Practical Personalized Genomics in the Encrypted Domain

Kalpana Singh, Renaud Sirdey, Sergiu Carpov

Practical Personalized Genomics in the Encrypted Domain

Kalpana Singh*, Renaud Sirdey†, Sergiu Carpov†
*Kalpana.SINGH@irt-systemx.fr, †{renaud.sirdey, sergiu.carpov}@cea.fr
* IRT SystemX, Paris-Saclay, France
† CEA, LIST, Point Courrier 172, 91191 Gif-sur-Yvette Cedex, France

***Abstract*—In this paper, we examine and propose a solution for the challenges of sharing of genome sequence data and of data querying on the genome sequence data on a cloud server in personalized medicine scenarios. We develop a privacy-preserving, a secure and efficient solution for personalized medicine. The solution that we propose making use of stream cipher-based homomorphic transciphering in a cloud server, and to show the effectiveness of transciphering solution in the personalized medicine scenario. This paper also provides the comparative analysis of well-known existing homomorphic encryption solutions BGV and FV schemes combined with the FLIP stream cipher to demonstrate the efficiency and privacy of our solution.**

***Keywords*—Homomorphic encryption, Stream cipher, Transciphering, Personalized medicine**

I. Introduction

Genomics is an emerging field facing multiple challenges for hosting, sharing, computing on, and interacting with the large data sets. The increasing availability of genome data is accompanied by increasing privacy concerns, such that an inappropriate disclosure of such data might put individuals at risk.

Cloud computing is becoming the preferred solution for efficiently dealing with data storage and analysis with the increasing amount of Genome Sequence Data (GSD). The cloud is providing the power of analytics to companies of every size and companies utilizing analytics are enjoying a significant advantage in the marketplace. Cloud computing is creating the vast new opportunities and dismantling entire approaches to business [28], [20]. Cloud computing has the potential to address the interoperability challenges currently present in health IT systems and to be the technical standard that enables individuals, health care providers, health care entities and medical researchers to securely share electronic health data [19]. Privacy regulations and concerns about the risks of leaking sensitive GSD add another layer of complexity to the problem. This calls for new highly developed techniques that ensure data protection from an untrusted cloud server and still enables healthcare persons/researchers to obtain useful information.

One effective way of addressing these concerns is to store genome data in an encrypted form in the untrusted cloud. Unfortunately, computing on encrypted data is notoriously difficult, often requiring highly refined cryptographic techniques to protect genome data when their storage and processing is outsourced to an untrusted cloud server such as homomorphic encryption (HE) scheme. The HE scheme can be used to encrypt data in such a way that storage can be outsourced to an untrusted cloud, and the data can be computed on in a meaningful way in encrypted form, without access to decryption keys. This scheme enables more flexible scenarios and functionalities. It requires fewer interactions, thereby reducing the communication complexity.

A. Homomorphic Encryption Solutions for Genomic Computations

The HE scheme is attracting attention as a tool for secure outsourcing of data analysis. Since the seminal work of Gentry [11], introducing the first fully homomorphic encryption (FHE) many other simpler and more efficient schemes have been proposed [10], [6], [5]. The HE based solutions have been applied to secure outsourcing of computation that involves genome data. Lauter et al. [16] demonstrated an approach to conducting private computation using encrypted genome data with FHE, and presented several statistical algorithms can be carried out on encrypted genomes by using somewhat HE (SHE) scheme. Unfortunately, these cryptographic solutions are not efficient in terms of time and space to conduct a genome-wide association study (GWAS)-scale computation.

Similarly, Bos et al. [3] proposed a working implementation of cloud service for private computation of encrypted health data using FHE. They give optimized performance numbers for HE and a particular application in health care to predictive analysis, along with an algorithm for automatically selecting parameters. In a similar line of research, Cheon et al. [15] adopted an SHE scheme to ensure the security of shared data in a cloud server. They only focused on computing the edit distance. Wang et al. [26] designed an SHE based technique to discover the rare disease variants and analyze the disease susceptibility in an untrusted cloud environment. McLaren et al. [17] proposed a technique based on the additive HE scheme for securely performing pharmacogenetic tests on the encrypted genomes.

B. Identified Gaps and Solution to fill the Gap for our Work

We identify shortcomings on uses of HE schemes on genome data analysis in the Subsection I-A which respectively focus on three main areas in this paper. The computational cost, memory cost, and noise increase are the limitations for the deployment of cloud services based on such HE schemes, and limits scalability to real-size genome data sets. The memory

This work was carried in part while the first author was a postdoctoral researcher at CEA, LIST, Point Courrier 172, 91191 Gif-sur-Yvette Cedex, France.

cost is mostly influenced by the homomorphic ciphertexts and public key sizes. These limitations make homomorphic solutions impractical while also resulting in unsatisfactory query privacy. Thus, our objective is to improve the efficiency and reduce the complexity of encryption, evaluation, and decryption algorithms.

To address these problems, we utilize the FLIP stream cipher [18] which is specifically designed to be combined with the HE scheme to improve the efficiency of HE frameworks. The main design principle of the FLIP stream cipher is to filter a constant key register with a time-varying public bit permutation, which allows for small and constant noise growth. The constant noise growth outcome of the FLIP is very useful to increase the performance of HE schemes. We use the "transciphering" [6] from FLIP to HE schemes to get a constant and small(er) noise (thus we get compressed ciphertext bits) in a cloud server. This is a very efficient solution for securely outsourcing genome data storage and processing.

C. Contribution

We propose an architecture for enhanced privacy protection of sensitive genome sequence data on the cloud server in the case of personalized medicine. We summarize the key contributions of this paper as follows:

1) A privacy-preserving homomorphic evaluation of a stream cipher is utilized to store and later compute on genome sequence data in an untrusted cloud server.
2) Transciphering towards to HE solution is adopted for reduced ciphertext bits which are used for efficient memory storage, low computation and communication costs.
3) Real-world genomic rules are generated, implemented, and performed testing for personalized medicine solution.
4) A comprehensive comparison study is performed on existing homomorphic solutions (BGV and FV) and our utilized transciphering schemes.

D. Paper Organization

The rest of the paper is organized as follows. Section II presents the basis for our architecture and demonstrates well-known existing HE schemes for comparison in experimental analysis. In Section III, we describe the definitions and notations for subsequent parts of the paper. Section IV provides descriptions of our proposed architecture and workflow processes between the entities of our architecture. In the experimental Section V, we present our experimental setup in Subsection V-A. Genome data sets and genomic rules are exhibited in Subsection V-B. Section VI demonstrates our computational results and comparative analysis. Section VII summarizes and presents conclusions.

II. Preliminaries

We provide background information about the FLIP stream cipher and Armadillo compilation chain which are used in our work. We also mention the brief information of BGV and FV schemes which are used in the experimental analysis for the comparative study.

a) FLIP Scheme.: FLIP is a new family of stream ciphers and specifically designed to be combined with the HE scheme to improve the efficiency of HE frameworks. This homomorphic-friendly design requires to drastically reduce the multiplicative depth of the decryption circuit. This achievement is made possible by the use of a new construction called as filter permutator [18]. Its operational principle is represented in Figure 2 of the paper [18]. Security analysis of this scheme is presented in the paper [18].

b) Armadillo.: The Armadillo compilation chain [7] provides an easy to use a compiler which builds a privacy-preserving binary for an application written in a high-level language using homomorphic encryption as a back-end. Armadillo aims at addressing the software engineering issues of cost-effectively writing programs for execution over encrypted data and automatically handling a large amount of parallelism in order to lead to non-prohibitive performances.

c) The BGV Scheme.: Gentry, Halevi and Smart [12] constructed an efficient BGV-type SHE scheme. The security of this scheme is based on the (decisional) Ring Learning With Errors (RLWE) assumption. The Brakerski-Gentry-Vaikuntanathan (BGV) scheme [4] stands today as one of the most efficient somewhat FHE scheme. The BGV scheme is more efficient for large plaintext moduli [9]. We use HElib software library for our experimental analysis (see in Section VI), which implements the BGV scheme, along with many optimizations to make homomorphic evaluation run faster.

d) The FV Scheme.: The FV scheme is defined in the paper [10]. The security of this scheme is based on the hardness of the "Ring-Learning With Errors" (R-LWE) problem. In the FV scheme, besides the public and private keys, a relinearization key is generated to be used during multiplication on ciphertexts in order to reduce the noise. For more details about this scheme, we refer the reader to the paper [10]. The FV scheme is implemented in the back-end of Armadillo compiler [7] without the bootstrapping step.

III. Notations and Definitions

We briefly describe the notations and definitions for subsequent parts of the paper. Let M be the plaintext space, C the ciphertext space, and λ the security parameter. H denotes an HE scheme which consists of four algorithms: $H.KeyGen(1^\lambda)$, $H.Enc(m, pk^H)$, $H.Dec(c, sk^H)$, $H.Eval(f, c_1, ..., c_k, pk^H)$. Similarly, F denotes a FLIP scheme which consists of three algorithms: $F.KeyGen(1^\lambda)$, $F.Enc(m, sk^F)$, $F.Dec(c, sk^F)$, where $m \epsilon M$ and $c \epsilon C$.

a) Homomorphic Encryption Scheme Definition:

- $H.KeyGen(1^\lambda)$. Output pk^H and sk^H the public and secret keys of the HE scheme.
- $H.Enc(m, pk^H)$. From the plaintext $m \epsilon M$ and the public key pk^H, output a ciphertext $c \epsilon C$.
- $H.Dec(c, sk^H)$. From the ciphertext $c \epsilon C$ and the secret key sk^H, output a $m^{'} \epsilon M$.
- $H.Eval(f, c_1, ..., c_k, pk^H)$. With $c_i = H.Enc(m_i, pk)$ for $1 \leq i \leq k$, output a ciphertext $c_f \epsilon C$ such that $H.Dec(c_f) = f(m_1, ..., m_k)$.

b) FLIP Scheme Definition: We utilize the definition of FLIP family of stream ciphers [18] in our architecture. The algorithms of the FLIP scheme are defined as:

- $F.KeyGen(1^{\lambda})$. Output sk^F secret key of the FLIP scheme.
- $F.Enc(m, sk^F)$. From the plaintext $m \in M$ and the secret key sk^F, output a ciphertext $c \in C$.
- $F.Dec(c, sk^F)$. From the ciphertext $c \in C$ and the secret key sk^F, output a $m^{'} \in M$.

The main feature of the filter permutator model, considering HE settings, is that it allows handling ciphertexts having the same constant and small amount of noise, whatever the number of output bits [18].

IV. Our Architecture

Our architecture is designed to provide a solution for personalized medicine in a secure and privacy-preserving manner on an untrusted cloud server. In the setup, we use five entities: Patient, Hospital, a Sequencing facility (SF), a Cloud server (CS), and Medical team/Doctor (MT/D). Our full architecture includes several patients, hospitals, and medical teams/doctors. To simplify the explanations, we present only one patient, one hospital, one MT/D as shown in Figure 1.

In our architecture, each patient belongs to the hospital for medical treatment. A hospital has sensitive biosamples of each patient to examine the medical problems and get a solution. Hospital sends biosamples of each patient to the trusted SF. The SF sequences, encrypts genome sequence data (GSD), and sends encrypted genome sequence data (e-GSD) to a CS in a secure, privacy preserve, and efficient manner. Now, the CS receives e-GSD and stores in the data repository. The e-GSD is a useful data to get personalized medicine solution for the corresponding patient.

The MT/D is mainly interested in receiving a portion of patient's genome data to get a solution for personalized medicine. The MT/D sends a query request using an HE algorithm and cooperation with a patient (Steps 3 and 4 in Table II). Patient encrypts own secret key using HE algorithm and sends to the CS. This is an essential step for transciphering. Additionally, this is an independent step and can be performed in an advance by the patient. The CS performs homomorphic evaluation algorithm using transciphering to process query request. Furthermore, MT/D uses a pre-defined knowledge in its query request. For example, MT/D knows the CS has storage of HLA or ABO genome data sets of patients. These observations are acknowledged as a pre-defined knowledge for MT/D. However, MT/D does not know the content of GSD of a patient. We reduce the communication cost using the pre-defined knowledge. The CS receives a query request, uses a transciphering method (Step 5 in Table II), and computes a requested data in an encrypted form. This allows to compute the data in a very less size compare to the FHE scheme computed data. This computed data is known as the compressed ciphertext bits and sends to the MT/D as a query response. The MT/D decrypts an encrypted query response and gets a result to investigate personalized medicine solution.

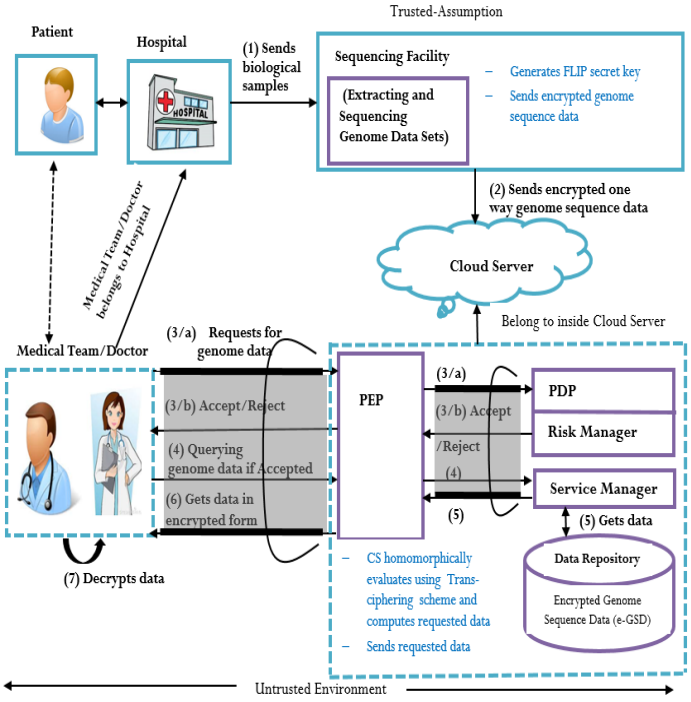


Fig. 1: Our Architecture: A Personalized Medicine Scenario

The communication steps of each entity are illustrated in detail as follows:

The Communication Steps. At the CS end, the service manager, a data repository, policy enforcement side (PEP), and policy decision point (PDP) are used for each communication step. We utilize a separation of the PDP from the PEP as described in [23], along with the introduction of an insider threat detection unit at the PDP side in our scheme to prevent PEP-side caching. The authors of [21] also use this separation to prevent the insider attack (see their Figure 4). The service manager is responsible for each process (send and receive) on the CS, in which policies are established using PEP and PDP, and decisions are made on the basis of these policies.

a) Step.1: Key generation Set-up (Initialization step): The SF runs the FLIP key generation algorithm $F.KeyGen(1^{\lambda})$, and gets the secret key sk^F. Now, the SF sends sk^F value to the corresponding patient. The MT/D uses an HE scheme and obtains pk^H, sk^H public and secret HE keys respectively. This step is defined in Table I.

TABLE I: Communication between SF, CS, and MT/D

Steps	SF	CS	MT/D
1.	$sk^F \longleftarrow F.KeyGen(1^{\lambda})$		$(sk^H, pk^H) \longleftarrow H.KeyGen(1^{\lambda})$
2.	$C^F(M) = F.Enc(M, sk^F)$	$\xrightarrow{C^F(M)}$	$C^F(M)$

b) Step.2: Communication between the SF and the CS: In this step, the SF sequences genome data and gets the

GSD as illustrated in Table I. Then, the SF performs FLIP encryption using sk^F on GSD (M), and gets cipher value of GSD: $C^F(M) = F.Enc(M, sk^F)$. The SF sends $C^F(M)$ to the CS, and CS stores this value in a data repository for future use. The FLIP scheme provides a better efficiency for both storage and computation costs (Table VII of Subsection VI-B).

c) Steps.3 and 4: Communication between the MT/D, a Patient, and the CS: In this step, an MT/D sends pk^H to a patient. A patient computes homomorphically ciphertext value of sk^F using this equation $C^H(sk^F) = H.Enc(sk^F, pk^H)$, and sends $C^H(sk^F)$ and pk^H to the CS. This is a necessary step to request data query as MT/D is unauthorized to access the sk^F. The CS can not send query result to an unauthorized entity. This step is an essential step for transciphering in Step 5 and can be computed in an advance as a result of an independent step. This step confirms the MT/D is a part of the hospital. These steps are described in Table II.

d) Steps.5 and 6: Communication between the CS, and the MT/D: Now, the CS has a data query from the MT/D. As we mentioned in the above paragraph of this section, an MT/D knows the pre-defined knowledge of an e-GSD, which are stored on the CS.

TABLE II: Communication between CS, MT/D, and Patient

Steps	CS	MT/D	Patient
3.		$pk^H \xrightarrow{pk^H}$	pk^H
4.	$C^H(sk^F), pk^H \xleftarrow{C^H(sk^F), pk^H}$		$C^H(sk^F)$ = $H.Enc(sk^F, pk^H)$
5.	$\widetilde{C}^H(f(M))$ = $H.Eval(f, (F.Dec(C^F(M))), C^H(sk^F), pk^H)$		
6.	$\widetilde{C}^H(f(M)) \xrightarrow{\widetilde{C}^H(f(M))}$	$\widetilde{C}^H(f(M))$, decrypts $f(M) = H.Dec(\widetilde{C}^H(f(M), sk^H)$	

In our case, we use the HLA, and the ABO data sets (detail in Subsection V-B). The pre-defined knowledge is applicable in our case for speed-up communication. The CS has these values $C^F(M)$, $C^H(sk^F)$, and pk^H from previous Steps 2 and 4. In step 5, the CS evaluates the FLIP algorithm $C^F(M)$ homomorphically using HE encrypted FLIP private key $C^H(sk^F)$ (Transciphering from FLIP to HE is used as defined in the paper [8]). The CS obtains a compressed result of an e-GSD $\widetilde{C}^H(f(M))$, and sends to the MT/D (Step-6). This compression is achieved using homomorphic evaluation of FLIP stream cipher scheme. In step 6, the MT/D decrypts data using HE secret key sk^H and gets the requested data. By using transciphering from FLIP to HE scheme, we get the compressed ciphertext bits $\widetilde{C}^H(f(M))$. This reduces the cost of storage on the CS and efficiently transmits ciphertext bits. We find the existing HE schemes (see in Section I) suffer from a very large ciphertext expansion. The transmission of ciphertext between the CS and an MT/D (in our architecture) is, therefore, a very significant bottleneck in practice. In our architecture, we reduce the size of ciphertext in a very efficient manner by using homomorphic evaluation of the FLIP stream cipher scheme. Steps 5 and 6 are characterized in Table II.

V. Experimental Setup and Data sets

We present the experimental setup in Subsection V-A. Our genome data sets and genomic rules are described in the Subsection V-B.

A. Experimental Setup

The BGV [12] and the FV [10] schemes are implemented in an HElib library [14], [13] and Armadillo [7] respectively. The FLIP stream cipher is implemented in both HElib and Armadillo for transciphering and comparative analysis with the BGV and the FV schemes. These experiments are performed on an Ubuntu 14.04 virtual box which is running on windows 7 with Intel (R) Core (TM) i5-5300U CPU, 2.30 GHz processor, and RAM 16 GB. We present performance costs in terms of the number of boolean operators (ANDs and XORs), computation times (encryption, evaluation, and decryption), and multiplicative depth (MD) in the Tables IV, V, VI, and VII. Number of ANDs and XORs are denoted as the "♯ANDs" and"♯XORs" respectively in Table IV. The AND and XOR gates are exhibited as multiplicative and additive gates respectively.

B. Genome Data sets and Genomic Rules

We utilize the publicly available data set Blood Group Antigen Gene Mutation Database (BGMUT), which is an online repository of the allelic variations in genes that determine the antigens of various human blood group systems [22]. The data set is in VCF file format. The VCF file has four parameters (chromosome, position, reference chromosome, alternate allele). ABO genomic rules are described in the paper [24]. A total of 2504 patients are included for each study in the analysis of ABO and HLA rules.

1) ABO Genomic Rule: We use ABO genomic rules from the paper [24]. ABO rules are classified into two parts: ABO-1 and ABO-2. ABO-1 is presented in Table 2 of the Paper [24]. Details of both ABO rules are presented in detail on pages 7 and 8 of the Paper [24]. Authors have implemented both rules using HElib library in the Paper [24]. We use their results (Tables 3 and 4 from the Paper [24]) in our experimental results for discussions and comparative analysis. Table IV delineates the number of AND and XOR operators for both ABO-1 and ABO-2 genomic rules.

2) HLA Genomic Rule: HLA complex helps the immune system and distinguishes the body's own proteins from proteins made by foreign invaders such as viruses and bacteria. HLA genes have many possible variations. For example, in our case, 50 very similar haplotypes for $G*01:01:01:01$ phenotype, 49 similar haplotypes for $G*01:01:01:02$ phenotype etc. The haplotypes of $G*01:01:01:01$ phenotype of HLA are designated as $G*01:01:01:01/G*01:01:01:01$ ..... $G*01:01:01:01/G*01:01:01:06$, $G*01:01:01:01/G*01:02$ ..... $G*01:01:01:01/G*01:18$. Similarly, we determine haplotypes of different phenotypes in our analysis of HLA rules. Table III exhibits the example of

TABLE III: HLA Genomic Rules on the basis of Haplotype, Phenotype, Chromosome position and Nucleotides

Phenotype	Haplotype	Rules using Chromosome positions and Nucleotide
$G*01:01:01:01$	$G*01:01:01:01$, $G*01:01:01:02$, $G*01:01:01:03$, $G*01:01:01:04$, $G*01:01:01:05$, $G*01:01:01:06$, $G*01:01:02:01$, $G*01:01:02:02$, $G*01:01:03:01$, $G*01:01:03:02$, $G*01:01:03:03$, $G*01:01:04$, $G*01:01:05$, $G*01:01:06$, $G*01:01:07$, $G*01:01:08$, $G*01:01:09$, $G*01:01:11$, $G*01:01:12$, $G*01:01:13$, $G*01:01:14$, $G*01:01:15$, $G*01:01:16$, $G*01:01:17$, $G*01:01:18$, $G*01:01:19$, $G*01:01:20$, $G*01:01:21$, $G*01:02$, $G*01:03:01:01$, $G*01:03:01:02$, $G*01:04:01$, $G*01:04:02$, $G*01:04:03$, $G*01:04:04$, $G*01:04:05$, $G*01:05N$, $G*01:06$, $G*01:07$, $G*01:08$, $G*01:09$, $G*01:10$, $G01:11$, $G*01:12$, $G*01:13N$, $G*01:14$, $G*01:15$, $G*01:16$, $G*01:17$, $G*01:18$	Rules for haplotype $G*01:01:01:01/G*01:01:01:01:02 \rightarrow$ $(6:29795720;A;G;1\|1)$ & $(6:29795747;G;C;1\|1)$ & $(6:29795751;C;T;1\|1)$ & $(6:29795768;T;C;0\|0)$ & $(6:29795855;C;G;0\|0)$ & $(6:29795983;A;G;1\|1)$ & $(6:29795987;G;A;0\|0)$ & $(6:29795993;G;A;0\|0)$ & $(6:29795914;C;T;0\|0)$ & $(6:29795918;G;A;0\|0)$ & $(6:29795927;G;A;0\|0)$ & $(6:29796029;C;T;0\|0)$ & $(6:29796327;C;T;0\|0)$ & $(6:29796348;C;T;0\|0)$ & $(6:29796399;C;G;0\|0)$ & $(6:29796436;C;.;0\|0)$ & $(6:29796523;T;C;0\|0)$ & $(6:29796555;C;T;0\|0)$ & $(6:29797280;G;A;0\|0)$ & $(6:29797421;G;A;0\|0)$ & $(6:29798459;C;G;1\|0)$

50 haplotypes of $G*01:01:01:01$ phenotype. Due to page limitation, we are not able to present genomic rule of each phenotype and haplotype in this paper. The first column of Table III denotes the phenotype of HLA $(G*01:01:01:01)$, the second column denotes 50 haplotypes of $G*01:01:01:01$ phenotype. The third column presents the rule of HLA haplotype $(G*01:01:01:02)$ in correspond to phenotype $G*01:01:01:01$. We choose $G*01:01:01:02$ haplotype because of smaller descriptions of chromosome position with nucleotide than the other haplotype descriptions (due to page limitation). In the third column, if the HLA-rule is satisfied then a patient has the presence of this antigen $G*01:01:01:01/G*01:01:01:02$ correspondingly. The output of the values is in two forms "0" or "1". "0" means corresponding haplotype/phenotype does not find on the patient genome and "1" means presence on the genome of a patient. On the third column, "6" indicates chromosome, followed by a separator ":" and an integer number "29795720" that corresponds to the position at the reference chromosome. The ";" is separator followed by "A" which is the nucleotide at the reference chromosome, followed by a separator ";" then the value "G" which indicates the nucleotides of the alternative allele, $1|1$ means that both alleles have that specific polymorphism and thus the individual is homozygote for that polymorphism. Similarly, we can define other values of the haplotype with a definition of $0|0$ means that both alleles of an individual have no change in the nucleotide compared to a reference chromosome. $0|1$ means that one of the alleles has a polymorphism at that specific chromosome position. "&" denoted as "AND" operator, "|" denoted as "OR" operator in homomorphic calculations. In total, we implemented and analyzed 1274 HLA rules. The number of AND and XOR operators for all HLA rules are presented in Table IV.

VI. Experimental Results and Analysis

This section presents the computational results and comparative analysis of our experimental work. We implement and test BGV, FV, and FLIP schemes. Additionally, we perform transciphering (FLIP to HE schemes) to compute homomorphic evaluation of the FLIP algorithm. We use Armadillo [7] to test FV scheme, and HElib is used for BGV. We compute results of FLIP on both HElib and Armadillo for transciphering and comparative analysis with the FV and BGV schemes.

A. Calculation of Boolean Operators

TABLE IV: Boolean Operators on HElib and Armadillo for ABO-1, ABO-2, HLA Genomic Rules

Genomic Rules	Number of Rules	Boolean Operators on HElib and Armadillo		
		♯ANDs	♯XORs	Total Number of Operators
ABO-1	15	100	58	158
ABO-2	16	3156	20132	23288
HLA	1274	13481	7040	20521

The number of boolean operators is the important parameter to calculate a multiplicative depth for each genomic rule in our experimental analysis. AND and XOR boolean operators are used in our genomic rules. AND denotes multiplication and XOR denotes addition operations in our genomic rule evaluation. The descriptions of these boolean operators for each rule is mentioned in Table IV.

B. Computation Costs of FV, BGV, Transciphering from FLIP to FV, and Transciphering from FLIP to BGV Schemes

The FV and transciphering from FLIP to FV schemes are implemented in C++ and compiled using Armadillo compilation chain. We set the security parameter to 128-bit for both schemes. Other parameters of the schemes are derived automatically following the procedure from the paper [7]. Armadillo addresses optimizing compiler and parallel runtime environment to minimize computation cost in an HE execution [7]. In our experimental analysis, we did not consider parallelism optimization technique [7] during evaluation of both FV and transciphering from FLIP to FV schemes in Armadillo. In case of parallelization, we can execute the circuits in parallel, for example, 8 threads. The number of parallel execution threads is used to minimize computation cost in an HE execution. Still, we have not employed parallelism in our experimental evaluation, and no batching. ABO-1, ABO-2, and HLA genomic rules are implemented on Armadillo in both FV and transciphering from FLIP to FV schemes. The computation costs of these genomic rules are illustrated in the Tables V, VI, and VII. The presented results are the measures of the actual compilation times for both FV and transciphering from FLIP to FV schemes.

The BGV and transciphering from FLIP to BGV schemes are implemented in HElib [14]. The same ABO-1, ABO-2, and HLA rules are evaluated using BGV and transciphering from FLIP to BGV schemes. The experimental results of both

schemes are mentioned in the Tables V, VI, and VII. We use the results of ABO-1 and ABO-2 genomic rules from Table 4 of the Paper [24]. Additionally, we use the same HElib parameter setting from the Paper [24] in our implementation.

TABLE V: HE Schemes (BGV and FV) Execution time in Seconds

Parameters		KeyGen	Encrypt	Evaluate	Decrypt	MD
Schemes	Genomic Rules					
BGV	ABO-1	2.732	0	3	1	3
	ABO-2	11.976	9	1549	5	13
	HLA	8.964	43	2047	24	7
FV	ABO-1	0	0	1	0	3
	ABO-2	1	6	427	2	13
	HLA	1	29	563	12	6

TABLE VI: HE Schemes (BGV and FV) Execution time in Seconds with additional FLIP Multiplicative Depth

Parameters		KeyGen	Encrypt	Evaluate	Decrypt	MD
Schemes	Genomic Rules					
BGV	ABO-1	14.128	1	27	5	7
	ABO-2	53.692	31	4757	23	17
	HLA	30.948	45	7554	271	11
FV	ABO-1	1	0	5	1	7
	ABO-2	3	18	1231	4	17
	HLA	3	40	1947	17	10

TABLE VII: FLIP Transciphering Evaluation time in Seconds

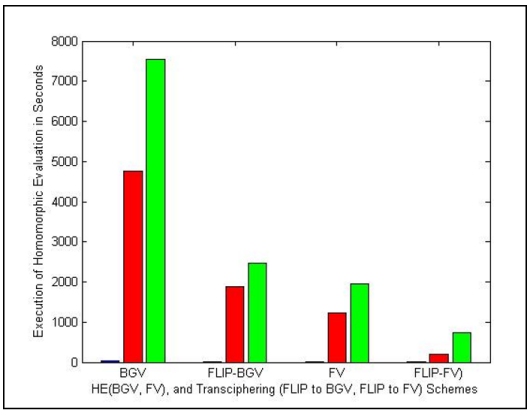
Parameters		Evaluate	MD
Schemes	Genomic Rules		
Transciphering FLIP to BGV	ABO-1	6.76	7
	ABO-2	1893.995	17
	HLA	2468.91	11
Transciphering FLIP to FV	ABO-1	0.467	7
	ABO-2	208.232	17
	HLA	731.132	10

Subsection 6.2.3 of the Paper [24] describes the Multiplicative Depth (MD) in detail. The results of transciphering from FLIP to BGV and FLIP to FV schemes are presented in Table VII. The computation costs of each genomic rule with original MD in FV and BGV schemes are delineated in Table V. The computation costs of each genomic rule with the original MD and the MD of the FLIP are executed using FV and BGV schemes in Table VI. The MD of the FLIP scheme is 4 [18]. For example, the MD of ABO-1 is 3 in Table V, and the value of MD for the same rule is 7 $(3 + 4 = 7)$ in Table VI, where 4 is a MD value of the FLIP. Similarly, we compute MD values for each rule, and execution times on new MD values in the Table VI. We use the addition of the MD of FLIP for each rule to compare the computation costs of each rule with transciphering using FLIP computational results in the Table VII. The computation costs of transciphering from FLIP to FV, and from FLIP to BGV schemes for each genomic rule with similar values of MD are defined in the Tables VI and VII.

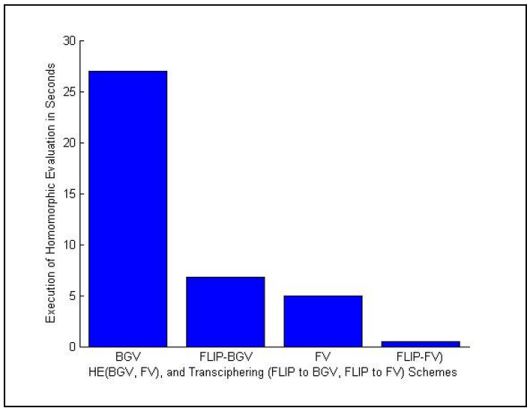
C. Comparative Analysis and Discussions

Comparative analysis of HE (BGV and FV) and transciphering schemes are determined from Tables VI and VII respectively. The FV scheme is conceptually simpler than the BGV scheme as shown in Table VI. Our transciphering computational results are mentioned in Table VII. We use transciphering from FLIP to BGV, and FLIP to FV schemes to prove that the transciphering method is more appropriate to minimize the computation cost in any HE schemes and to compare with both well-known HE schemes.

From Table VI, the evaluation times needed for BGV scheme to compute each genomic rule ABO-1, ABO-2, and HLA are 27, 4757, 7554 seconds respectively. Similarly, the evaluation times needed from Table VI for FV scheme in the execution of ABO-1, ABO-2, and HLA are 5, 1231, 1947 seconds respectively. From the results of Table VII, we can see that the time needed in evaluations of transciphering from FLIP to BGV for each ABO-1, ABO-2, and HLA are 6.76, 1893.995, 2468.91 seconds respectively. Evaluations of transciphering from FLIP to FV for each ABO-1, ABO-2, and HLA are 0.467, 208.232, 731.132 seconds respectively. Thus, their highest evaluation time is achieved on BGV scheme. The graphical comparison of these schemes is depicted in Figure 2a.



(a) Homomorphic evaluation of ABO-1, ABO-2, HLA rules using BGV, FV, and Transciphering (FLIP to BGV, and FLIP to FV) schemes



(b) Homomorphic evaluation of ABO-1 using BGV, FV, and Transciphering (FLIP to BGV, and FLIP to FV) schemes

Fig. 2: Comparative Analysis of BGV, FV, and Transciphering (FLIP to BGV, and FLIP to FV) schemes

Figure 2a illustrates the advantages of using a transciphering method in the HE schemes during homomorphic evaluation step. The Figure 2a delineates the computation cost of the transciphering scheme is better than the both well-known BGV and FV schemes for each genomic rule. Figure 2b depicts the ABO-1 results for each scheme as depicted in the Figure

2a. We put ABO-1 results separately because the graphs are not clear in Figure 2a due to lower values of ABO-1. We find the evaluation time of transciphering from FLIP to FV scheme is efficient than the other schemes. The sequence success of schemes in the homomorphic evaluation are: BGV <Transciphering FLIP to BGV < FV < Transciphering FLIP to FV. In this study, we achieve the FV scheme is better than the BGV scheme. Thus, transciphering from FLIP to FV is better than the transciphering from FLIP to BGV scheme.

VII. Summary and Conclusions

This paper presents an entire secure framework for genome data sets processing leveraging on an untrusted cloud. This study assesses the steps required for deployment of privacy-preserving genetic testing in a personalized medicine scenario. We enable the transciphering within our architecture in order to enhance the storage efficiency of the genome data sets, computation, and communication costs. We utilize a FLIP scheme as the basis of a secure scheme in the well-known BGV and FV schemes to provide a high level of efficiency. Additionally, we perform a comparison study between BGV, FV, and transciphering (FLIP to BGV, and FLIP to FV) schemes to validate the efficiency of transciphering in HE schemes using ABO-1, ABO-2, and HLA rules. The testing results have proven that the transciphering provides efficient results than both BGV and FV schemes. Additionally, we achieve the best efficient results in transciphering from FLIP to FV scheme as FV scheme is efficient than the BGV scheme.

Our next step is to extend our architecture for achieving interoperability and overcoming data exchange issues using the Blockchain [1], [25], [27]. For example, Microsoft provides Blockchain as a Service (BaaS) on the Azure cloud platform. IBM provides Watson IoT platform to manage IoT data in a private blockchain ledger, which is integrated with IBM's business-level cloud services. Edge computing is a promising solution for the blockchain applications [2]. Additionally, our forthcoming work is to enhance the current architecture by utilizing the edge computing in order to enhance the computing power for data analytics and processing.

Acknowledgement

This research work has been partially carried out under the leadership of the Institute for Technological Research SystemX, and therefore granted with public funds within the scope of the French Program Investissements d'Avenir. We also thank François Artiguenave and David Cohen for discussions and for helping us to understand genome data sets and HLA genomic rules.

References

[1] S. Amyx, "Ready for the disruption from edge computing?" IBM Internet of Things Blog, 2016, https://www.ibm.com/blogs/internet-of-things/edge-computing/.

[2] Z. X. andYang Zhang, D. Niyato, P. Wang, and Z. Han, "When mobile blockchain meets edge computing: Challenges and applications," Cornell University Library, 2017, https://arxiv.org/abs/1711.05938.

[3] J. W. Bos, K. Lauter, and M. Naehrig, "Private predictive analysis on encrypted medical data," in *Cryptology ePrint Archive: Report 2014/336*, 2014.

[4] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) Fully homomorphic encryption without bootstrapping," in *ITCS '12*, 2012, pp. 309–325.

[5] A. Canteaut, S. Carpov, C. Fontaine, T. Lepoint, M. Naya-Plasencia, and R. P. Paillier, "Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression," in *FSE 2016*, 2016, pp. 313–333.

[6] A. Canteaut, S. Carpov, C. Fontaine, T. Lepoint, María, Naya-Plasencia, P. Paillier, and R. Sirdey, "How to compress homomorphic ciphertexts," Cryptology ePrint Archive, Report 2015/113, Tech. Rep., 2015.

[7] S. Carpov, P. Dubrulle, and R. Sirdey, "Armadillo: A compilation chain for privacy preserving applications," in *SCC'15, ACM*, 2015, pp. 13–19.

[8] S. Carpov, T. H. Nguyen, R. Sirdey, G. Constantino, and F. Martinelli, "Practical privacy-preserving medical diagnosis using homomorphic encryption," in *CLOUD 2016*, 2016, pp. 593–599.

[9] A. Costache and N. P. Smart, "Which ring based somewhat homomorphic encryption scheme is best?" in *Topics in Cryptology - CT-RSA 2016*, 2016, pp. 325–340.

[10] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," Cryptology ePrint Archive: Report 2012/144, 2012.

[11] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Stanford University, 2015.

[12] C. Gentry, S. Halevi, and N. P. Smart, "Homomorphic evaluation of the AES circuit," in *Advances in Cryptology-CRYPTO 2012*, 2012, pp. 850–867.

[13] S. Halevi, "HElib: An implementation of homomorphic encryption," 2013, https://github.com/shaih/HElib.

[14] S. Halevi and V. Shoup, "Algorithms in HElib," in *Cryptology-CRYPTO 2014*, 2014, pp. 554–571.

[15] C. J. Hee, K. Miran, and L. Kristin, "Homomorphic computation of edit distance," in *Financial Cryptography and Data Security*, 2015, pp. 194–212.

[16] K. Lauter, A. López-Alt, and M. Naehrig, "Private computation on encrypted genomic data," Progress in Cryptology - LATINCRYPT 2014, Tech. Rep., 2014.

[17] P. J. McLaren, J. L. Raisaro, M. Aouri, M. Rotger, E. Ayday *et al.*, "Privacy-preserving genomic testing in the clinic: a model using HIV treatment," *Genetics in Medicine*, vol. 18, pp. 814–822, 2016.

[18] P. Méaux, A. Journault, F.-X. Standaert, and C. Carlet, "Towards stream ciphers for efficient FHE with low-noise ciphertexts," in *EUROCRYPT 2016*, 2016, pp. 311–343.

[19] N. Mohan and J. Kangasharju, "Edge-fog cloud: A distributed cloud for internet of things computations," Cloudification of the Internet of Things (CIoT), 2017, http://ieeexplore.ieee.org/document/7872914/.

[20] M. B. Mollah, A. K. Azad, and A. Vasilakos, "Secure data sharing and searching at the edge of cloud-assisted internet of things," *IEEE Cloud Computing*, vol. 4, no. 1, pp. 34 – 42, 2017.

[21] S. D. Nogoorani and R. Jalili, "TIRIAC: A trust-driven risk-aware access control framework for grid environments," *FGCS*, vol. 55, no. C, pp. 238–254, 2016.

[22] S. K. Patnaik, W. Helmberg, and O. O. Blumenfeld, "BGMUT database of allelic variants of genes encoding

human blood group antigens," *Transfus Med Hemother*, vol. 41, no. 5, pp. 346–351, 2014.

[23] Q.Yaseen, Q. Althebyan, and Y. Jararweh, "PEP-side caching: An insider threat port," in *IEEE 14th International Conference on IRI*, 2013, pp. 137–144.

[24] K. Singh, R. Sirdey, F. Artiguenave, D. Cohen, and S. Carpov, "Towards confidentiality-strengthened personalized genomic medicine embedding homomorphic cryptography," in *ICISSP*, 2017, pp. 325–333.

[25] B. Siwicki, "2018 is primed for blockchain, big data and cloud computing advancements, all with a better security plan," Healthcare IT News, 2017, http://www.healthcareitnews.com/news/2018-primed-blockchain-big-data-and-cloud-computing-advancements-all-better-security-plan.

[26] S. Wang, Y. Zhang, W. Dai, K. Lauter, M. Kim, Y. Tang, H. Xiong, and X. Jiang, "HEALER: Homomorphic computation of ExAct Logistic rEgRession for secure rare disease variants analysis in GWAS," *Bioinformatics*, vol. 32, no. 2, pp. 211–218, 2016.

[27] Q. Xia, E. B. Sifah, A. Smahi, S. Amofa, and X. Zhang, "BBDS: Blockchain-based data sharing for electronic medical records in cloud environments," *Information*, vol. 8, no. 44, 2017.

[28] YuanAi, MugenPeng, and KechengZhang, "Edge cloud computing technologies for internet of things: A primer," Digital Communications and Networks, 2017, https://www.sciencedirect.com/science/article/pii/S2352864817301335?via\%3Dihub.