クラウド環境におけるゲノム秘匿検索に向けた完全準同型暗号ライブラリの比較と分析

山田 優輝[1]　小口 正人[1]

概要：近年ヒトゲノムの解析と応用が可能になり，特に医療分野でのゲノムデータ利用の実用化が注目されるようになった．特にバイオインフォマティクスの研究において頻繁に行われるマッチ判定演算について，各医療機関が保持するゲノムデータに研究者などがアクセス出来るアプリケーションを実現するためには，ゲノムデータをクラウドに保持し，利用者の問い合わせに応じてクラウドで演算を行う委託システムが望ましいと考えられている．この委託システムにおいてクラウドと利用者との間で相互にデータを秘匿するためには，データの暗号化処理が必須となり，クラウドに秘密鍵を渡さずに演算を行うために，従来の共通鍵暗号方式ではなく暗号化されたデータ同士での演算が可能な完全準同型暗号（FHE: Fully Homomorphic Encryption）を用いる秘匿検索手法やその高速化が研究されている．完全準同型暗号を用いることで暗号文同士での加算と乗算がどちらも可能となるが，暗号文同士での演算は計算量が大きく，クラウド側での演算実行時間が課題となっている．本研究では，先行研究で提唱されている二種類のシステムデザインについて，それぞれを異なる二種類の暗号スキーム及び暗号ライブラリを用いて実装し，クラウド上で行われる完全準同型暗号演算の実行時間の比較・分析を行う．

Comparison and Analysis of Fully Homomorphic Encryption Libraries for Genome Secure Search on the Cloud Environment

YUKI YAMADA[1]　MASATO OGUCHI[1]

1. 研究背景

ヒトゲノム計画 [1] の完了や 1000 人ゲノムプロジェクト [2] によるデータベースの公開などとともに，様々な分野でゲノムデータ利用の実用化が注目されるようになった．ゲノムデータを統計処理する際には大型のストレージと処理能力の高い計算機が必要になるため，処理したいゲノムデータをクラウドに預け，利用者の問い合わせを受けてクラウドで演算を行うゲノムデータ委託システムが今後普及していくと考えられる．この際，各医療機関や研究機関にとって所持・管理しているゲノムデータを完全に公開することは難しく，またクライアントとなる研究者は自身の未発表の研究内容を外部に知られたくはないため，相互にプライバシを保護するための暗号化処理が必須となる．

特にバイオインフォマティクスの研究ではゲノムデータの特定の位置に特定の文字列が存在するか否かの判定演算が良く行われるため，これを安全に行うためのシステムが求められる．クライアント・サーバ型のシステムで従来の共通鍵暗号方式を用いる手法を考えると，演算を委託するためにはクラウドに秘密鍵を渡さなければならず，データの漏洩時にはこの鍵を用いて復号されてしまうため適さない．また，暗号文同士での加法演算が成立する加法準同型暗号による暗号化も挙げられるが，複雑な演算が困難なために，演算結果からサーバ側のデータ漏洩を防ぐことは難しいと考えられている [3].

これに対して先行研究 [4]-[5] では，暗号化方式として暗号化されたデータ同士での演算が可能な完全準同型暗号（以下 FHE: Fully Homomorphic Encryption）を用いる秘匿検索手法が提案されている．離散データ構造 Positional-Burrows Wheeler Transform (PBWT)[6] の利用や計算手順の最適化

[1] お茶の水女子大学
Ochanomizu University

などによるアルゴリズムの高速化が進められているが，依然としてサーバ側での負荷が大きくなりやすく課題は解決されていない．一方で，完全準同型暗号演算自体の高速化も盛んに研究されており，これを実装した複数の暗号ライブラリがオンラインで公開されている．

このゲノム秘匿検索システムを実現する際には，システムデザインやアルゴリズムだけでなくデータ構造や暗号スキーム，また暗号ライブラリなど様々な要素が影響し合うため，これらを総合的に検討・評価する必要がある．そこで本研究では，先行研究で提唱されている二種類のシステムデザインについて，それぞれを異なる二種類の暗号スキーム及び暗号ライブラリを用いて実装し，クラウド上で行われる完全準同型暗号を用いた検索演算の実行時間の比較とその分析を行う．

2. アプリケーション

ゲノム秘匿検索アプリケーションの目的は，ゲノムデータベースに対して問い合わせを行い，クエリとデータベース間でのマッチの有無について検索結果を得ることである [3]．ゲノムデータは四種類のヌクレオチド – A, C, G, T – から構成される配列である [7] ため，ゲノム秘匿検索は 4 種に限定された文字列検索とみなすことが出来る．図 1 はこのアプリケーションが行う操作について示している．

図 1: ゲノム検索演算

(1) ユーザは検索したい文字列とその文字列の検索開始位置（ポジション）とを指定する．
(2) クエリ文字列とゲノムデータベースとの間で指定された検索開始位置からマッチ検索を行う．
(3) ユーザはマッチが存在したかどうかの検索結果を得る．

これを 1 章で述べた通りクライアント・サーバ型のシステムにすると，図 2 に示されるようなアプリケーションとなる．

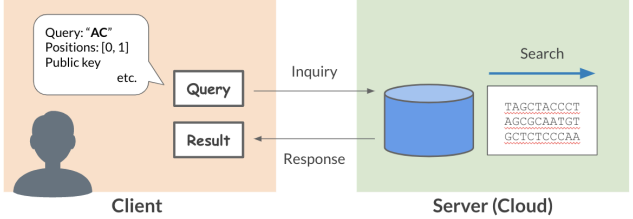

図 2: アプリケーション概要

(1) クライアントは検索したい文字列とその文字列の検索開始位置（ポジション）とをクエリとしてサーバに送る．
(2) サーバはクエリ文字列とゲノムデータベースとの間で指定された検索開始位置からマッチ検索を行い，クライアントに結果を送る．
(3) クライアントはマッチが存在したかどうかの検索結果を得る．

データベース内でゲノム配列データをサンプルごとに並べることによって，各サンプルを特定のポジションから検索することが可能となる．また，クライアントはダミーを含む複数のポジションを指定することで，実際に用いるポジションを秘匿することが出来る．

3. 完全準同型暗号

以下の式 (1) のように暗号文同士での加算が成立する性質を加法準同型性，また式 (2) のように暗号文同士での乗算が成立する性質を乗法準同型性と言う．

加法準同型性，乗法準同型性

$$Encrypt(m) \oplus Encrypt(n) = Encrypt(m+n) \quad (1)$$

$$Encrypt(m) \otimes Encrypt(n) = Encrypt(m \times n) \quad (2)$$

完全準同型暗号 FHE はこの両方の性質を持ち合わせた暗号化手法である．FHE を用いることで，ユーザは平文上で行うのと同様に暗号文同士での加法演算・乗法演算を行うことが出来る．

3.1 特徴

FHE の概念自体は公開鍵暗号が考案された当初である 1978 年に Rivest らによって提唱されていた [8] が，実装はその 31 年後である 2009 年になってから Gentry によって提案された [9]．この実装は多項式環やイデアル格子を応用した暗号スキームであり，読解困難性を保つために，暗号文は平文を暗号化したものにランダムなノイズを加えた形式で表現される．しかしこの手法を用いた場合，1bit の平文を暗号化するとその暗号文は 1GB 程にもなってしまうなど，当時は計算量の大きさから実用性がないとされていた．近年ではより効率の良いスキームやその実装について多くの研究がなされており，実用化への期待が高まっている．

問題点としては計算量が大きいことの他に，暗号文に含まれるノイズが演算の度に増加し，閾値を越えると復号することが出来なくなる，というものが挙げられる．特に乗算を行った際のノイズの増加が著しいため，暗号文に対する乗算操作の演算回数を限定した Somewhat Homomorphic

Encryption (SHE，SwHE) の範囲内で，一度の乗算と複数回の加算によって計算することが出来る分散や相関，ベクトルの内積等の演算を行うことも研究されている [10]．また，bootstrap と呼ばれるノイズをリセットする手法の導入を行うことで演算回数の限定は解決することが出来るが，計算量が非常に大きくなるなど難点は残る．

3.2 ライブラリ

暗号スキームの研究が進むに連れて，多くのオープンソースの暗号ライブラリが幅広い用途に向けてオンラインで公開されるようになった．

HElib[11] は初期に公開されたよく知られているライブラリの一つである．C++で実装され, bootstrap をサポートしている．スキームについては Brakerski-Gentry-Vaikuntanathan (BGV) スキーム [12] を様々な最適化とともにサポートしており，また，最近のバージョンでは CKKS スキーム [13] もサポートしている．

PALISADE[14] はより新しく公開された一般的な格子暗号向けのオープンソースライブラリである．多くのスキームをサポートしており，現在は LTV[15] と Stehle-Steinfeld[16]，BFV[17] と BGV[12] が提供されている．また，HElib とは異なり，PALISADE は現在 bootstrap をサポートしていない．

SEAL[18] や TFHE[19] など他にも多くの暗号ライブラリが公開されているが, 本研究では上述の HElib と PALISADE を用いる．

4. 先行研究

本章ではゲノム秘匿検索に関する複数の先行研究について概要を述べる．

4.1 PBWT-sec

PBWT-sec[3] は加法準同型暗号と Positional-Burrows Wheeler Transform (PBWT)[6] と呼ばれる離散データ構造とを組み合わせた効率的なクライアント・サーバ型のマッチカウント向けプロトコルである．PBWT はゲノムデータをサンプルごとに並べた離散データ構造であり，その並べられたサンプルデータを用いて効率よくマッチ検索することが出来る．PBWT-sec のサーバはゲノム配列データベースを PBWT に変換して保持し，PBWT から再帰的に得られる look-up vector を用いて検索を行う．これは recursive oblivious transfer (ROT)[6] と呼ばれる．クライアントから送られてきたクエリの文字列長が l であるとき，ROT は l 回 look-up vector にアクセスする．この際クライアントとサーバとの間では l 往復の通信が必要となる．また，PBWT-sec では，クライアントはサーバに対してダミーを含んだ複数の検索開始位置（ポジション）を指定することが出来る．これによりクライアントはサーバに対して実際に用いるポジションを秘匿することが可能となる．

PBWT 及び PBWT-sec を C++で実装したものはそれぞれ GitHub 上で公開されている [20]-[21].

4.2 FHE を用いたゲノム秘匿検索

PBWT-sec に FHE を導入したゲノム秘匿検索システムについても複数の先行研究が存在する．FHE を用いた場合，加法準同型暗号よりも多くの計算コストを要するが，FHE を導入することで PBWT-sec の手法をワイルドカード検索など様々なアプローチにも適用することが可能となる．本研究においては石巻らによる以下の二つの先行研究をもとに，システムデザインや用いる暗号スキーム及び暗号ライブラリについて議論・分析を行う．

一つ目の先行研究 [4] では，PBWT-sec と同様にクエリの文字列長だけクライアント・サーバ間での通信が必要となるゲノム秘匿検索システムが提案されている．これは PBWT-sec で用いられている加法準同型暗号を FHE に置き換えたもので，Smart et al. によるパッキング技術 [22] も導入されている．

二つ目の先行研究 [5] では，bootstrap を導入したゲノム秘匿検索システムが提案されている．Bootstrap の導入により，クライアント・サーバ間での通信回数はクエリの文字列長に関わらず一定となる．また，計算アルゴリズムの最適化により，実行時間の削減にも取り組んでいる．

これらの先行研究では FHE の演算に HElib[11] により提供される BGV スキーム [12] が用いられている．システムデザインの詳細は5章で議論する．

また，本研究では議論しないが，これらの他に，FHE を用いたゲノム秘匿検索システムを分散処理の導入によって高速化するという研究もなされている [7]．分散処理の手法についてはデータベースの分割による分散，独立な計算の並列処理による分散，独立なアルゴリズムの並列処理による分散などが挙げられるが，PBWT-sec のプロトコルでは直前の計算結果を用いた演算が繰り返し行われるため，この先行研究では最もシンプルなデータベースの分割による分散が行われている．

5. システムデザイン

5.1 デザインとトレードオフ

本章では先行研究 [4]-[5] で提案された二つのデザインについて，それぞれの特徴と利点・欠点を議論する．3.1節で述べたとおり，FHE では計算のたびに暗号文に含まれるノイズが増えてしまうため，復号を保証するためには複数のアプローチが考えられる．図 3 に示されているデザイン 1 では，サーバ上での演算回数を減らすことで復号を保証している．

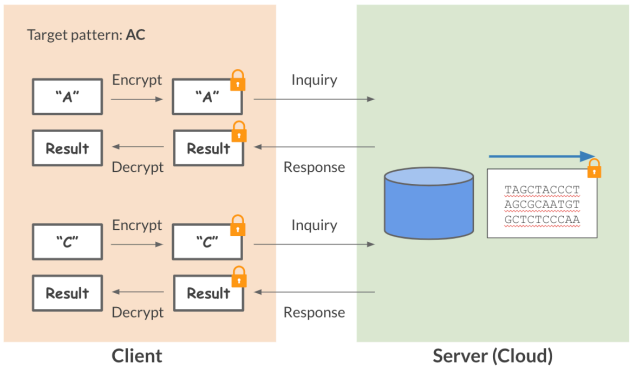

図 3: アプリケーションデザイン 1

(1) クライアントはクエリを一文字暗号化し，その他のパラメータと一緒にサーバに送信する．
(2) サーバはクエリを元に FHE 演算を行い，結果をクライアントに送信する．
(3) クライアントはサーバから送られてきたデータを復号する．
(4) クライアントは得られた結果を用いて次のクエリを暗号化し，サーバに送信する．
(5) (2)-(4) をクエリの文字列長だけ繰り返し，最終的な結果を得る．

デザイン 1 では，サーバは一度に一文字の検索しか行わず，クライアントは問い合わせを繰り返すことで最終的な結果を得る．これによって一度あたりのサーバ上での FHE の計算量を削減し，bootstrap を用いずに検索を行うことが出来る．しかしその一方で，クエリ長が増大するとともに，クライアントでの計算負荷も増加してしまうという問題もある．また，毎回の通信では暗号データを始めとする大容量のデータが転送されるため，このデザインは計算資源の乏しいクライアントには適さないと考えることが出来る．

図 4 はもう一方のデザインについて示している．

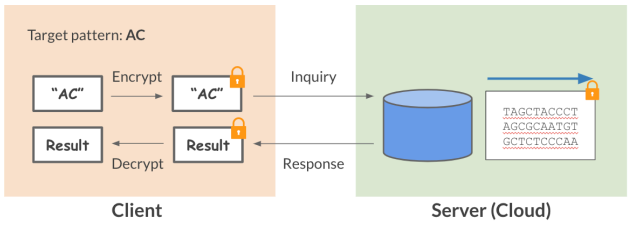

図 4: アプリケーションデザイン 2

(1) クライアントはクエリを全文暗号化し，その他のパラメータと一緒にサーバに送信する．
(2) サーバはクエリを元に FHE 演算を行い，結果をクライアントに送信する．
(3) クライアントはサーバから送られてきたデータを復号し，結果を得る．

デザイン 2 では，クエリの文字列長に関わらず一往復の通信で検索を行うことが出来る．これはデザイン 1 よりも計算資源の乏しいクライアントには適しているが，暗号文の容量やサーバ上での毎回の FHE 演算の計算量はデザイン 1 と比べて増大してしまう．

デザイン 2 を可能にするためには，bootstrap の導入もしくはより多くの演算を可能にするための大きなパラメータの指定が必要となる．Bootstrap を用いる場合，暗号文のノイズをリセットすることが出来る一方で bootstrap 自体が計算量の大きな演算であるため，大きなオーバーヘッドが発生する．また，大きなパラメータを指定した場合，bootstrap によるオーバーヘッドは発生しないものの，毎回の FHE 演算の計算コストが増大してしまうという問題点がある．

5.2 実装

先行研究においては，デザイン 1 及びデザイン 2 の両方において HElib[11] が提供する BGV[12] を用いた実験が行われていた．本研究ではこれに加えて，PALISADE[14] が提供する BFV[17] でもシステムを実装し，比較実験を行う．デザイン 2 については，PALISADE では bootstrap が実装されていないことから，BFV を用いた場合は bootstrap を用いずに実装を行うが，BGV を用いた場合は先行研究 [5] と同様に bootstrap を用いた実装とする．

6. 実験

6.1 問題設定

ゲノムデータのサンプルとして，1000 人ゲノムプロジェクト [2] により提供されるデータから，個体差の現れやすい特定位置の塩基を取り出した一塩基多型 (SNP: Single-Nucleotide Polymorphism[23]) を並べた SNP 配列を 512 サンプル用いる．それぞれのサンプルの長さはは 10,000 文字とする．クエリ長は 5 から 25 まで 5 ずつ変化させて実験を行う．クライアントは2章で述べられている通り，ダミーを含んだ 5 つのポジションを指定する．

6.2 実験環境

デザイン 1 とデザイン 2 をそれぞれ BGV スキーム [12] 及び BFV スキーム [17] を用いて C++で実装する．いずれにおいても Smart et al. による パッキング技術 [22] を利用する．実験を行うマシンのスペックを表 1 に，実験に用いたパラメータを表 2 にそれぞれ示す．

表 1: 実験環境

Server	OS	CentOS 6.9
	CPU	Intel®Xeon®Processor E5-2643 v3 (3.4GHz) 6 Cores × 2 Sockets
	Main Memory	512GB
	SSD	80GB
	HDD	2TB

表 2: パラメータ

Scheme	Design	Parameter
BGV	Design 1	*level L = 23*
	Design 2	*level L = 8*
BFV	Design 1	*numAdds A = 10*
	Design 2	*numAdds A = 50*

6.3 実験結果

それぞれの実験を三回ずつ行い，平均の実行時間を算出した．図 5 と図 6 はそれぞれ図 3 と 図 4 に示されているデザイン 1 及びデザイン 2 の，各サーバ上での検索演算の平均実行時間をグラフにしたものである．図 7 はデザイン 2 を HElib が提供する BGV で実装した場合の，サーバ上での検索演算の平均実行時間の内訳を示している．また，表 3 はデザイン 2 を HElib が提供する BGV で実装した場合の，秘匿検索演算全体に占める bootstrap の計算時間の割合を示している．

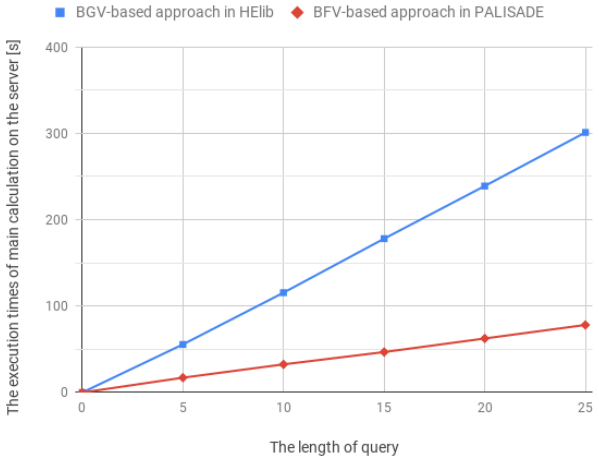

図 5: デザイン 1 のサーバ上での秘匿検索演算の平均実行時間

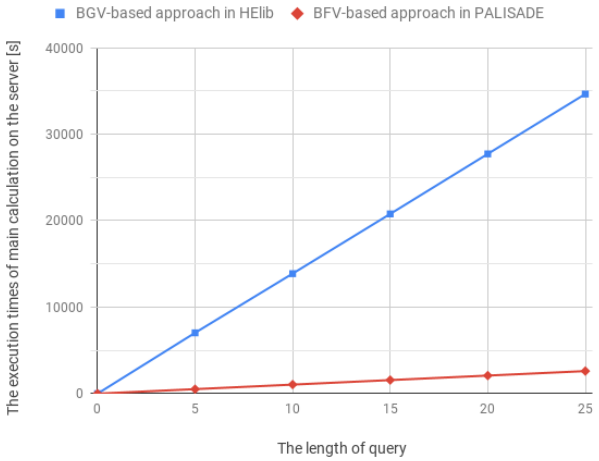

図 6: デザイン 2 のサーバ上での秘匿検索演算の平均実行時間

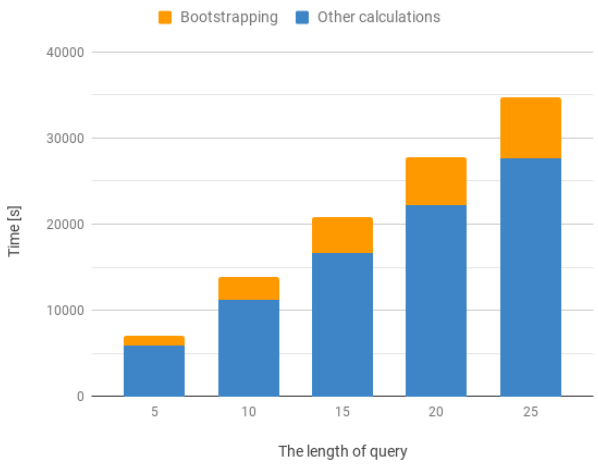

図 7: BGV を用いたデザイン 2 のサーバ上での秘匿検索演算の平均実行時間の内訳

表 3: BGV を用いたデザイン 2 のサーバ上での秘匿検索演算の平均実行時間全体に占める bootstrap 演算時間の割合

クエリ長	Bootstrap 演算時間の割合
5	0.16698
10	0.18987
15	0.19698
20	0.20099
25	0.20312

7. 分析

7.1 スキーム

図 5 からは，HElib が提供する BGV よりも PALISADE が提供する BFV のほうが良いパフォーマンスを示すこと，また実行時間の差はクエリ長が増加するに従って開いていくこと，の二つを読み取ることが出来る．図 6 も同様のことを示しているが，実行時間の差は図 5 よりも大きくなっている．以上より，このアプリケーションにおいては，HElib が提供する BGV よりも PALISADE が提供する BFV のほうが各 FHE の演算において高速であることが分かる．

7.2 Bootstrap

図 7 と表 3 は BGV を用いた場合のデザイン 2 の実験結果に焦点を当てており，bootstrap の実行時間が秘匿検索演算全体に対してかなりの割合を占めていること，クエリ長の増大に従ってその割合は増加することを示している．

5.1 節で述べたとおり，デザイン 2 においては復号を保証するために，bootstrap を導入してノイズをリセットするか，十分に大きなパラメータを設定する必要がある．図 6 及び 図 7 より，大きな値のパラメータを設定することで各演算の計算量は大きくなるが，これによるオーバーヘッドよりも bootstrap の導入によって発生するオーバーヘッドの方がより負荷が高いと考えることが出来る．

7.3 実用性

図 5 に示されている通り，デザイン 1 の場合，PALISADE が提供する BFV を用いたサーバ上での実行時間はクエリ長が 25 のときでも約 80 秒となっている．これは実用に耐えうる計算時間だと考えられる．対して，図 6 に示されているデザイン 2 では，BFV を用いた場合でもサーバ上での実行時間はクエリ長が 5 の時点で 500 秒を上回っている．これを実用化するためには，より高速化を推し進める必要がある．

8. まとめと今後の課題

先行研究に基づき，ゲノム秘匿検索システムを二種類のデザイン及び二種類の暗合スキーム・ライブラリを用いて実装し，その実験結果について分析を行った．その結果，このアプリケーションでは PALISADE により提供される BFV スキームが良い性能を示すこと，大きな値のパラメータによるオーバーヘッドよりも bootstrap の導入によるオーバーヘッドのほうが大きいことが確認された．しかしながら，今回の実験で用いられたパラメータは十分に調整されておらず，まだ議論の余地は残っている．また，HElib によって提供される BGV スキームを用いた場合でも，bootstrap を用いずにデザイン 2 を実装し，比較する必要がある．今後は実装やアルゴリズムの改善による高速化に取り組むとともに，実行時間だけではなくクライアント・サーバ間での通信コストについても計測していきたい．

謝辞

本研究は JST CREST JPMJCR1503 の支援を受けております．

参考文献

[1] “The Human Genome Project - NHGRI.” URL: https://www.genome.gov/human-genome-project.

[2] “The International Genome Sample Resource.” URL: http://www.internationalgenome.org/.

[3] K. Shimizu, K. Nuida, and G. Rätsch: “Efficient privacy-preserving string search and an application in genomics.” *Bioinformatics* 32.11 (2016), pp. 1652–1661.

[4] Y. Ishimaki et al.: “Poster: Privacy-preserving string search for genome sequences using fully homomorphic encryption.” *IEEE Symposium on Security and Privacy*. 2016.

[5] Y. Ishimaki et al.: “Privacy-preserving string search for genome sequences with FHE bootstrapping optimization.” *2016 IEEE International Conference on Big Data (Big Data)*. IEEE. 2016, pp. 3989–3991.

[6] R. Durbin: “Efficient haplotype matching and storage using the positional Burrows–Wheeler transform (PBWT).” *Bioinformatics* 30.9 (2014), pp. 1266–1272.

[7] Y. Yamamoto and M. Oguchi: “A decentralized system of genome secret search implemented with fully homomorphic encryption.” *the 1st IEEE International Workshop on Big Data and IoT Security in Smart Computing (BITS2017)*. IEEE. 2017, pp. 1–6.

[8] R. L. Rivest, L. Adleman, M. L. Dertouzos, et al.: “On data banks and privacy homomorphisms.” *Foundations of secure computation* 4.11 (1978), pp. 169–180.

[9] C. Gentry et al.: “Fully homomorphic encryption using ideal lattices.” *Stoc*. Vol. 9. 2009. 2009, pp. 169–178.

[10] M. Naehrig, K. Lauter, and V. Vaikuntanathan: “Can homomorphic encryption be practical?” *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*. ACM. 2011, pp. 113–124.

[11] “shaih/HElib: An Implementation of homomorphic encryption.” URL: https://github.com/shaih/HElib.

[12] Z. Brakerski, C. Gentry, and V. Vaikuntanathan: “Fully Homomorphic Encryption without Bootstrapping.” *IACR Cryptology ePrint Archive* 2011 (2011), p. 277.

[13] J. H. Cheon et al.: “Homomorphic encryption for arithmetic of approximate numbers.” *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2017, pp. 409–437.

[14] “PALISADE.” URL: https://git.njit.edu/palisade/PALISADE.

[15] A. López-Alt, E. Tromer, and V. Vaikuntanathan: “On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption.” *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*. ACM. 2012, pp. 1219–1234.

[16] D. Stehlé and R. Steinfeld: “Faster fully homomorphic encryption.” *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2010, pp. 377–394.

[17] J. Fan and F. Vercauteren: “Somewhat Practical Fully Homomorphic Encryption.” *IACR Cryptology ePrint Archive* 2012 (2012), p. 144.

[18] “Microsoft SEAL: Fast and Easy-to-Use Homomorphic Encryption Library.” URL: https://www.microsoft.com/en-us/research/project/microsoft-seal/.

[19] “TFHE Fast Fully Homomorphic Encryption over the Torus.” URL: https://tfhe.github.io/tfhe/.

[20] “richarddurbin/pbwt: Implementation of Positional Burrows-Wheeler Transform for genetic data.” URL: https://github.com/richarddurbin/pbwt.

[21] “iskana/PBWT-sec.” URL: https://github.com/iskana/PBWT-sec.

[22] N. P. Smart and F. Vercauteren: “Fully homomorphic SIMD operations.” *Designs, codes and cryptography* 71.1 (2014), pp. 57–81.

[23] “What are single nucleotide polymorphisms (SNPs)? - Genetics Home Reference - NIH.” URL: https://ghr.nlm.nih.gov/primer/genomicresearch/snp.