 core and Apple M1 8 core machine. This leads us to understand that as hardware moves from traditional CPUs to high efficiency CPUs, software that is able to maximally parallelize will dramatically outperform traditional CPUs, regardless of the number of traditional CPUs available. For example, we see the Apple 8 Core machine which boasts 4 HE CPUs dramatically outperform the 20-core Intel machine without any HE cores.

Amongst the Intel machines (running Linux OS), the 20 core machine outperforms the 8-core machine, with the 20 HE core machine being the fastest of the three.

Interestingly, increasingly large RAM has minimal impact on the runtime of the FlexFHE system. We see the 8-core M1 chip with only 8 GB of RAM perform comparatively to the Apple 10-core machine with 64 GB of RAM. This is encouraging from a usability perspective, as machines that are RAM constrained can still operate the FlexFHE system effectively. For consumer electronics, it is often cheaper to buy better CPUs while increasing RAM space tends to be more expensive. Therefore, it is our hope that the parallelized system

provided by FlexFHE allows for organizations running in RAM constrained environments to access strong cryptographic security guarantees without needing to spend more money to access performance gains.

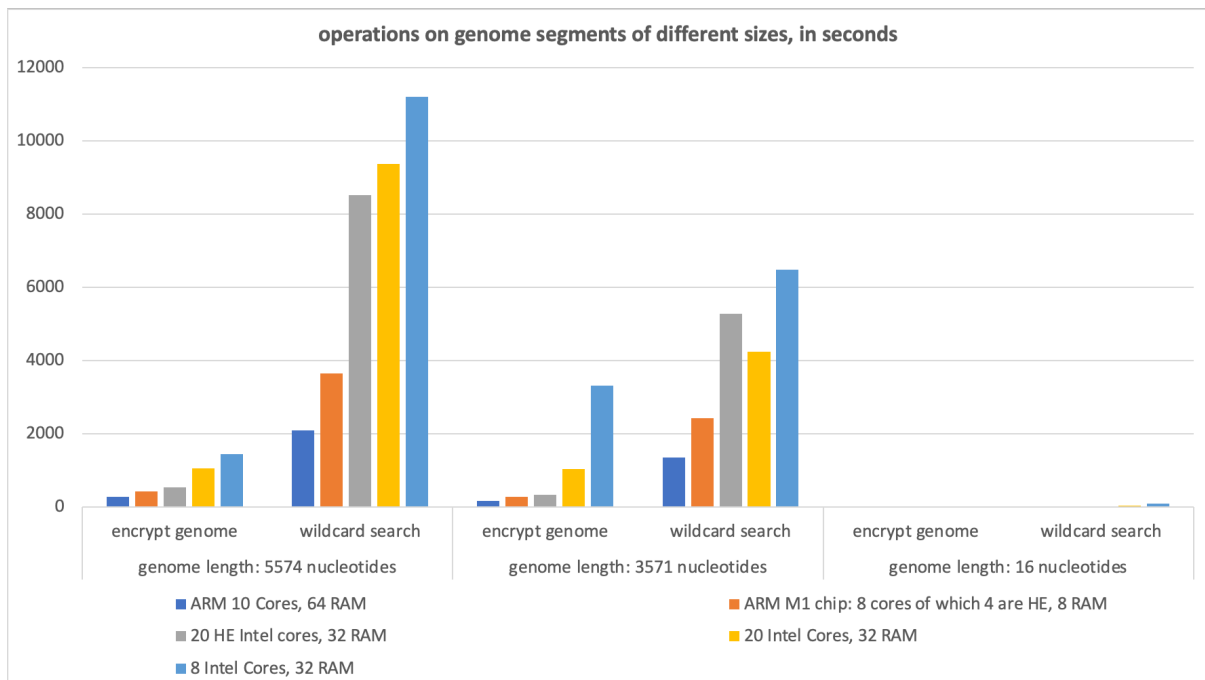


Figure 7.1 Runtime of different operations on genome segments of different sizes, compared for different machines, in seconds

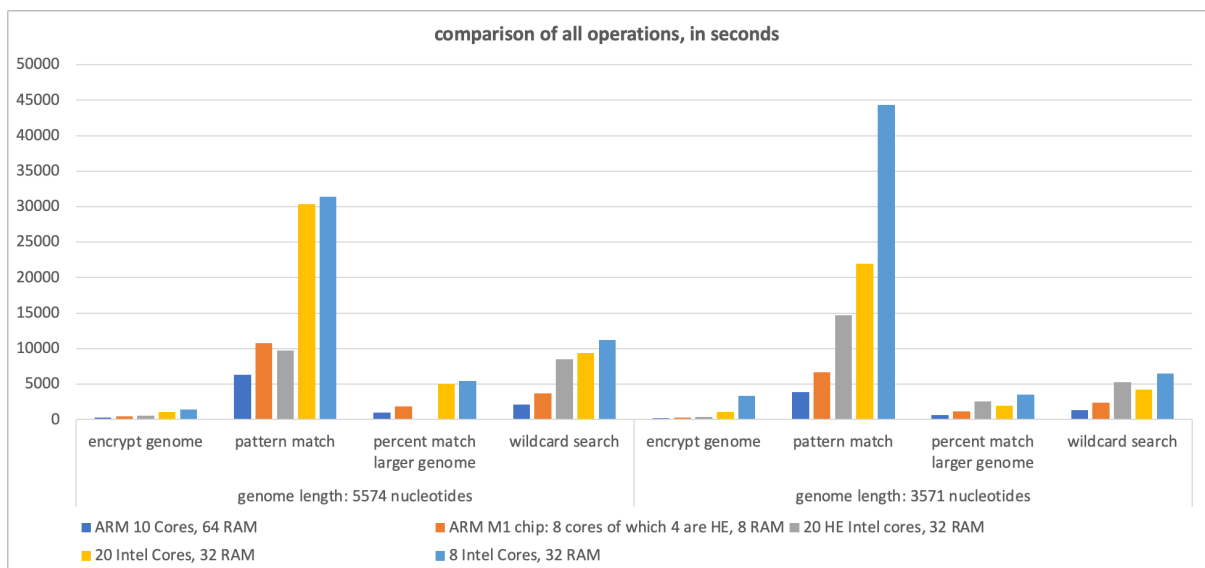


Figure 7.2 Comparison of all operations implemented on genomes of different sizes, across different machines, in seconds

## 8 Future Work

### 8.1 Implement IND-CCA1 LWE scheme

By definition of homomorphism, no scheme which has homomorphic properties may achieve indistinguishability under adaptive ciphertext attacks, or IND-CCA2 security [34].

However, it is theoretically possible to create a homomorphic scheme that is IND-CCA1 secure, or secure against non-adaptive chosen ciphertext attacks. Several generic constructions of IND-CCA1 FHE schemes have been suggested, but as of the time of this paper none have been implemented, and thus remains an open problem [34]. There are group-homomorphic schemes that have achieved IND-CCA1 security, such as the Cramer-Shoup-Lite scheme [34] [26]; this implies that IND-CCA1 fully homomorphic scheme exists and can be implemented. However, at the time of this paper, no such scheme has been implemented in software; and in the theoretical cryptography realm no such scheme has been concretely defined.

Future work should focus on first the construction and then the implementation in software of an IND-CCA1 FHE scheme.

### 8.2 Implement GPU threading

Parallelization should be extended using an OpenCL or CUDA API to allow for increased number of sub-threads executing each operation, resulting in decreased runtime. The changes that must occur in the software is semantic modification, i.e., transforming functions into labeled kernel functions, removal of the OpenMP API and appropriate replacement, and appropriate compilation with necessary hardware.

### 8.1 Modification for RNA variants

To modify this code to work for RNA variants such as mRNA and tRNA, the only function that needs to be modified is the “one-hot encode” function.

If you would like to implement the ability to encode only RNA, modify the condition in the switch statement identifying thymine (t) with one identifying uracil (u).

If you would like to implement the ability to encode both RNA and DNA, create a  $5 \times \langle \text{len of sequence} \rangle$  matrix (instead of a  $4 \times \langle \text{len of sequence} \rangle$  matrix). This can be accomplished by adding a condition to check for uracil in the switch statement and appending another vector to the vector <of ciphertext vectors> that is populated by function. The fifth row will be populated with the indexes, if any, of uracil. Because the rest of the code does a simple pattern match, no other changes are necessary for the software to function as expected.



## 9 Appendix

The complete software and ReadMe for this thesis is available open-source at:  
[https://github.com/lattias/Thesis\\_project](https://github.com/lattias/Thesis_project)

### A User Manual: Reproduce Performance and Timing Results

Follow the steps below to reproduce the performance metrics and timing data in this report:

#### A.1 Getting Started

##### 🔗 Installation steps for running the executable:

---

##### 1. install OpenFHE

[https://openfhe-development.readthedocs.io/en/latest/sphinx\\_rsts/intro/installation/installation.html](https://openfhe-development.readthedocs.io/en/latest/sphinx_rsts/intro/installation/installation.html)

The location to where you install OpenFHE, and the location to where you install this project, should share one parent folder. In other words, they should be "sibling nodes" in the file tree.

you need to copy the 'lib' folder of OpenFHE into the parent folder of this repository

For example, after installing OpenFHE and this repository your file tree should look as follows:

```
parent_directory 📄
- Data
- install_location_of_this_repository (this repo will install the files nested in this folder)
  - lib (copy this lib folder from the OpenFHE project into here)
  - thesis_central_code
    - build
    - flexFHE.cpp
    - serialized_data
    - pattern
      - zero
      - one
      - two
      - three
    - text
      - zero
      - one
      - two
      - three
  - install_location_of_openFHE (openFHE will install the files nested in this folder for you)
    - OpenFHEDevelopment
      - benchmark
      - build
      - cisd
      - configure
      - demodata
      - docker
      - docs
      - scripts
      - src
      - test
      - third-party
```

## 2. Clone this repository

```
git clone <this repository>
```

## 3. Put your data into the "data" folder

## 4. Execute the program

```
cd Thesis_project/thesis_central_code/build/  
./flexFHE
```

## Intallation steps for building the project:

---

### 1. install OpenFHE

```
brew install cmake  
brew install libomp
```

### install location of OpenFHE

```
mkdir /Users/username/documents/research/code/openfhe-development  
cd /Users/username/documents/research/code/openfhe-development
```

### clone the OpenFHE repository

```
git clone https://github.com/openfheorg/openfhe-development.git
```

### 2. create the directory where you want your project code to go into

```
mkdir /Users/username/documents/research/flexFHESource
```

### call cmake from OpenFHE inside your project directory

```
cd /Users/username/documents/research/flexFHESource  
cmake -DCMAKE_CROSSCOMPILING=1 -DRUN_HAVE_STD_REGEX=0 -DRUN_HAVE_POSIX_REGEX=0 /Users/username  
make
```

### 3. Build my project

🔗 Copy CMakeLists.User.txt from the root directory of the git repo to the folder for your project.

I want my code to go into

```
/Users/username/documents/research/flexFHESource
```

Rename CMakeLists.User.txt to CMakeLists.txt.

/Users/username/documents/research/flexFHESource/ should contain CMakeLists.txt

Update CMakeLists.txt to specify the name of the executable and the source code files. For example, include the following line

Now create the build folder, and execute make from from the build folder

```
cd /Users/username/documents/research/flexFHESource/  
mkdir build  
cd build  
cmake ..  
make  
./flexFHE
```

each time you modify the project start file, only run

```
make
```

## A 2. Executing timing and performance data only

To execute timing data only, and reproduce the results of this study in one pass, run the executable ./flexFHE:

```
cd build  
./flexFHE
```

## A.3 Usage Steps Overview

This section explains the overview of the pattern used to run FlexFHE, and an explanation of which functions run in which environment. For exact and simplified run steps, see appendix B.

1. Encrypt the genome. A private symmetric key will automatically be generated for you. A file titled “encrypted\_genome” will be populated in your local system. This file contains the encrypted genome.
  - a. ./trusted
    - i. You will need to populate the absolute path to your genome file
    - ii. Note that each pattern is a sequence of nucleotides such as aattt, a wildcard pattern such as aaXXXttt, or an absolute path to another genome (or any sequence of nucleotides) to perform a percent match on.



## B User Manual: Execute FlexFHE as a user, securely

Follow the steps in this sections to execute FlexFHE securely. Detailed instructions can be found at [https://github.com/lattias/Thesis\\_project](https://github.com/lattias/Thesis_project).

- Install OpenFHE
- Clone the Thesis\_project repository
- Create a new build folder and build the project
  - `rm -r build`
  - `mkdir build`
  - `cd build`
  - `cmake ..`
  - `make`
- In Alice's secure environment, encrypt the genome
  - `./trust`
- In Bob's untrusted environment, run calculations on the encrypted genome
  - `./untrust`
- In Alice's trusted environment, verify the calculations are correct by running:
  - `./trust_decr`
- In Bob's untrusted environment, create a public/private key pair for Bob.
  - `./bob_key`
- In Alice's secure environment, run the re-encryption protocol to re-encrypt the data
  - `./trust_reenc`
- In Bob's untrusted environment, decrypt the results with Bob's private key
  - `./untrust_decr`

### Where To Place Genome, Pattern, and Homolog Files:

- In **trust.cpp**, define the absolute paths to your local files.
  - set the variable **infilename** to the absolute path of your genome.
  - set the variable **pattern** to your pattern
    - The **pattern** is a sequence of nucleotides such as `aattt`
  - set the variable **homo (short for homolog)** to a wildcard pattern
    - A wildcard pattern contains the character `X` such as `aaXXXttt`
  - set the variable **percent\_file\_name** to the absolute path of another genome (or any sequence of nucleotides) to perform a percent match operation on.
    - The results of a percent match operation will tell you the percent of matching nucleotides between the original genome sequence and another genome sequence. The longer the continuous matching sequence and the higher the percent match between the two sequences, the greater likelihood of familial relationship between the two genomes.

## 10 References

- [1] Al Badawi, A., et al, (2022). OpenFHE: Open-Source Fully Homomorphic Encryption Library. Cryptology ePrint Archive, Paper 2022/915. <https://eprint.iacr.org/2022/915>
- [2] A Novel Recombinant DNA System for High Efficiency Affinity Purification of Proteins in *Saccharomyces cerevisiae*. Carrick BH, Hao L, Smaldino PJ, Engelke DR. G3 (Bethesda). 2015 Dec 29. pii: g3.115.025106. doi: 10.1534/g3.115.025106. 10.1534/g3.115.025106 PubMed 26715090
- [3] Annas, G., Glantz, L., and Roche, P. (1995). The Genetic Privacy Act and Commentary. Guidelines for Protecting Privacy of Information Stored in Genetic Data Banks, US Department of Energy. Boston University School of Public Health, Heath Law Department.
- [4] Asharov, G., Jain, A., and Wichs, D. (2011). Multiparty Computation with Low Communication, Computation and Interaction via Threshold FHE. <https://eprint.iacr.org/2011/613>
- [5] Asharov, G., Halevi, S., Lindell, Y., and Rabin, T. (2018). Privacy-Preserving Search of Similar Patients in Genomic Data. Cryptographic ePrint Archinve, Paper 2017/144 and PoPETS. <https://eprint.iacr.org/2017/144>
- [6] Aslett, L. J. M., Esperança, P. M. and Holmes, C. C.(2015), A review of homomorphic encryption and software tools for encrypted statistical machine learning. Technical report, University of Oxford.
- [7] Associated Press (2019). DNA from cigarette leads to Dakota Access arrest 3 years on. AP News. <https://apnews.com/article/sd-state-wire-us-news-north-dakota-riots-dakota-access-pipeline-abb444c2e6f14ca49a675e82d4b0d520>
- [8] Arshad, S. et al. (2021). Analysis of security and privacy challenges for DNA-genomics applications and databases. Journal of Biomedical Informatics. <https://doi.org/10.1016/j.jbi.2021.103815>.
- [9] Bellovin, S. Thinking Security: Stopping Next Year’s Hackers.
- [10] Bellovin, S. M., & Cheswick, W. R. (2007). Privacy-enhanced searches using encrypted bloom filters. <https://mice.cs.columbia.edu/getTechreport.php?techreportID=483>
- [11] Bohannon, P., Jakobsson, M., Srikwan, S. (2000). Cryptographic Approaches to Privacy in Forensic DNA Databases. In: Imai, H., Zheng, Y. (eds) Public Key Cryptography. PKC 2000. Lecture Notes in Computer Science, vol 1751. Springer, Berlin, Heidelberg. [https://doi.org/10.1007/978-3-540-46588-1\\_25](https://doi.org/10.1007/978-3-540-46588-1_25)
- [12] Bonomi L, Huang Y, Ohno-Machado L. (2020). Privacy challenges and research opportunities for genomic data sharing. National Genetics. doi: 10.1038/s41588-020-0651-0. <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7761157/>
- [13] Bonte, C. and Iliashenko, I. (2015). Homomorphic string search with constant multiplicative depth. <https://eprint.iacr.org/2020/931.pdf>
- [14] Cereal- A C++ library for serialization. <https://uscilab.github.io/cereal/>
- [15] Check Hayden, E. (2014). Technology: The \$1,000 genome. Nature 507, 294–295. <https://doi.org/10.1038/507294a>
- [16] Chadwick, J. Q., Copeland, K. C., Branam, D. E., Erb-Alvarez, J. A., Khan, S. I., Percy, M. T., Rogers, M. E., Saunkeah, B. R., Tryggstad, J. B., & Wharton, D. F. (2019). Genomic Research and American Indian Tribal Communities in Oklahoma: Learning From Past Research Misconduct and Building Future Trusting Partnerships.

- American journal of epidemiology, 188(7), 1206–1212.  
<https://doi.org/10.1093/aje/kwz062>
- [17] Chandra, R., Dagum, L., Kohr, D., Menon, R., Maydan, D., & McDonald, J. (2001). Parallel programming in OpenMP. Morgan kaufmann.
- [18] Chen, H., Chillotti, I., and Song, Y. (2019). Multi-key homomorphic encryption from TFHE. International Association for Cryptologic Research.  
<https://eprint.iacr.org/2019/116.pdf>
- [19] Cheon, J. H., Kim, D., Lee., J., and Song, Y. (2016). Lizard: Cut Off the Tail! A Practical Post-quantum Public-Key Encryption from LWE and LWR.  
<https://eprint.iacr.org/2016/1126.pdf>
- [20] Chillotti, I. (2022). TFHE Deep Dive- Part I- Ciphertext types. Zama.ai blog.  
<https://www.zama.ai/post/tfhe-deep-dive-part-1>
- [21] Chillotti , I., Gama, N., Georgieva, M., and Izabachene, M. (2016). Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In J. H. Cheon and T. Takagi, editors, Advances in Cryptology – ASIACRYPT 2016, pages 3–33, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg. 4, 7, 9
- [22] Chillotti , I., Gama, N., Georgieva, M., and Izabachene, M. (2016). TFHE: Fast Fully Homomorphic Encryption Library. <https://tfhe.github.io/tfhe/>
- [23] Chillotti, I., Ligier, D., Orfila, J., Tap, S. (2021). Improved Programmable Bootstrapping with Larger Precision and Efficient Arithmetic Circuits for TFHE.  
<https://eprint.iacr.org/2021/729.pdf>
- [24] Cho, H., Simmons, S., Kim, R., and Berger, B. (2020). Privacy-Preserving Biomedical Database Queries with Optimal Privacy-Utility Trade-Offs. Cell Systems.  
<https://doi.org/10.1016/j.cels.2020.03.006>
- [25] Clayton, E. W., Evans, B., J., Hazel, J., A., Rothstein, M. ,A. (2019). The law of genetic privacy: applications, implications, and limitations, Journal of Law and the Biosciences, Volume 6, Issue 1, Pages 1–36, <https://doi.org/10.1093/jlb/lasz007>
- [26] Cramer, R., Shoup, V. (1998). A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In: Krawczyk, H. (ed.) CRYPTO’98. LNCS, vol. 1462, pp. 13–25. Springer, Heidelberg
- [27] Cyphers, B. and Mackey, A. (2022). Fog Data Science puts our Fourth Amendment Rights up for sale. Electronic Frontier Foundation.  
<https://www.eff.org/deeplinks/2022/08/fog-data-science-puts-our-fourth-amendment-rights-sale>
- [28] Dahl, J. Y. and Saetnan, A. R. (2009). “It all happened so slowly” – on controlling function creep in forensic DNA databases. International Journal of Law, Crime and Justice. Volume 37, Issue 3. <https://doi.org/10.1016/j.ijlcj.2009.04.002>
- [29] DeDiS Laboratory, EPFL. Cothority network library. <https://github.com/dedis/onet> (2021).
- [30] Ducas, L., and Micciancio, D. (2015) FHEW: Bootstrapping homomorphic encryption in less than a second. In Eurocrypt, pages 617–640.
- [31] Dupont, C., Armant, D. R., & Brenner, C. A. (2009). Epigenetics: definition, mechanisms and clinical perspective. Seminars in reproductive medicine, 27(5), 351–357. <https://doi.org/10.1055/s-0029-1237423>
- [32] Eidelman, V. (2018). Why the Golden State Killer investigation is cause for concern. ACLU. <https://www.aclu.org/news/privacy-technology/why-golden-state-killer-investigation-cause>
- [33] Erabelli, S. (2020). pyFHE - A Python Library for Fully Homomorphic Encryption. Massachusetts Institute of Technology.



- <https://dspace.mit.edu/bitstream/handle/1721.1/129204/1227275316-MIT.pdf?sequence=1&isAllowed=y>
- [34] Fauzi, P., Hovd, M. N., and Raddum H. (2022). On the IND-CCA1 Security of FHE Schemes. *Cryptography*. <https://doi.org/10.3390/cryptography6010013>
- [35] Ferguson, D. (2020). Bentaas case: Judge says DNA evidence from trash pull can be used in murder trial. *Sioux Falls Argus Leader*. <https://www.argusleader.com/story/news/crime/2020/03/17/theresa-bentaas-judge-issues-decision-dna-evidence-trash-pull-can-used-court/5062205002/>
- [36] Fisch, B., Vo, B., Krell, F., Kumarasubramanian, A., Kolesnikov, V., Malkin, T., and Bellovin, S. (2015). "Malicious-Client Security in Blind Seer" In Proceedings of the 36th IEEE Symposium on Security & Privacy (S&P).
- [37] Focardi, R., and Luccio, F. (2021). A Formally Verified Configuration for Hardware Security Modules in the Cloud. In Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21). Association for Computing Machinery, New York, NY, USA, 412–428. <https://doi.org/10.1145/3460120.3484785>
- [38] Ge, J., and Budowle, B. (2020). Forensic investigation approaches of searching relatives in DNA databases. *Journal of Forensic Sciences*. <https://doi.org/10.1111/1556-4029.14615>
- [39] Genetic Information Nondiscrimination Act of 2008. (2008). United States 100<sup>th</sup> congress.
- [40] Gentry, C. (2009). A Fully Homomorphic Encryption Scheme. Stanford University. <https://crypto.stanford.edu/craig/craig-thesis.pdf>
- [41] Gentry, C. (2009), Fully Homomorphic Encryption Using Ideal Lattices. Association for Computing Machinery <https://www.cs.cmu.edu/~odonnell/hits09/gentry-homomorphic-encryption.pdf>
- [42] Gentry, C., Sahai, A., and Waters, B. (2013). Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. <https://eprint.iacr.org/2013/340.pdf>
- [43] Goodluck, K. (2020). Indigenous data sovereignty shakes up research: in the COVID-19 era, tribal nations want research in service of their people. *High Country News*. <https://www.hcn.org/issues/52.11/indigenous-affairs-covid19-indigenous-data-sovereignty-shakes-up-research>
- [44] Grant, C. (2022). Police are using newborn genetic screening to search for suspects, threatening privacy and public health. *ACLU*. <https://www.aclu.org/news/privacy-technology/police-are-using-newborn-genetic-screening>
- [45] Guillén, M., Victoria Lareu, M., Pestoni, C., Salas, A., Carracedo, A. (2000). Ethical-legal problems of DNA databases in criminal investigation. *Journal of Medical Ethics*. <http://dx.doi.org/10.1136/jme.26.4.266>
- [46] Halevi, S., and Shoup, V. (2014). Algorithms in HElib. In J. A. Garay and R. Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of LNCS, pages 554–571.
- [47] Halevi, S. and Shoup, V. (2014). Bootstrapping for HElib. *IACR Cryptology ePrint Archive*. <http://eprint.iacr.org/2014/873>.
- [48] Harmon, A. Indian tribe wins its fights to limit research of its DNA. (2010). *The New York Times*. <https://www.nytimes.com/2010/04/22/us/22dna.html>
- [49] *Havasupai Tribe of Havasupai Reservation v Arizona Board of Regents*, 204 P3d 1063, November 28, 2008. (App 2008).
- [50] Heled, Y. and Vertinsky, L. (2020). Genetic Paparazzi: Beyond Genetic Privacy. *Ohio State Law Journal* 409 (2021), Georgia State University College of Law, Legal



Studies Research Paper No. 2022-04, Available at SSRN:

<https://ssrn.com/abstract=3559405>

- [51] Henn, B. Hon, L., Macpherson, J. M., Eriksson, N., Saxonov, S., Pe'er, I. (2021). Cryptic distant relatives are common in both isolated and cosmopolitan genomic samples. *PLoS ONE* 7, e34267.  
<https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0034267>
- [52] Hyka, D. et al. (2020). A Study of Pohlig-Hellman Cryptosystem Parameters. *International Journal of Scientific & Engineering Research*. ISSN 2229-5518.  
<https://www.ijser.org/researchpaper/A-STUDY-OF-POHLIG-HELLMAN-CRYPTOSYSTEM-PARAMETERS.pdf>
- [53] Jarecki, S., Jutla, C., Krawczyk, H., Rosu, M., Steiner, M. (2013). Outsourced Symmetric Private Information Retrieval. *Cryptology ePrint Archive*.  
<https://eprint.iacr.org/2013/720>.
- [54] Jouvenal, J. (2018). To find alleged Gold State Killer, investigators first found his great-great-great-grandparents. *The Washington Post*.  
[https://www.washingtonpost.com/local/public-safety/to-find-alleged-golden-state-killer-investigators-first-found-his-great-great-great-grandparents/2018/04/30/3c865fe7-dfcc-4a0e-b6b2-0bec548d501f\\_story.html](https://www.washingtonpost.com/local/public-safety/to-find-alleged-golden-state-killer-investigators-first-found-his-great-great-great-grandparents/2018/04/30/3c865fe7-dfcc-4a0e-b6b2-0bec548d501f_story.html)
- [55] Kahrobaei, D., Wood, A., and Najaran, K. (2020). Homomorphic Encryption for Machine Learning in Medicine and Bioinformatics. *ACM Comput. Surv.* ISSN 0360-0300. <https://eprints.whiterose.ac.uk/151333/7/main.pdf>
- [56] Kaminsky, M. (2018). Madonna loses fight to reclaim Tucac's letter, other 'highly personal items'. *Forbes.com*.  
<https://www.forbes.com/sites/michellefabio/2018/04/23/madonna-loses-fight-to-reclaim-tupacs-letter-other-highly-personal-items/?sh=4cb4862b38b0>
- [57] Kirchner, L. (2016). DNA Dragnet: In some cities, police go from Stop-and-Frisk to Stop-and-Spit. *ProPublica*. <https://www.propublica.org/article/dna-dragnet-in-some-cities-police-go-from-stop-and-frisk-to-stop-and-spit>
- [58] Krishna R, Kelleher K, Stahlberg E. Patient confidentiality in the research use of clinical medical databases. *Am J Public Health*. (2007) Apr;97(4):654-8. doi: 10.2105/AJPH.2006.090902. Epub 2007 Feb 28. PMID: 17329644; PMCID: PMC1829362.
- [59] Lauter, K., Naehrig, M., and Vaikuntanathan, V. (2011). Can Homomorphic Encryption be Practical? <https://eprint.iacr.org/2011/405.pdf>
- [60] Legrandin (2022). *PyCryptodome Documentation 3.15.0* [Computer Software].  
<https://pypi.org/project/pycryptodome/>
- [61] Littlewood, Bev et al. (1993). Towards Operational Measures of Computer Security. *Journal of Computer Security*.
- [62] Lyubashevsky, V., Peikert, C., Regev, O. (2010). On Ideal Lattices and Learning with Errors over Rings. In: Gilbert, H. (eds) *Advances in Cryptology – EUROCRYPT 2010*. EUROCRYPT 2010. *Lecture Notes in Computer Science*, vol 6110. Springer, Berlin, Heidelberg. [https://doi.org/10.1007/978-3-642-13190-5\\_1](https://doi.org/10.1007/978-3-642-13190-5_1)
- [63] Mahdi, M. et al(2021). Privacy-preserving string search on encrypted genomic data using a generalized suffix tree. <https://doi.org/10.1016/j.imu.2021.100525>
- [64] Malin, B. and Sweeney, L. (2000). Determining the identifiability of DNA database entries in *Proceedings of the AMIA Symposium 537* (American Medical Informatics Association, 2000).
- [65] Martins, G., Bhatia, S., Koutsoukos, X., Stouffer, K., Tang, C., & Candell, R. (2015). Technical Report: Towards a Systematic Threat Modeling Approach for Cyber-physical Systems. *Proceedings, 2015 Resilience Week (RSW) : Hilton*

- Philadelphia at Penn's Landing, Philadelphia, PA, 18-20 August, 2015. Resilience Week (2015 : Philadelphia, Pa.), 2015, 10.1109/RWEEK.2015.7287428.  
<https://doi.org/10.1109/RWEEK.2015.7287428>
- [66] Masters, O., Hunt, H., Steffinlongo, E., Crawford, J., & Bergamaschi, F. (2019). Towards a Homomorphic Machine Learning Big Data Pipeline for the Financial Services Sector. IACR Cryptol. ePrint Arch., 2019, 1113.
- [67] Microsoft SEAL (release 4.0). (2022). Microsoft Research, Redmond, WA.  
<https://github.com/Microsoft/SEAL>
- [68] Nassar, M., Malluhi, Q., Atallah, M., & Shikfa, A. (2019). "Securing Aggregate Queries for DNA Databases," in IEEE Transactions on Cloud Computing, vol. 7, no. 3, pp. 827-837, 1 July-Sept. 2019, doi: 10.1109/TCC.2017.2682860.
- [69] Naveed M., Ayday E., Clayton E. W., Fellay J., Gunter C. A., Hubaux J. P., et al. (2015). Privacy in the genomic era. ACM Comput. Surv. 48 1–44. 10.1145/2767007
- [70] OpenFHE. Appropriate error parameters for the noise flooding.  
<https://openfhe.discourse.group/t/appropriate-error-parameters-for-the-noise-flooding/95>
- [71] OpenFHE and Bruce, D. [https://github.com/openfheorg/openfhe-integer-examples/blob/master/src/strsearch\\_enc\\_1.cpp](https://github.com/openfheorg/openfhe-integer-examples/blob/master/src/strsearch_enc_1.cpp)
- [72] Paddock, S., Abedtash, H., Zummo, J. et al. (2019). Proof-of-concept study: Homomorphically encrypted data can support real-time learning in personalized cancer medicine. BMC Med Inform Decis Mak. <https://doi.org/10.1186/s12911-019-0983-9>
- [73] Pappas, V., Krell, F., Vo, B., Kolesnikov, V., Malkin, T., Choi, S. G., George, W., Keromytis, A. D., and Bellovin, S. M. (2014). "Blind Seer: A Scalable Private DBMS" In Proceedings of the 35th IEEE Symposium on Security & Privacy (S&P).
- [74] Petitcolas, F.A.P. (2011). Kerckhoffs' Principle. In: van Tilborg, H.C.A., Jajodia, S. (eds) Encyclopedia of Cryptography and Security. Springer, Boston, MA.  
[https://doi.org/10.1007/978-1-4419-5906-5\\_487](https://doi.org/10.1007/978-1-4419-5906-5_487)
- [75] Pitts, P. (2017). The privacy delusions of genetic testing. Forbes.  
<https://www.forbes.com/sites/realspin/2017/02/15/the-privacy-delusions-of-genetic-testing/?sh=151d452b1bba>
- [76] Pohlig, S., and Hellman, M. (1978). An Improved Algorithm for Computing Logarithms over GF(p) and its Cryptographic Significance.  
<https://ee.stanford.edu/~hellman/publications/28.pdf>
- [77] Pulivarti, R. (2022). How secure is your DNA? National Institute of Standards and Technology, Taking Measure, Just a standard blog.
- [78] pySEAL (2017). <https://github.com/Lab41/PySEAL/>
- [79] python-paillier (2016). <https://python-paillier.readthedocs.io/en/develop/>
- [80] Rahim, M., Zulkarnain, I, and Jaya, H. (2017). A review: search visualization with Knuth Morris Pratt algorithm. IOP Conf. Ser.: Mater. Sci. Eng. 237 012026. DOI 10.1088/1757-899X/237/1/012026. <https://iopscience.iop.org/article/10.1088/1757-899X/237/1/012026/meta>
- [81] Raisaro, J. L., Klann, J. G., Waghlikar, K. B., Estiri, H., Hubaux, J. P., & Murphy, S. N. (2018). Feasibility of Homomorphic Encryption for Sharing I2B2 Aggregate-Level Data in the Cloud. AMIA Joint Summits on Translational Science proceedings. AMIA Joint Summits on Translational Science, 176–185.
- [82] Ramirez, A. (2020). Police need a warrant to collect DNA we inevitably leave behind. ACLU. <https://www.aclu.org/news/privacy-technology/police-need-a-warrant-to-collect-dna-we-inevitably-leave-behind>

- [83] Raykova, M., Vo, B., Bellovin, S., and Malkin, T. (2009) Secure Anonymous Database Search". In the Cloud Computing Security Workshop (CCSW).  
[https://www.cs.columbia.edu/~smb/papers/sads\\_ccsw.pdf](https://www.cs.columbia.edu/~smb/papers/sads_ccsw.pdf)
- [84] Regev, O. (2005). On lattices, learning with errors, random linear codes, and cryptography. In Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing, STOC '05, pages 84–93, ACM
- [85] Reichmuth, A. M., Oberli, M. A., Jaklenec, A., Langer, R., & Blankschtein, D. (2016). mRNA vaccine delivery using lipid nanoparticles. *Therapeutic delivery*, 7(5), 319–334. <https://doi.org/10.4155/tde-2016-0006>
- [86] Rescorla, E. (2022). Privacy for genetic genealogy: happy goldfish bowl everyone. *Educated Guesswork*. Blog. <https://educatedguesswork.org/posts/dna-genealogy/>
- [87] Rubin, Paul. (2004). Indian Givers. *Phoenix New Times*.  
<https://www.phoenixnewtimes.com/news/indian-givers-6428347?showFullText=true>
- [88] Satha, S. et al (2018). A Review of Homomorphic Encryption Libraries for Secure Computation. <https://doi.org/10.48550/arXiv.1812.02428>
- [89] Sboner, A., Mu, X. J., Greenbaum, D., Auerbach, R. K., & Gerstein, M. B. (2011). The real cost of sequencing: higher than you think!. *Genome biology*, 12(8), 125.  
<https://doi.org/10.1186/gb-2011-12-8-125>
- [90] Schumacher GJ, Sawaya S, Nelson D, and Hansen AJ. (2020). Genetic Information Insecurity as State of the Art. *Front Bioeng Biotechnol*. doi: 10.3389/fbioe.2020.591980. PMID: 33381496; PMCID: PMC7768984.  
<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7768984/>
- [91] Shor, P. W. (1999). Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41 (1999), pp. 303-332.  
<https://doi.org/10.1137/S003614459834701>
- [92] Shruthi, G., et al. (2021). A General Purpose Transpiler for Fully Homomorphic Encryption. *Cryptology ePrint Archive*, Paper 2021/811.  
<https://eprint.iacr.org/2021/811>. <https://github.com/google/fully-homomorphic-encryption>
- [93] Simmons, G. (1983). A “weak” privacy protocol using the RSA crypto algorithm. *Cryptologia*, 7:2, 180-182. <https://doi.org/10.1080/0161-118391857900>
- [94] Stanley, J. (2016). Local police using and abusing DNA and other biometric technologies. *American Civil Liberties Union*. <https://www.aclu.org/news/privacy-technology/local-police-using-and-abusing-dna-and-other>
- [95] Stanley, J. and Steinhardt, B. (2003). Bigger monster, weaker chains: the growth of an American surveillance society. *American Civil Liberties Unions: Technology and Liberty Program*.
- [96] Sterling, R. L. (2011). Genetic Research among the Havasupai: A Cautionary Tale. *AMA Journal of Ethics: Health Law*. doi: 10.1001/virtualmentor.2011.13.2.hlaw1-1102. PMID: 2312185.
- [97] Sweeney L, Abu A & Winn J. (2013). Identifying Participants in the Personal Genome Project by Name (A Re-identification Experiment).  
<http://privacytools.seas.harvard.edu/files/privacytools/files/1021-1.pdf>
- [98] Tanner, A. (2017). *Our Bodies, Our Data: How Companies Make Billions Selling Our Medical Records*. Boston: Beacon Press. ISBN: 9780807033357
- [99] Vaidya J, et al. (2013). Identifying inference attacks against healthcare data repositories. *AMIA Jt Summits Transl Sci Proc*. 2013; 2013: 262–266.  
<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3845790/>

- [100] Vengadapurvaja, A. M., Nisha, G., Aarthy, R., Sasikaladevi, N. (2017). An Efficient Homomorphic Medical Image Encryption Algorithm For Cloud Storage Security. *Procedia Computer Science*. <https://doi.org/10.1016/j.procs.2017.09.150>.
- [101] Vijaya Kumar, A. et al. (2020). Secure Multiparty computation enabled E-Healthcare system with Homomorphic encryption. *IOP Conference Series: Materials Science and Engineering*. <https://doi.org/10.1088/1757-899x/981/2/022079>
- [102] Wang, X., Luo, T., Li, J. (2020). An efficient fully homomorphic encryption scheme for private information retrieval in the cloud. *International Journal of Pattern Recognition and Artificial Intelligence*. <https://doi.org/10.1142/S0218001420550083>
- [103] Wheeler, D., Srinivasan, M., Egholm, M. et al (2008). The complete genome of an individual by massively parallel DNA sequencing. *Nature* 452, 872–876. <https://doi.org/10.1038/nature06884>
- [104] Wohlwender, J., Huesmann, R., Heinemann, A., and Wiesmaier, A.. 2022. Cryptolib: Comparing and selecting cryptography libraries. In *Proceedings of the 2022 European Interdisciplinary Cybersecurity Conference (EICC '22)*. Association for Computing Machinery, New York, NY, USA, 6–11. <https://doi.org/10.1145/3528580.3528582>

## **11 Acknowledgements**

Very special thanks are given to Dr. Steven Bellovin of the Columbia Computer Science Department for his mentorship, guidance, support, and supervision of this thesis.