2020/2021

MINI PROJECT

Submitted in fulfillment of the requirements for the

COMMUNICATION AND MINI PROJECT GRADE FROM THE LEBANESE UNIVERSITY
FACULTY OF ENGINEERING – BRANCH III

Major: Electrical and Telecommunication Engineering

Prepared By:

Dani Zeineddine

Hussein Kazem

Secure Cloud Based Genetic Disease Detection Using Homomorphic Encryption Algorithm

Supervised by:

Dr. Abed Ellatif Samhat
Dr. Khalil Hariss

Defended on ---- March 2021 in front of the jury:

Prof. Youssef Harkouss	**President**
Dr. Abed Ellatif Samhat	**Member**
Dr. Khalil Hariss	**Member**

ACKNOWLEDGEMENTS

Immeasurable appreciation and deepest gratitude are extended to the following persons who, in one way or another, have contributed in making this project possible.

Dr. **Hassan Shreim**, the faculty's director, and Prof. **Youssef Harkouss** chief of the Electrical Engineering department, for their administration and leadership in addition to their efforts in keeping our faculty and its students ahead.

Dr. **Abed Ellatif SAMHAT** and Dr. **Khalil HARISS**, our supervisors, for their constant encouragement and inspiration to conduct efficient work on this project and obtain promising results. We are extremely grateful and indebted for their valuable guidance and orientation.

All the instructors of the faculty of engineering that we had the opportunity to be their students and for their continuous support across the years.

ABSTRACT

The cloud is a modern data storing technique that gives opportunities for outsourcing of storage and computation. In addition, the cloud offers a lot of benefits for customers such as scalability, non IT maintenance costs, disaster assistance, etc. A main problem of the cloud solution is users' privacy since the cloud is considered a non-trusted party. Homomorphic Encryption (HE) came as a new cryptographic research topic that allows processing and computations over encrypted data. Several big companies started to investigate in this new promising domain one of them is Microsoft.

Microsoft launched in 3/12/2018 Microsoft SEAL (Simple Arithmetic Encrypted Library), a homomorphic library written under C++ and available on Microsoft Windows, macOS and Linux.
The main goal of this mini-project is to understand the concept of HE, to do a state of art of some existing HE schemes such as RSA, BFV, CKKS, etc. Then we explore the Microsoft SEAL Library and design a secure cloud based application as a simple Proof of Concept (P.o.C.).

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

pk: public key
sk: secret key
HE: Homomorphic Encryption
BFV: Brakerski-Fan-Vercauteren
CKKS: Cheon-Kim-Kim-Song
AES: Advanced Encryption Standard
LWE: Learning with error
RLWE: Ring learning with error.
VCF: Variant Call Format

GENERAL INTRODUCTION

In cryptography, encryption is the process of hiding primitive data using encryption algorithms. During encryption, we convert the plaintext, which is the original representation of the information, into a ciphertext, which is the alternative encrypted form. Ideally, only authorized parties can decrypt a ciphertext back to plaintext and access the original information. Encrypting the data does not in itself prevent interference but denies access to the intelligible content to a would-be interceptor.
An encryption scheme requires the generation of one or two keys usually using pseudo-random generation algorithms. It remains possible to decrypt the ciphertext without knowledge of the key(s) but, for a well-designed encryption scheme, considerable computational resources and skills are required. On the other hand, authorized recipients can easily decrypt the message with the correct provided/available key, only if they are the intended recipients.
The process of encrypting and decrypting messages involves keys. The two main types of encryption in cryptographic systems are symmetric encryption and asymmetric encryption.
In symmetric-key schemes, the same key is used for both encryption and decryption. Communicating parties must have the same key in order to achieve secure communication.
In asymmetric encryption schemes, there exist 2 keys: the encryption public key is published for anyone to use and encrypt messages and, the recipient party's decryption key (private key) that enables messages to be read.

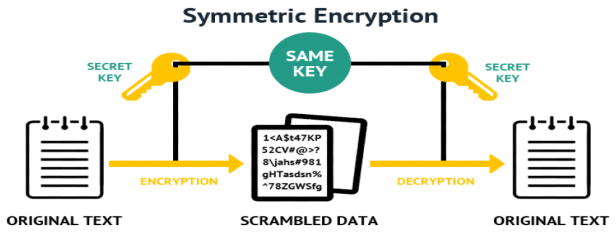

Figure 0.1 : Symmetric Encryption

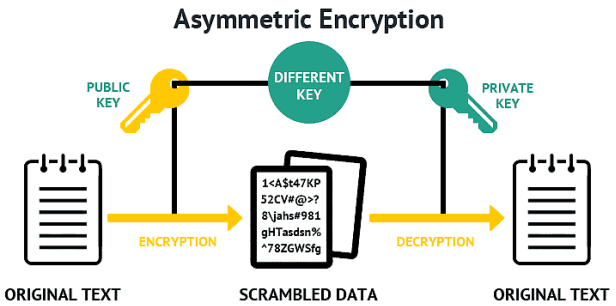

Figure 0.2: Asymmetric Encryption

CHAPTER 1: HOMOMORPHIC ENCRYPTION

1 Introduction

In today's world, sensitive information, such as personally identifiable information (PII) and financial data are handled by organizations. These information need to be encrypted both when being stored (data at rest) and when being transmitted (data in transit). Modern encryption algorithms are almost impossible to break with today's technology as they require too much processing power and time to break, at least until the coming of quantum computing. In other words, the process of breaking them is too costly and time-consuming to be feasible.

It is well known that the cloud is a data storing technique that allows the outsourcing of storage and computation. In addition, cloud offers flexibility and cost saving. In the concerned scenario, we have one main problem, user privacy is exposed to a third non trusted party which is the cloud. In some critical cases, users sometimes are obliged to reveal some of their secret parameters to the cloud, so that the latter is able to compute over their data after decryption.

In other words, the problem faced when encrypting data on the cloud is that sooner or later, it must be decrypted. This decrypted data is vulnerable to attackers(hackers). Cloud files can be kept cryptographically scrambled using a secret key, but as soon as a user wants to do anytime of operation something with those files, anything from editing a word document or querying a database of financial data, he is forced to decrypt the data and leave it vulnerable. An advancement in the science of cryptography known as Homomorphic encryption could change that.

Homomorphic encryption (HE) is a form of encryption allowing one to perform calculations on encrypted data without decrypting it first. The result of the computation is in an encrypted form, when decrypted the output is the same as if the operations had been performed on the unencrypted data.

With the use of HE, users are now able to store the data encrypted at the cloud side, the latter will compute over encrypted data without the need for decrypting it. Encrypted answers are shipped back to the users side where the decryption is possible.

An illustration of cloud computing using HE is given in Figure 1.1.

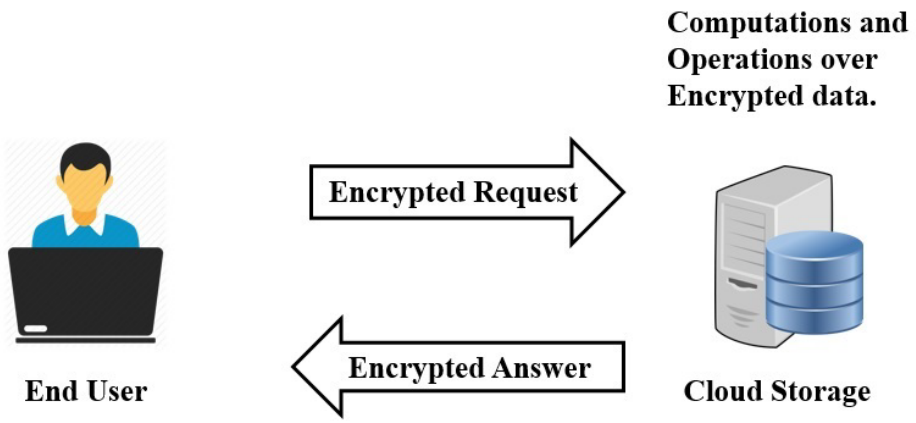

Fig. 1.1 Cloud Scenario Using HE

2 Homomorphic Concept and Properties

2.1 Asymmetric Schemes

As it was written in [1], a homomorphic public key or asymmetric encryption scheme ε is defined by 4 basic functions that are:

1. $KeyGen_{\varepsilon}$: Generates pair of keys: public key **pk** and secret key **sk**.
2. $Encrypt_{\varepsilon}\ (pk,\pi_i)$: Outputs a cipher-text ψ_i, where π_i is the plain-text.
3. $Decrypt_{\varepsilon}\ (sk,\psi_i)$: Outputs the plain-text π_i, where ψ_i is the cipher-text.
4. $Evaluate_{\varepsilon}$: This function takes as input the public key pk, a circuit C and a tuple of cipher-texts $\Psi = (\psi_1,\psi_2,\psi_2, ...,\psi_t)$. The evaluation function outputs a cipher-text ψ given by: $\psi = Evaluate\varepsilon\ (pk,C,\Psi)$.

Definition Homomorphic Encryption (HE) Scheme: The scheme ε is said to be homomorphic if:

$$\psi = Encrypt_{\varepsilon}\ (pk,C(\pi_1,\pi_2, ...,\pi_3)) \qquad (2.1)$$

To build a HE scheme we should satisfy the two following homomorphic properties:

1. Addition:

$$Enc_{\varepsilon}(pk, x_1) + Enc_{\varepsilon}(pk, x_2) = Enc_{\varepsilon}(pk, x_1 + x_2) \quad (2.2)$$

2. Multiplication:

$$Enc_{\varepsilon}(pk, x_1) \times Enc_{\varepsilon}(pk, x_2) = Enc_{\varepsilon}(pk, x_1 \times x_2) \quad (2.3)$$

Where x_1, x_2 are two plain-texts and $Enc\varepsilon\ (pk,x)$ is the encryption function.

2.2 Symmetric Schemes

The same approach discussed above for asymmetric schemes can be applied for symmetric ones. We need to build an encryption scheme Enc_K that satisfies the following relation:

$$Cipher_{Output} = Enc_K(Plain_{Output}) \quad (2.4)$$

If we have the same basic properties of addition and multiplication of asymmetric schemes given below where x_1 and x_2 are two symmetric plaintexts:

1. Addition:

$$Enc_K(x_1 + x_2) = Enc_K(x_1) + Enc_K(x_2) \quad (2.5)$$

2. Multiplication:

$$Enc_K(x_1 \times x_2) = Enc_K(x1) \times Enc_K(x2) \quad (2.6)$$

by applying the properties given in Equations (2.5, 2.6), we can demonstrate Equation (2.4) as follows:

$Cipher_{Output} = ((c_1 + c_2) \times (c_3 + c_4)) = (Enc_K(x_1) + Enc_K(x_2)) \times (Enc_K(x_3) + Enc_K(x_4)) = Enc_K(x_1 + x_2) \times Enc_K(x_3 + x_4) = Enc_K((x_1 + x_2) \times (x_3 + x_4)) = Enc_K(Plain_{Output})$.

3 Microsoft SEAL Library

Microsoft SEAL is an easy-to-use open-source (MIT licensed) homomorphic encryption library developed by the Cryptography and Privacy Research group at Microsoft. Microsoft SEAL is implemented in

a modern standard C++ and is simple to compile and run in many different environments.

Microsoft SEAL is powered by open-source homomorphic encryption technology. It allows the use of a set of encryption libraries allowing computations to be performed directly on encrypted data. This helps software engineers in building end-to-end encrypted data storage and computation services where the consumer never needs to share their key with the service provider which is a non trusted third party.

Today, Microsoft SEAL looks to reach its goal of making HE easy to use and available for *everyone* by providing a simple and convenient API with state-of-the-art performance. Microsoft SEAL comes premade with several detailed and thoroughly commented examples, demonstrating how the library can be used correctly and securely, and explaining any necessary background material.

In traditional cloud storage and computation solutions, customers need to trust the service provider to store and manage their data appropriately without exposing it untrusted to parties. Microsoft SEAL replaces this trust with state-of-the-art cryptography, allowing cloud services to provide both encrypted storage and computation capabilities, while still guaranteeing that their customer's data will never be exposed to anyone in unencrypted form as the customer never needs to share his private key with the cloud.

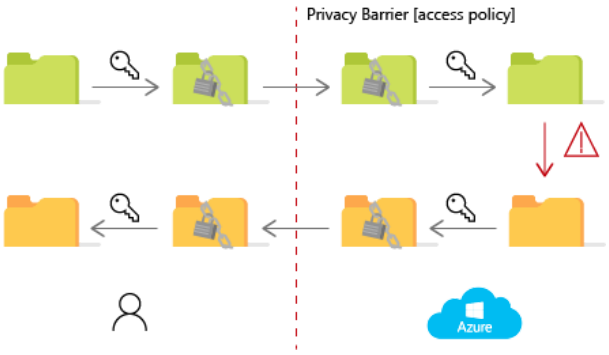

Figure 1.2:Traditional cloud Storage and Computation

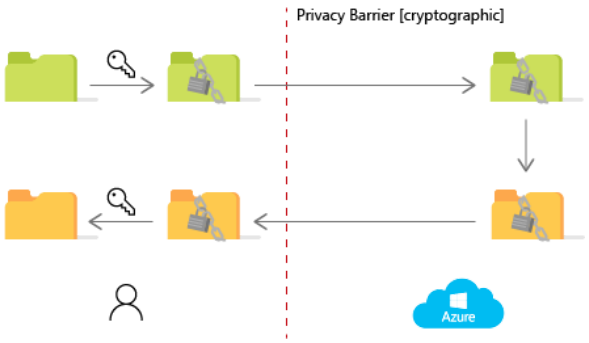

Figure 1.3:Microsoft SEAL CLoud Storage and Computation

Microsoft SEAL basic classes and functions:

EncryptionParameters class: There are three encryption parameters that are necessary to set (will be explained in the next section) :

- poly_modulus_degree (degree of polynomial modulus);
- coeff_modulus ([ciphertext] coefficient modulus);
- plain_modulus (plaintext modulus; only for the BFV scheme).

Example:

```
EncryptionParameters parms(scheme_type::bfv); //schemes will be discussed in the next section
size_t poly_modulus_degree = 4096; //Depending on the type of your application
parms.set_poly_modulus_degree(poly_modulus_degree);
parms.set_coeff_modulus(CoeffModulus::BFVDefault(poly_modulus_degree));
parms.set_plain_modulus(1024);
```

SEALContext is a heavy class that checks the validity and properties of the parameters we just set.
Example: *SEALContext context(parms);*

Generating secret and public keys: for this purpose an instance of the **KeyGenerator** class is needed. Constructing a KeyGenerator automatically generates a secret key. Then, multiple public keys can be generated using the KeyGenerator::create_public_key.
Example:

```
KeyGenerator keygen(context);
SecretKey secret_key = keygen.secret_key();
PublicKey public_key;
keygen.create_public_key(public_key);
```

To be able to encrypt we need to construct an instance of **Encryptor** class. Note that the Encryptor only requires the public key, as expected.

```
Encryptor encryptor(context, public_key);
```

Computations on the ciphertexts are performed with the **Evaluator** class:

```
Evaluator evaluator(context);
```

When decrypting the results, an instance of the **Decryptor** class is needed. Note that the **Decryptor** requires the secret key:

```
Decryptor decryptor(context, secret_key);
```

Evaluator class contains a lot of useful calculation functions to be used to compute over ciphertexts like:
evaluator.square(x_encrypted, x_sq_plus_one);to square a ciphertext,
evaluator.add(ct1, ct2, ct3) to add ciphertext 1 with ciphertext 2 and store the result in ciphertext 3.

In addition to more functions to perform all arithmetic calculations over encrypted data.

4 BFV and CKKS schemes

Basic Notations: Let q > 1 be an integer. We denote by Z_q the set of integers (-q/2; q/2]. Note that Z_q is simply considered to be a set, and as such should not be confused with the ring Z/qZ. Similarly, R_q denotes the set of polynomials in R with coefficients in Z_q. For "a" in Z, $[a]_q$ denotes the unique integer in Z_q with $\mathbf{[a]_q = a \bmod q}$. In very few places there is the need for reduction in the interval [0; q), which will be denoted as $\mathbf{r_q(a)}$ **(remainder modulo q).**
The basic object we will work with is the polynomial ring R= Z[x]/(f(x)), which means that a polynomial K in R is in fact computed modulo f(x).

4.1 BFV: Brakerski-Fan-Vercauteren

As written in [4], BFV is a fully homomorphic encryption scheme based on Learning With Error (LWE) to work under the security assumption of Ring Learning with Error (RLWE).
The plaintext space in BFV is $R_t = Z_t[x]/(x^d + 1)$ where t is the plaintext coefficient modulus and d is the plaintext polynomial modulus. The encryption of a plaintext in BFV generates a ciphertext which is represented by two polynomials from the ring with the same polynomial modulus d, but a different coefficient modulus $q \succ\succ t$, i.e.
$R_q = Z_q[x]/(x^q + 1)$.
Fully homomorphic computations can be carried out in BFV fashion in the following manner:
Let λbe the security parameter, $q \succ 1$ be an integer polynomial modulus, d be a degree with $d = 2^n$, t be an integer plaintext coefficient modulus with $1 \prec t \prec q$, $\mathbb{R}_2$ represents the polynomial ring with coefficients modulo 2, i.e., the coefficients of the form {-1, 0, 1}, and χ be a discrete Gaussian distribution over the integers with a standard deviation σ used for error sampling.
Let δ= [q/t] and denote with $r_t(q) = q \bmod t$ then we clearly have
$q = \delta.t + r_t(q)$.
The BFV scheme can be broken down into 7 functions, of which we will talk about 5 in this mini project:
1. **PrivateKeyGen(λ):** Takes as input the security parameter λ and randomly samples $\mathbf{s} \leftarrow R_q$. It outputs a private key sk = **s**. The private key **sk** in $\mathbb{R}_2$ can take the form:

$$sk = x^{15} + x^{13} - x^{12} + x^{11} + x^9 + x^8 - x^6 + x^4 + x^2 + x + 1$$

2. **PublicKeyGen(sk):** Takes as input the private key sk and sets s = sk. Sample $\mathbf{a} \leftarrow R_q$ and $\mathbf{e} \leftarrow \chi$. It outputs the public key pk as:

$$pk = ([-\mathbf{a} * \mathbf{s} + \mathbf{e})]_q; \mathbf{a})$$

where $pk[0] = [-\mathbf{a} * \mathbf{s} + \mathbf{e})]_q$; $pk[1] = \mathbf{a}$.

3. **Encrypt(pk; m):** Takes as input public key pk and a message m in $\mathbb{R}_t$. First, it separates pk into $pk[0] = p_0$ and $pk[1] = p_1$. Then, it randomly samples **u**, **e1**, **e2** $\leftarrow \sigma$. It computes and returns the ciphertext ct as:

$$\mathbf{ct} = ([\delta * \mathbf{m} + \mathbf{p_0 u} + \mathbf{e_1}]_q; [\mathbf{p_1 u} + \mathbf{e_2}]_q)$$

4. **Decrypt(sk; ct):** Takes as input private key sk and ciphertext ct. First, it sets sk = s, $\mathbf{c_0} = \mathbf{ct}[0]$, and $\mathbf{c_1} = \mathbf{ct[1]}$. Then, it computes and outputs the message m' in R_t as:

$$\mathbf{m'} = [(t * [\mathbf{c_0} + \mathbf{c_1} * \mathbf{s}]_q)/q]_t$$

5. **Add(ct_0; ct_1):** Takes as input two ciphertexts ct_0 ct_1 and returns the resulting ciphertext ct' as:

$$\mathbf{ct'} = ([\mathbf{ct_0}[0] + \mathbf{ct_1}[0]]_q; [\mathbf{ct_0}[1] + \mathbf{ct_1}[1]]_q)$$

4.2 CKKS: Cheon-Kim-Kim-Song

BFV has a disadvantage in that it can only perform computations over the integers, or put another way, it only supports discrete computations such as Boolean, integer, or modulo operations. This makes it not very practical for the majority of real world applications since most real-world data belongs to a continuous space such as $\mathbb{R}$ or $\mathbb{C}$. CKKS solves this problem by allowing computation on complex numbers with limited precision by treating the encryption noise as part of the error that occurs naturally during approximate computations.

In the floating-point number system, a real number is represented by a product of an integer called a significant and a scaling vector which is usually called a scaling factor. For example, $1.011101 = 1011101 * 2^{-6}$ has the significant 1011101 and a scaling factor of 2^{-6}. The intuition behind this representation scheme is that the floating point number is not the exact value we want to store, but it is just the approximate value.

In CKKS, there is a distinction between a message m and a plaintext pk. The message m is a vector of floating-point numbers or complex numbers. It is first transformed into a plaintext by a public encoding map before it is encrypted and computations can be carried out. Specifically, this encoding map is a complex canonical embedding map. The use of this type of map allows the transformation to preserve the precision of the plaintext after encoding and decoding. In similar fashion, to obtain the

decrypted message, the returned plaintext from the decryption function will have to be decoded back to a vector of either floating-point or complex numbers. As mentioned, both the encoding and decoding functions are public and are just transformations from $\mathbb{C}^{n/2}$ x $\mathbb{R}$ to $\mathbb{R} = Z[x]/(x^n +1)$ for encoding and the opposite for decoding. The main application of our mini project will work on the BFV scheme so we will not present the mathematical details of CKKS scheme. The 2 schemes are very similar in mathematical topology, both are based on LWE except for the difference regarding the homomorphic calculation where BFV outputs exact values and CKKS doesn't.

5 Conclusion

In this chapter we provided an introduction to HE and its importance in the real world especially in secure cloud based applications. Then we have explained basic ideas about Microsoft SEAL library and its most important classes that we will use in our secure cloud application in addition to the mathematical equation of BFV scheme and an introduction to CKKS scheme.

CHAPTER 2: IMPLEMENTATION

1 Working Environment

To achieve the required implementation, a **virtual environment** was built using VMware workstation pro 16.1 .

The final project and performance analysis are performed on **Ubuntu Linux 18.04.4** with **8GBs of RAM, 20 GBs of storage and 1 virtual CPU and 4 virtual CPU cores,** with installed C++ compiler GNU G++ as well as the latest version of cmake (all platforms of Microsoft SEAL are built with CMake) using these commands:

$ sudo apt - get install build - essential
$ sudo apt - get install cmake

Host machine specifications:

I7 7700HQ 4 cores 2.8 GHz
16 GBs ram
512 GBs SSD

Installing Microsoft SEAL library on ubuntu linux:

1. Clone the SEAL github repository to the local machine:

   $ mkdir SEAL
   $ cd SEAL
   $ git clone https :// github . com / microsoft / SEAL .git -- recurse -submodules

2. *Build SEAL*

   $ cd native / src
   $ cmake .
   $ make
   $ cd ../..

3. Install SEAL

   $ cd native / src
   $ cmake .
   $ make
   $ sudo make install
   $ cd ../..

SEAL also makes it simple for users to run their own programs through using CMake.

After creating the program, the user writes the CMakeLists file in the same directory.
To link a C++ file with SEAL library, the user writes in a txt file named as CMakelists.txt, the following:

```
cmake_minimum_required(VERSION 3.12)

#set project name
project(SEALBFV)

#project(SEAL VERSION 3.4.5 LANGUAGES CXX C)
find_package(SEAL 3.4 REQUIRED)


#add the executable
add_executable(SEALBFV SealBFV.cpp)
target_link_libraries(SEALBFV SEAL::seal)
```

Figure 2.1 : Example SEAL CMake file.

2 Implementation

First step of this mini project code implementation was exploring the Microsoft SEAL library schemes performance and studying the speed of encoding/decoding, encrypting/decrypting of the 2 HE algorithms BFV and CKKS. A simple code was implemented to calculate an average execution time over 100 iterations of encryption/decryption, encoding/decoding functions of each of the 2 algorithms by varying the plain-text length from 1 to 50 bytes with steps equal to 1.

Different results are included in the following tables and deep analysis is achieved by plotting the results as graphs using MATLAB.

Size in Bytes	bfv_encode	bfv_decode	bfv_encrypt	bfv_decrypt	bfv_add	bfv_total
1	13	17	16346	72	12	16448
2	22	27	57206	123	15	57378
3	33	40	120834	179	21	121086
4	44	54	209586	245	29	209929
5	57	68	325799	300	33	326224
6	74	89	467984	391	45	468538
7	100	117	668173	531	59	668921
8	114	134	909029	604	69	909881
9	132	150	1182141	697	77	1183120
10	142	169	1489412	758	86	1490481
11	161	185	1827532	841	94	1828719
12	171	204	2196758	907	102	2198040
13	186	218	2597933	987	111	2599324
14	200	232	3031134	1058	120	3032624
15	214	250	3495942	1135	126	3497541
16	231	271	3993605	1216	135	3995323
17	243	286	4523895	1286	143	4525710
18	257	297	5083202	1353	148	5085109
19	271	315	5673954	1428	155	5675968
20	284	333	6296592	1509	165	6298718
21	301	346	6951432	1578	172	6953657
22	312	366	7637723	1658	179	7640059
23	329	380	8356141	1731	189	8358581
24	342	400	9107350	1809	198	9109901
25	358	417	9891036	1879	206	9893690
26	373	429	10705994	1953	216	10708749
27	384	449	11551179	2032	230	11554044
28	401	468	12429629	2108	231	12432606
29	414	489	13340121	2191	240	13343215
30	432	504	14283378	2260	246	14286574
31	441	521	15258366	2334	256	15261662
32	458	558	16265535	2411	264	16268962
33	470	573	17304674	2486	274	17308203

34	493	603	18377454	2627	284	18381177
35	572	746	19592093	2979	291	19596390
36	515	602	20736876	2707	297	20740700
37	529	622	21902856	2782	307	21906789
38	543	633	23099781	2851	314	23103808
39	559	656	24329566	2943	323	24333724
40	571	679	25592785	3018	332	25597053
41	586	693	26888434	3094	338	26892807
42	600	699	28214972	3158	346	28219429
43	615	728	29571073	3247	354	29575663
44	630	738	30958909	3316	362	30963593
45	644	754	32380677	3389	370	32385464
46	660	777	33836321	3482	380	33841240
47	672	781	35321057	3537	388	35326047
48	684	798	36835439	3618	394	36840539
49	699	819	38381728	3686	405	38386932
50	716	825	39961542	3752	413	39966835

Table 2.1 : BFV performance time in MicroSeconds

Size in Bytes	ckks_encode	ckks_decode	ckks_encrypt	ckks_decrypt	ckks_add	ckks_total
1	464	411	17519	13	11	18407
2	891	788	65075	22	16	66776
3	1341	1187	144628	34	25	147190
4	1794	1591	255530	46	35	258961
5	2227	1979	398037	57	42	402300
6	2678	2377	572220	68	51	577343
7	3128	2778	778393	80	60	784379
8	3576	3170	1015991	92	69	1022829
9	4473	4014	1324414	166	78	1333067
10	4473	3977	1661186	119	87	1669755
11	4898	4351	1993506	124	91	2002879
12	5349	4748	2355259	134	98	2365490
13	5797	5140	2749085	148	106	2760170
14	6235	5541	3174160	157	115	3186093

15	6679	5926	3629883	168	122	3642656
16	7101	6318	4115677	179	130	4129275
17	7578	6727	4633969	190	139	4648464
18	8013	7114	5185010	201	150	5200338
19	8468	7520	5766963	213	155	5783164
20	8906	7913	6380903	224	165	6397946
21	9347	8298	7025890	236	173	7043771
22	9799	8705	7702610	248	180	7721362
23	10228	9085	8410747	260	189	8430320
24	10687	9496	9149575	269	197	9170027
25	11134	9901	9921361	279	207	9942675
26	11971	10658	10728014	346	215	10750989
27	12112	10773	11631983	321	222	11655189
28	12442	11051	12494756	315	230	12518564
29	12888	11456	13388325	328	237	13412997
30	13350	11862	14315414	340	246	14340966
31	13806	12265	15275772	349	254	15302192
32	14227	12635	16264828	358	263	16292048
33	14709	13061	17287625	371	273	17315766
34	15111	13422	18341452	385	279	18370370
35	15590	13859	19425565	395	288	19455409
36	16553	14770	20564166	472	295	20595961
37	16482	14634	21776862	416	303	21808394
38	17573	15696	22969847	454	332	23003570
39	17647	15705	24279512	457	331	24313321
40	17808	15828	25528483	450	330	25562569
41	18318	16298	26806487	466	338	26841569
42	18678	16600	28116151	473	346	28151902
43	19140	17009	29453791	484	354	29490424
44	20113	17919	30872128	564	362	30910724
45	20038	17797	32301857	507	370	32340199
46	23429	20926	34028916	795	379	34074066
47	20931	18598	35606697	531	388	35646757
48	21369	18995	37103006	538	396	37143908
49	21801	19380	38629199	551	404	38670931

50	22732	20231	40202947	621	414	40246531

Table 2.2 : CKKS performance time MicroSeconds

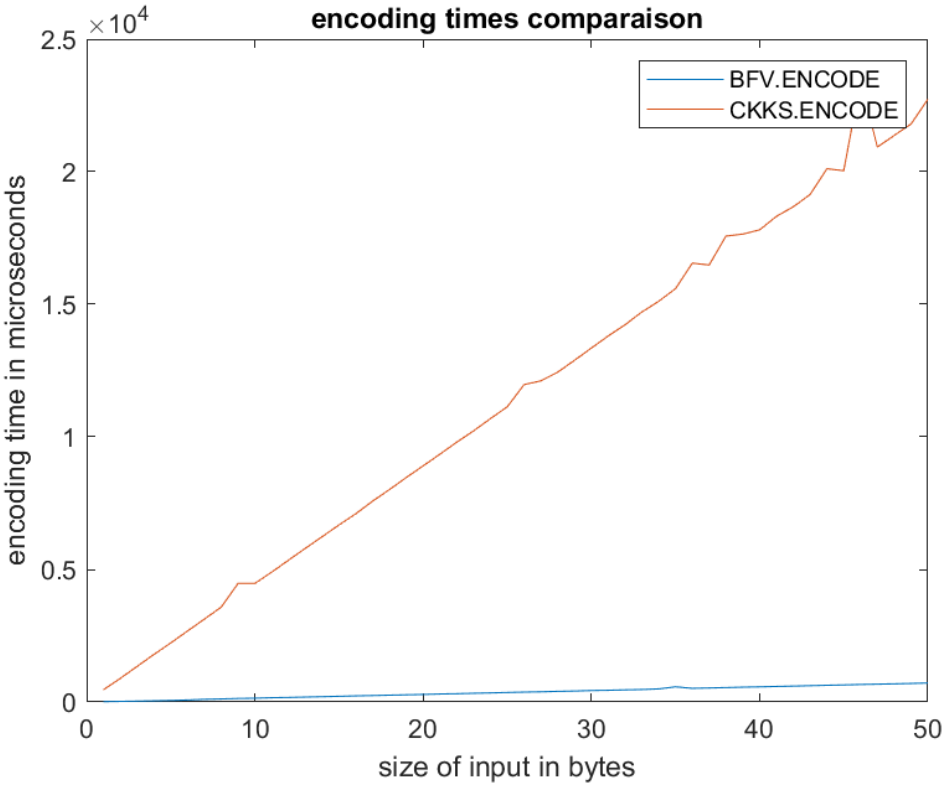

Figure 2.2: Encoding times comparison

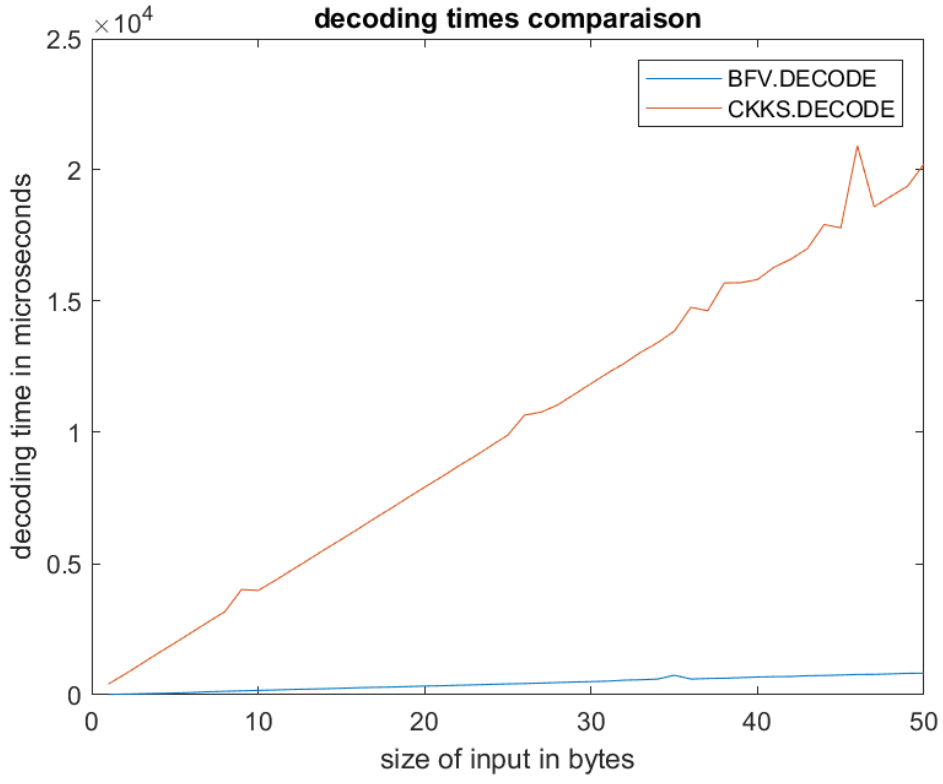

Figure 2.3: Decoding times comparison

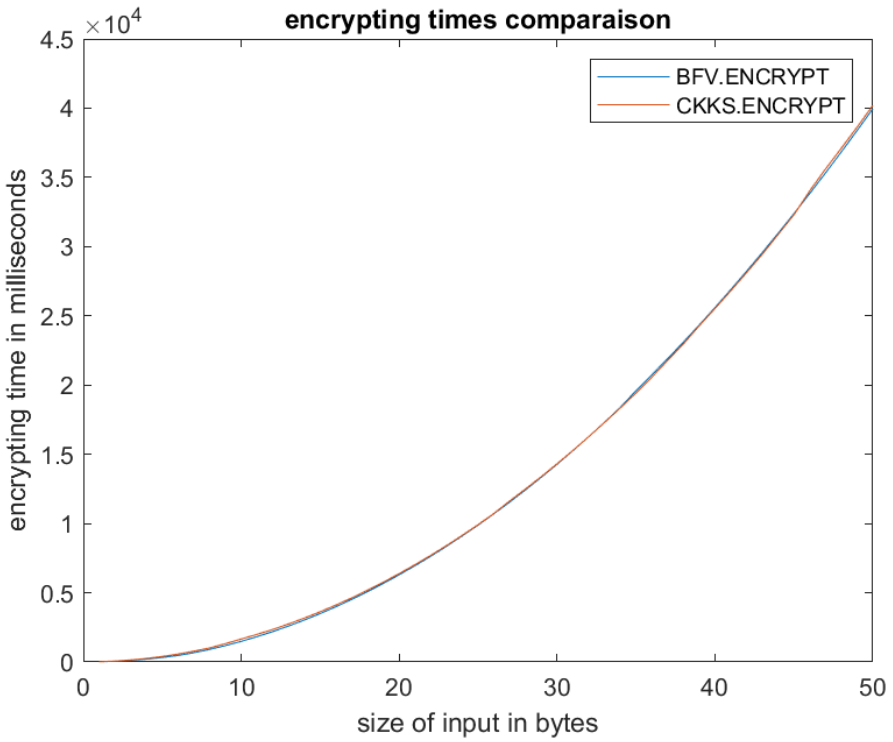

Figure 2.4: Encrypting times comparison

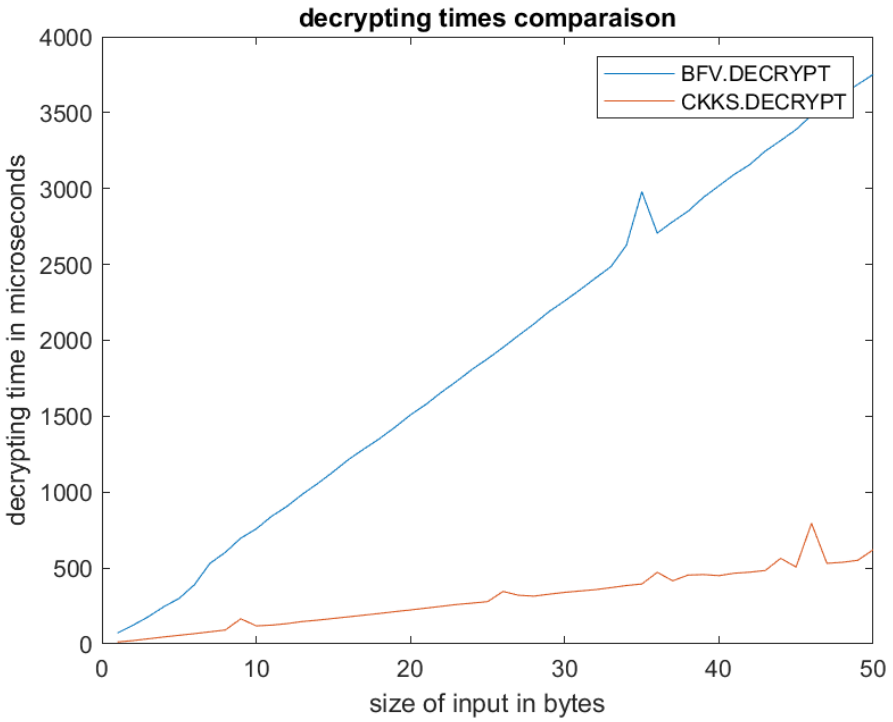

Figure 2.5: Decrypting times comparison

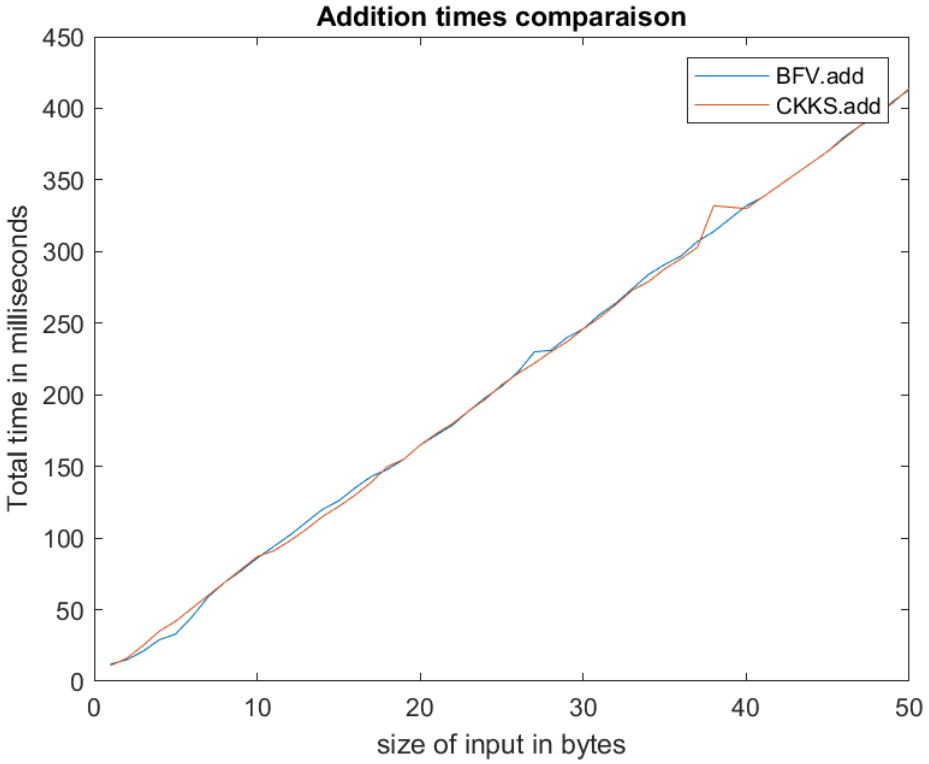

Figure 2.6: Addition times comparison

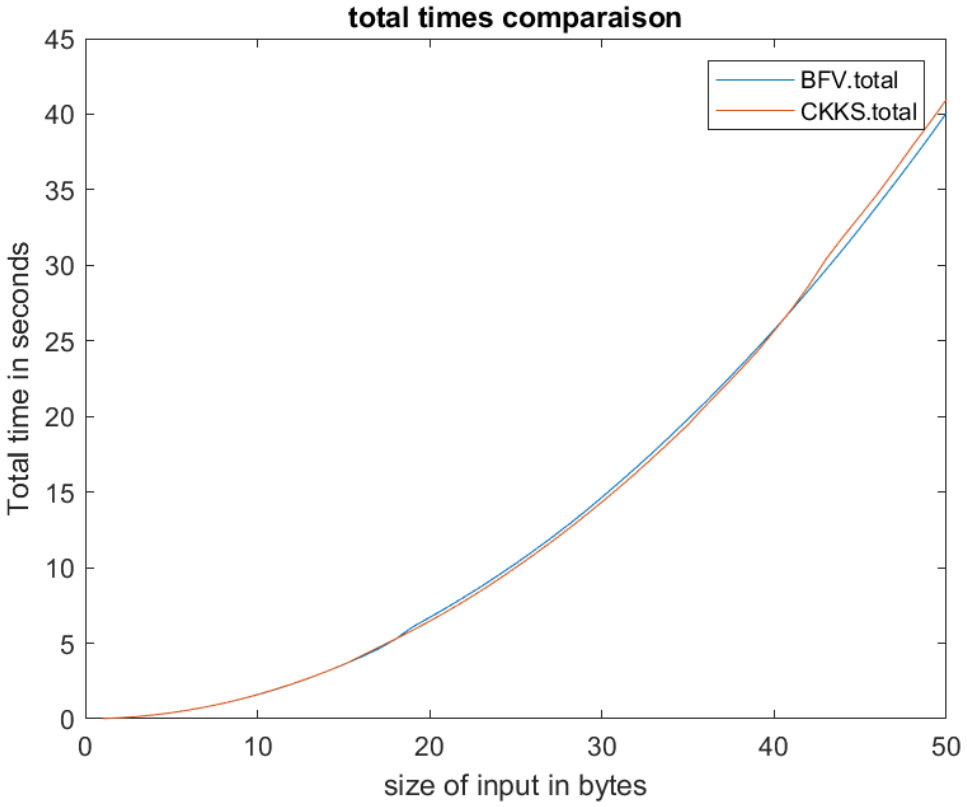

Figure 2.7: Total times comparison

3 Discussion and performance analysis

By analyzing the above results and assuming both schemes are compared over the same level of security, we can see that encoding/decoding in the BFV scheme is very fast in reference to CKKS. Encrypting graphs show that the 2 schemes have the same performance, unlike decrypting which shows that CKKS is faster .

Total time comparison graph shows clearly that BFV and CKKS schemes have the same performance which may be a point set by Microsoft.

And because the BFV scheme provides accurate encrypting/decrypting calculations which is a feature not available in the CKKS scheme which provides approximate calculations, our application will use BFV scheme as it is based on encrypting human genetic data in integer form with high accuracy.

4 Conclusion

In this chapter, we provided our working environment and our project operating system in addition to a comparison of the 2 HE schemes (BFV and CKKS) in terms of execution times for different cryptographic functions and analytic functions over the cipher-texts. Based on the results we have selected the BFV scheme to start implementing our application.

CHAPTER 3: GENETIC DISEASE DETECTION USING HOMOMORPHIC ENCRYPTION

1 Introduction

Genome data can be used in many applications including healthcare, biometric research, and forensics. This kind of data is very confidential to the patient and should not be shared to anyone unauthorized in plain-text. Several data analytics operations should be done on patient genome data to produce results of different types of DNA tests. But as mentioned before, this kind of data must be encrypted and then sent to the cloud as it requires a lot of storing capacity. The cloud is a maturing technology that provides huge capacity and computation power. The best way to assure patient genome security is to encrypt these data using HE.

2 Case Study DNA

The gene data is uploaded and stored in a folder "Cloud" simulating a cloud environment. The data file contains multiple genotype information entries, where each of them consists of :

- Chr (chromosome): represents the chromosome where the gene is located between 1 to 22, *X*, and *Y*.
- Pos: represents the base position of the gene variation in the Chromosome.
- Loc (rsid) : indicates the location of the gene.
- Ref: represents the base information before the mutation occurs.
- Alt: represents the base information after the mutation occurs.

rsid	chromosome	position	Ref	Alt
rs369202065	1	569388	G	G
rs199476136	1	569400	T	T
rs3131972	1	752721	A	G
rs114525117	1	759036	G	G
rs12124819	1	776546	A	A
rs4040617	1	779322	A	G
rs141175086	1	780397	G	G
rs11240777	1	798959	A	G
rs6681049	1	800007	C	C

Table 3.1: VCF file of a patient (Source: Harvard University)

2.1 Genetic Data Encoding

To locate gene mutation we use the 2 entries Chr (chromosome) and Pos(Position) only, capturing a mutation can be obtained by simply comparing the Ref base and Alt base.

We only need to match the Chr and Pos information between the patient encrypted data from the cloud and the normal base information, then we can get the corresponding ref and alt information of base variation at the same location. Then we compare the client base change information from the cloud with the normal base change information, in order to get the final result, which may be match or no match.

To apply HE scheme, we must encode the genetic data using the specified algorithm hereunder:

Let d_i be the position information of the *ith* entry in the gene database, α_i the variation information of the *ith* entry in the gene database, and α_{iRef} and α_{iAlt} the integers encoding of reference genome and alternate genome, respectively.

First, for the encoding of the gene position d_i, we define a mapping from (Chr, Pos) to d_i in order to get a simple integer output:

$$\theta:(\mathbb{Z},\mathbb{Z})\rightarrow\mathbb{Z}$$
$$(Chr,Pos)\rightarrow d_i=Chr_i+24*Pos_i.$$

For the variation information α_i,we describe how to encode it as follows:
Each SNP is represented by two binary numbers as follows:

- $A\rightarrow 00$
- $T\rightarrow 01$
- $G\rightarrow 10$
- $C\rightarrow 11$

We encode them according to their order and then pad 1 to the left of the bit string so as to not get a 0 output in case of the A string, and finally compute the corresponding decimal value. For example, an instance A is encoded as follows 1| 0 0=(100)=4 in decimal. n_{nsp} denotes the number of genetic rows in the VCF file, which makes the length of the base string is $l_{nsp}=2.n_{nsp}+1$. Encoding the variation information αi is expressed as an integer using the following formula:

$$\alpha_i = 2^{l_{nsp}.\alpha_{iRef}} + \alpha_{iAlt}.$$

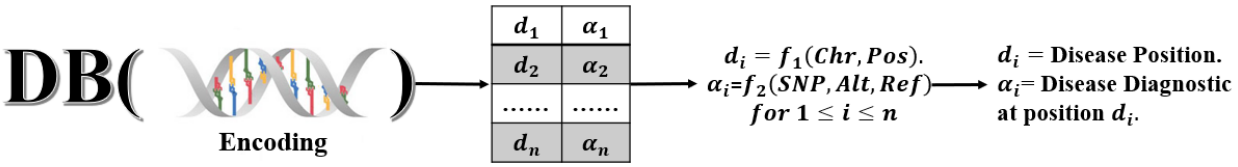

Figure 3.1: Genetic Data Encoding

After performing this algorithm, we obtain the following input data (by using only 5 rows from the last VCF file),

d_i	α_i
13665313	12294
13665601	10245
18065305	8198
18216865	12294
18637105	8196

Table 3.2: Encoded Genome data

2.2 Secure Genome Analysis

After encoding genome data using the previous method and producing a table including several d's and α's, we start encrypting each d using the HE BFV scheme and store the result in our cloud folder. In addition, we encrypt each α using AES encryption algorithm, because α is not eligible for computations.

Our application is simple, given a position d' in addition to its variation information α' , the hospital looks if the position d' exists in the cloud. If it exists, it returns the index of the position of d_i which is i . After that, the cloud returns to the hospital the value of encrypted AES α corresponding to the position i in the cloud.

We finally decrypt α and compare it to α', if α=α' then the client has normal gene information and doesn't suffer from any anomaly. Otherwise, the client suffers from an anomaly at this position.

Next figure from [2] explains our application algorithm.

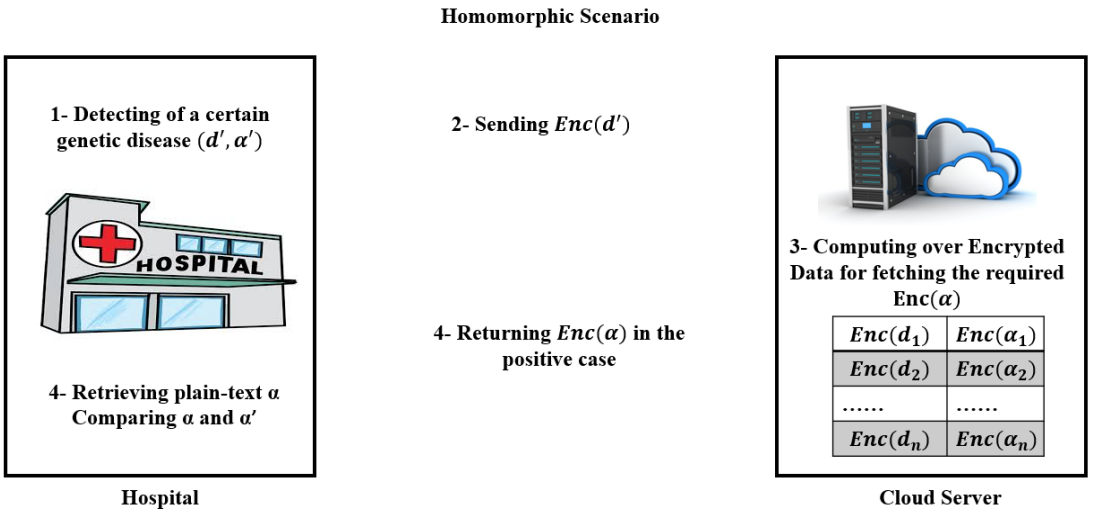

Figure 3.2: Genetic Disease Detection

Our application gives the user (which is the hospital in our case) 2 options at the startup, the first option is to add user data to the cloud (the application takes the input, encrypts and stores it in the cloud). The second option is to check to user data anomaly allowing the use to enter a value *d*' and α' and perform the algorithm listed in the previous section.

2.3 Genetic Disease detection

1. The Hospital takes the genetic position d' to be tested, and its normal genetic variation information α', as inputs.
2. The Hospital sends the homomorphic encryption $Enc(-d')$ to the cloud.
3. The Cloud performs homomorphic addition of $Enc(-d')$ with the whole genome genetic position of the client and ships them back to the Hospital as a table.
For all i, it performs $Enc(d_i) \oplus_c Enc(-d') = Enc(d_i - d')$ where $\oplus_c$ is the addition operation over the ciphertext space C.
4. The Hospital decrypts the answer shipped back by the Cloud using HE secret parameters.
5. If d' exists in the whole genome of the client, so we can find i such that $d_i = d'$,then we can find in the shipped back results from the cloud an instance which is equal to $Enc(d_i - d') = Enc(0)$. So, the decryption of this instance is 0. Afterwards, we must find the index of this instance in the results table.

6. The Cloud using index i of the required d_i, finds the required $Enc(\alpha_i)$ in the stored encrypted table of genetic variation information. Then the Cloud ships back the encryption of the required α_i to the Hospital.

7. The Hospital decrypts $Enc(\alpha_i)$ with the AES decryption algorithm, and finally gets the required α_i, then checks if $\alpha_i = \alpha'$.

The following is an example of encrypted α_i using AES of the results written in the table 3.2

```
U2FsdGVkX1/XUDIp8MsdkzjMuy+D/mgrHAQZ9a33axI=
U2FsdGVkX19rFdsN2dLP191nnHrWpvTiiMEGjksfWYQ=
U2FsdGVkX19SwNFzc26LzpWEwYftauSi2Rg2zaq1/T4=
U2FsdGVkX19ttmakAU6k5G7Rx69/RTuG1Vw9l1M4e4c=
U2FsdGVkX1/69roPMQVryMKe8+i8nnGdO07oqnaBnNc=
U2FsdGVkX19scPgVkQzSm/IBCP79UinMc/dKRCENj2Y=
U2FsdGVkX1+KXNa/T5KqemA1dauSFz0+EFiZa41Pbkw=
U2FsdGVkX18hdzFO3k0uzxpOehBq7IX1RPuu/5dmcJI=
U2FsdGVkX1/Lk2h65NFya7nLkbA+Q6JHYGyMOweI+ng=
U2FsdGVkX1/Bp/6eNFewG1qFXD9PJtsH0M/3NmZTvVU=
U2FsdGVkX18Gjdl7L+NrGUVOEY4V36V2804ji5l25M0=
U2FsdGVkX1/UDSpgAUKGb/6hsoohNGj9QP5VlQGzJug=
```

Table 3.3: AES encryption of each α_i listed in table 3.2

AES encryption and decryption is computed using OpenSSL 1.1.1 on Linux.

2.4 Query Time

The required time to do the homomorphic addition at the cloud side, return the table of d's, find α, decrypt and return it is on an average of 100 iterations is the following: **17909** MicroSeconds.

3 Conclusion

In this chapter we provided the basic goal of our application as well as the method of encoding the genome data in an effort to do homomorphic operations on it while encrypted in the cloud. We also listed our main application algorithm.

CONCLUSION

Project Conclusion:

Encryption is paving its path in all areas of technology. HE is the process of performing computation on the encrypted data without revealing the original data. In this mini project we provided a description of HE and explored a new homomorphic library called Microsoft SEAL while comparing the performance of BFV and CKKS schemes. We then chose BFV to implement our secure genetic disease application as we had no need for floating point or complex numbers.

Personal Opinion and further development:

We believe that HE is a very interesting development in the world of cryptography and data confidentiality, especially in a world where human personal data is becoming a commodity to be shared and sold to advertisement agencies.

As for our project, further developments can include the implementation of a fully functioning graphical user interface GUI along with the use of a database to store the information on a real cloud.

REFERENCES

[1] Hariss, K., Chamoun, M., & Samhat, A. E. (2019). (rep.). Phd thesis "*Homomorphic Encryption for Modern Security Applications"* Lebanese University and Saint Joseph University.

[2] Gentry, R. *(2009). "Fully homomorphic encryption using ideal lattices. In Proceedings of the forty-first annual ACM symposium on Theory of computing (STOC '09)"*. Association for Computing Machinery, New York, NY, USA, 169–178. DOI:https://doi.org/10.1145/1536414.1536440

[3] Cheon, J. H., Kim, A., Kim, M., & Song, Y. (2017). Homomorphic encryption for arithmetic of approximate numbers. *Advances in Cryptology – ASIACRYPT 2017,* 409-437. doi:10.1007/978-3-319-70694-8_15

[4] Carey, A. (2020). "*On the Explanation and Implementation of Three Open-Source Fully Homomorphic Encryption Libraries"*. Computer Science and Computer Engineering Undergraduate Honors Theses.
Retrieved from https://scholarworks.uark.edu/csceuht/77

[5] Microsoft seal: Fast and Easy-to-use homomorphic encryption library. (2020, November 12). Retrieved from https://www.microsoft.com/en-us/research/project/microsoft-seal/

[6] Microsoft. (n.d.). Microsoft/seal. Retrieved from https://github.com/microsoft/SEAL

[7] Chen, H., Laine, K., & Player, R. (2017). Simple encrypted arithmetic library - seal v2.1. *Financial Cryptography and Data Security,* 3-18. doi:10.1007/978-3-319-70278-0_1

[8] Zhou, T. P., Li, N. B., Yang, X. Y., Lv, L. Q., Ding, Y. T., & Wang, X. A. (2018). "Secure testing for genetic diseases on encrypted genomes with homomorphic encryption scheme". *Security and Communication Networks, 2018*, 1-12. doi:10.1155/2018/4635715

[9] Wiley, J. & Sons (2019) "*Cryptography Apocalypse: Preparing for the Day When Quantum Computing Breaks Today's Crypto*"

[10] O. Regev, "*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, (New York, NY, USA), pp. 84–93, ACM, 2005

[11] Ball, M. P., Bobe, J. R., Chou, M. F., Clegg, T., Estep, P. W., Lunshof, J. E., Vandewege, W., Wait Zaranek, A., & Church, G. M. (2014). *Harvard Personal Genome Project: Lessons from Participatory Public Research*. Genome Medicine, 6, 10. https://doi.org/10.1186/gm527

APPENDIX

Time Performance calculation code:

```
#include "seal/seal.h"
#include <chrono>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <sstream>
#include <string>
#include <vector>
using namespace std;
using namespace seal;
using namespace std::chrono;

int main(){
        high_resolution_clock::time_point start, stop;
        int count=100;
    cout<<"This is an easy example of homomorphic encryption using BFV scheme"<<endl;
    cout << "Setting encryption parameters";
        //setting BFV parameters
        EncryptionParameters parms(scheme_type::bfv);
    parms.set_poly_modulus_degree(1024);
    parms.set_coeff_modulus(CoeffModulus::BFVDefault(1024));
    parms.set_plain_modulus(12289);
    SEALContext context(parms);
    cout<< " ..... Done." << endl;

        //generating Keys, Encryptor and Decryptor
    cout << "Generating public,private keys and evaluaters ";
    KeyGenerator keygen(context);
    PublicKey pk;
    keygen.create_public_key(pk);
    SecretKey sk=keygen.secret_key();
    Encryptor encryptor(context, pk);
        BatchEncoder encoder(context);
    Evaluator evaluator(context);
    Decryptor decryptor(context, sk);
    cout<< " ..... Done." << endl;

        microseconds time_encode_sum(0);
        microseconds time_decode_sum(0);
        microseconds time_encrypt_sum(0);
        microseconds time_decrypt_sum(0);

        vector<microseconds> total_encode(0);
        vector<microseconds> total_decode(0);
        vector<microseconds> total_encrypt(0);
        vector<microseconds> total_decrypt(0);

    cout<<"Starting Timing"<<endl;
    int s=1;
        //creating vector data1 with 1 element "s1" to use encoder in case s1 is negative or a floating
point and encrypting into ct1
        vector<int64_t> data = {s};
        vector<int64_t> result;
        for (int n=0;n<=50;n++){

                total_encode.push_back(microseconds::zero());
                total_decode.push_back(microseconds::zero());
                total_encrypt.push_back(microseconds::zero());
                total_decrypt.push_back(microseconds::zero());
```

```
		for (int i=0;i<count;i++){

			Plaintext pt;
			for (int j=0;j<n;j++){
				start = high_resolution_clock::now();
				encoder.encode(data, pt);
				stop = high_resolution_clock::now();
				time_encode_sum += duration_cast<microseconds>(stop-start);

				Ciphertext ct;

				start = high_resolution_clock::now();
				encryptor.encrypt(pt, ct);
				stop = high_resolution_clock::now();
				time_encrypt_sum += duration_cast<microseconds>(stop-start);

				start = high_resolution_clock::now();
				decryptor.decrypt(ct, pt);
				stop = high_resolution_clock::now();
				time_decrypt_sum += duration_cast<microseconds>(stop-start);

				start = high_resolution_clock::now();
				encoder.decode(pt, result);
				stop = high_resolution_clock::now();
				time_decode_sum += duration_cast<microseconds>(stop-start);

				}
			total_encode[n]+=time_encode_sum;
			total_decode[n]+=time_decode_sum;
			total_encrypt[n]+=time_encrypt_sum;
			total_decrypt[n]+=time_decrypt_sum;

			time_encode_sum=microseconds::zero();
			time_decode_sum=microseconds::zero();
			time_decrypt_sum=microseconds::zero();
			time_decode_sum=microseconds::zero();
				}

			}
	cout<<"---------encode:----------"<<endl;
	for(int i=1;i<total_encode.size();i++){
			auto avg_encode = total_encode[i].count() / count;
			cout<<avg_encode<<endl;
			}

	cout<<"---------decode:----------"<<endl;
	for(int i=1;i<total_encode.size();i++){
			auto avg_decode = total_decode[i].count() / count;
			cout<<avg_decode<<endl;
			}

	cout<<"---------encrypt:----------"<<endl;
	for(int i=1;i<total_encode.size();i++){
			auto avg_encrypt = total_encrypt[i].count() / count;
			cout<<avg_encrypt<<endl;
			}

	cout<<"---------decrypt:----------"<<endl;
	for(int i=1;i<total_encode.size();i++){
			auto avg_decrypt = total_decrypt[i].count() / count;
			cout<<avg_decrypt<<endl;
			}
cout<<"------------------------BFVEND-------------------------------"<<endl<<endl;
```

```
        cout<<"------------------------CKKS--------------------------------"<<endl;

        cout<<"This is an easy example of homomorphic encryption using CKKS scheme"<<endl;
    cout << "Setting encryption parameters";
        //setting CKKS parameters
        EncryptionParameters parms1(scheme_type::ckks);
        parms1.set_poly_modulus_degree(1024);
        parms1.set_coeff_modulus(CoeffModulus::BFVDefault(1024));
        SEALContext context1(parms1);
        double scale1 = sqrt(static_cast<double>(parms1.coeff_modulus().back().value()));
        cout<< " ..... Done." << endl;

        //generating Keys, Encryptor and Decryptor
    cout << "Generating public,private keys, evaluaters and encoder";
        KeyGenerator keygen1(context1);
        auto secret_key1 = keygen1.secret_key();
        PublicKey public_key1;
        keygen1.create_public_key(public_key1);
        Encryptor encryptor1(context1, public_key1);
        Evaluator evaluator1(context1);
        Decryptor decryptor1(context1, secret_key1);
        CKKSEncoder encoder1(context1);
        cout<< " ..... Done." << endl;

        microseconds time_encode_sum1(0);
        microseconds time_decode_sum1(0);
        microseconds time_encrypt_sum1(0);
        microseconds time_decrypt_sum1(0);

        vector<microseconds> total_encode1(0);
        vector<microseconds> total_decode1(0);
        vector<microseconds> total_encrypt1(0);
        vector<microseconds> total_decrypt1(0);

        cout<<"Starting Timing"<<endl;
    double s1=1;
        //creating vector data1 with 1 element "s1" to use encoder in case s1 is negative or a floating
point and encrypting into ct1
        vector<double> data1 = {s1};
        vector<double> result1;
        for (int n=0;n<=50;n++){
                total_encode1.push_back(microseconds::zero());
                total_decode1.push_back(microseconds::zero());
                total_encrypt1.push_back(microseconds::zero());
                total_decrypt1.push_back(microseconds::zero());
                for (int i=0;i<count;i++){
                        Plaintext pt1;
                        for (int j=0; j<n; j++){
                                start = high_resolution_clock::now();
                                encoder1.encode(data1, scale1, pt1);
                                stop = high_resolution_clock::now();
                                time_encode_sum1 += duration_cast<microseconds>(stop-start);

                                Ciphertext ct1;
                                start = high_resolution_clock::now();
                                encryptor1.encrypt(pt1, ct1);
                                stop = high_resolution_clock::now();
                                time_encrypt_sum1 += duration_cast<microseconds>(stop-start);

                                start = high_resolution_clock::now();
                                decryptor1.decrypt(ct1, pt1);
                                stop = high_resolution_clock::now();
                                time_decrypt_sum1 += duration_cast<microseconds>(stop-start);
```

```
                                start = high_resolution_clock::now();
                                encoder1.decode(pt1, result1);
                                stop = high_resolution_clock::now();
                                time_decode_sum1 += duration_cast<microseconds>(stop-start);

                                }
                        total_encode1[n]+=time_encode_sum1;
                        total_decode1[n]+=time_decode_sum1;
                        total_encrypt1[n]+=time_encrypt_sum1;
                        total_decrypt1[n]+=time_decrypt_sum1;

                        time_encode_sum1=microseconds::zero();
                        time_decode_sum1=microseconds::zero();
                        time_decrypt_sum1=microseconds::zero();
                        time_decode_sum1=microseconds::zero();
                                }

                        }
                cout<<"---------encode:----------"<<endl;
                for(int i=1;i<total_encode1.size();i++){
                        auto avg_encode1 = total_encode1[i].count() / count;
                        cout<<avg_encode1<<endl;
                        }

                cout<<"---------decode:----------"<<endl;
                for(int i=1;i<total_encode1.size();i++){
                        auto avg_decode1 = total_decode1[i].count() / count;
                        cout<<avg_decode1<<endl;
                        }

                cout<<"---------encrypt:----------"<<endl;
                for(int i=1;i<total_encode1.size();i++){
                        auto avg_encrypt1 = total_encrypt1[i].count() / count;
                        cout<<avg_encrypt1<<endl;
                        }

                cout<<"---------decrypt:----------"<<endl;
                for(int i=1;i<total_encode1.size();i++){
                        auto avg_decrypt1 = total_decrypt1[i].count() / count;
                        cout<<avg_decrypt1<<endl;
                        }
                cout<<"-------------------------CKKSEND--------------------------------"<<endl;}
```

Homomorphic Encryption Application Code:

```
#include "seal/seal.h"
#include <algorithm>
#include <chrono>
#include <cstddef>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <string>
#include <vector>

using namespace std::chrono;
using namespace seal;
using namespace std;

void banner(){
        cout<<"        ****************************************"<<endl;
        cout<<"        *                                      *"<<endl;
        cout<<"        *          Genetic Disease Department  *"<<endl;
        cout<<"        *                                      *"<<endl;
        cout<<"        ****************************************"<<endl<<endl;
```

```
}

void pause(){
	cin.get();
	do {
		cout << '\n' << "Press the Enter key to continue.";
	} while (cin.get() != '\n');
}

string retrievealpha(int index, string filename){
	ifstream in;
	//system("pwd");
	string alphalocation="Cloud/"+filename+"/alpha.txt";
	in.open(alphalocation);
	string temp;
	for (int i=0; i<=index; i++){
		getline(in, temp);
		}
	system(("echo "+temp+ " | openssl enc -aes-256-cbc -d -a -iter 1000 -pass pass:\"pass\"
>temp.txt").c_str());
	in.close();
	in.open("temp.txt");
	in>>temp;
	in.close();
	system("rm temp.txt");
	return temp;
}

void saveCiphertext(Ciphertext encrypted, string filename){
	ofstream ct;
	ct.open(filename, ios::binary);
	encrypted.save(ct);
}

Ciphertext loadCiphertext(string filename, EncryptionParameters parms){

	SEALContext context(parms);

	ifstream ct;
	ct.open(filename, ios::binary);
	Ciphertext result;
	result.load(context, ct);

	return result;
}

void saveSecretKey(SecretKey sk, string filename){
	ofstream ct;
	ct.open(filename, ios::binary);
	sk.save(ct);
}

SecretKey loadSecretKey(string filename, EncryptionParameters parms){

	SEALContext context(parms);

	ifstream ct (filename);
	if (ct.is_open()){
	SecretKey result;
	result.load(context, ct);

	return result;}

}
void savePublicKey(PublicKey pk, string filename){
	ofstream ct;
	ct.open(filename, ios::binary);
	pk.save(ct);
}
PublicKey loadPublicKey(string filename, EncryptionParameters parms){
```

```
        SEALContext context(parms);

        ifstream ct (filename);
        if (ct.is_open()){
        PublicKey result;
        result.load(context, ct);

        return result;}
}

bool exists(const char *fileName){
   ifstream infile(fileName);
   return infile.good();
}

int convert (char z){
        if(z=='A') return 4;
        if (z=='T') return 5;
        if (z=='G') return 6;
        if (z=='C') return 7;
}

bool snpEncode(string filename, vector<int64_t> &d, vector<int64_t> &alpha){
        ifstream infile(filename);
        if(infile.is_open()){
                string s,b;
                int chr,pos;
                char ref,alt;
                getline(infile,s);
                while(!infile.eof()){
                        infile>>b>>chr>>pos>>ref>>alt;
                        d.push_back(-(chr+24*pos));
                        alpha.push_back(2048*convert(ref)+convert(alt));
                        }
                infile.close();
                return true;
                }
        else
                cout<<"file not found"<<endl;
                return false;
}
bool sendtocloud(string filename, EncryptionParameters parms, PublicKey pk){
        vector<int64_t> d;
        vector<int64_t> alpha;
        bool test=snpEncode("Client/Users/"+filename, d, alpha);
        if (test){
                SEALContext context(parms);
                Encryptor encryptor(context, pk);
                BatchEncoder encoder(context);
                Plaintext pt;
                encoder.encode(d,pt);
                Ciphertext ct;
                encryptor.encrypt(pt,ct);
                system(("mkdir Cloud/"+filename).c_str());
                string dsavelocation= "Cloud/"+filename+"/d.txt";
                string alphasavelocation= "Cloud/"+filename+"/alpha.txt";
                saveCiphertext(ct, dsavelocation);
                for(int i=0; i<alpha.size(); i++){
                        system(("echo \""+ to_string(alpha[i])+ "\" | openssl enc -aes-256-cbc  -a -iter 1000
-pass pass:\"pass\" >> "+alphasavelocation).c_str());
                        //system("clear");
                        }
                ofstream out;
                out.open("Client/db", ios_base::app);
                out<<filename<<endl;
                out.close();
                return true;
                }
        else return false;
}
```

```
Ciphertext retrievefromcloud(int64_t d1, EncryptionParameters parms, PublicKey pk, string filename){
        SEALContext context(parms);
        Encryptor encryptor(context, pk);
        BatchEncoder encoder(context);
        Evaluator evaluator(context);
        vector<int64_t> d;
        for (int i=0; i<5; i++){d.push_back(d1);}
        Plaintext pt;
        encoder.encode(d,pt);
        Ciphertext ct,ct1,ct2;
        encryptor.encrypt(pt,ct1);
        string dsavelocation= "Cloud/"+filename+"/d.txt";
        ct2=loadCiphertext(dsavelocation, parms);
        evaluator.add(ct1,ct2,ct);
        return ct;
}
int findindex(Ciphertext ct, EncryptionParameters parms, SecretKey sk){
        SEALContext context(parms);
        Evaluator evaluator(context);
        BatchEncoder encoder(context);
    Decryptor decryptor(context, sk);
        Plaintext pt;
        decryptor.decrypt(ct, pt);
        vector<int64_t> d;
        encoder.decode(pt, d);
        for(int i=0; i<5; i++)
                if(d[i]==0) return i;
                return -1;

}
bool userexists(string filename){
        ifstream in("Client/db");
        string user;
        while(!in.eof()){
        in>>user;
        if(user==filename) return true;}
        return false;
}
int main(){
        //setting BFV parameters

        system("clear");
        cout << "Setting encryption parameters";

        EncryptionParameters parms(scheme_type::bfv);
        parms.set_poly_modulus_degree(2048);
        parms.set_coeff_modulus(CoeffModulus::BFVDefault(2048));
        parms.set_plain_modulus(PlainModulus::Batching(2048, 30));
        SEALContext context(parms);


        cout<< " ..... Done." << endl;

        //generating Keys, Encryptor and Decryptor
        cout << "Generating public,private keys";

        KeyGenerator keygen(context);

        PublicKey pk;
        if(exists("Client/Keys/PublicKey"))     pk=loadPublicKey("Client/Keys/PublicKey",parms);
        else {keygen.create_public_key(pk);     savePublicKey(pk,"Client/Keys/PublicKey");}

        SecretKey sk;
        if(exists("Client/Keys/SecretKey"))     sk=loadSecretKey("Client/Keys/SecretKey",parms);
        else {sk=keygen.secret_key();           saveSecretKey(sk,"Client/Keys/SecretKey");}

        cout<< " ..... Done."<<endl;
        bool file;
        string filename;
        string alphai;
        string alpha1;
```

```
	int index;
	Ciphertext ct;
	if(!exists("Client/db")) system("touch Client/db");
	//start of code

	char choice='0';
	while(true){

		switch (choice){

			case '0':
				system("clear");
				banner();
				cout<<"Welcome, what would you like to do?"<<endl
					<<"(1) To input user data to cloud"<<endl
					<<"(2) To check user data"<<endl
					<<"(q) To exit"<<endl;
				cin>>choice;
				break;

			case '1':
				system("clear");
				banner();
				cout<<"Enter Patient Filename: "<<endl;
				cin>>filename;
				if(userexists(filename)) {cout<<"User already exists!"<<endl; choice='1';}
					else{
					file=sendtocloud(filename, parms, pk);//happening on the server
and saving on the cloud
					if (file) {cout<<"Done."<<endl; choice='0';}
					else choice='0';
					}
				pause();
				break;
			case '2':
				system("clear");
				banner();
				cout<<"Input user Filename:"<<endl;
				cin>>filename;
				if(userexists(filename)){
					cout<<"Input value d':"<<endl;
					int64_t d1;
					cin>>d1;
					ct=retrievefromcloud(d1, parms, pk, filename);//happening on
the cloud
					cout<<"Input value alpha':"<<endl;
					cin>>alpha1;
					index=findindex(ct, parms, sk);
					if(index>=0) {
						cout<<"Alpha at index: "<<index<<endl;
						alphai=retrievealpha(index, filename);
						if(stoi(alphai)==stoi(alpha1)) cout<<"No genertic
desease"<<endl;
						else cout<<"Genetic desease found"<<endl;

						}
					else cout<<"d' not found"<<endl;
					choice='0';
						}
				else{ cout<<"User doesn't exist!"<<endl; choice='0';}
				pause();
				break;
			case 'q':
				cout<<"Thank you for coming!"<<endl;
				return 0;

			default:
				choice='0';

			}
		} }
```