Comparison and Analysis of System Designs for Privacy-Preserving Genome Sequences Search

Yuki Yamada

Dept. of Information Sciences
Ochanomizu University
Tokyo, Japan
Email: yuki@ogl.is.ocha.ac.jp

Masato Oguchi

Dept. of Information Sciences
Ochanomizu University
Tokyo, Japan
Email: oguchi@is.ocha.ac.jp

***Abstract*—Genome sequences search is useful, for example, in clinical applications where a care provider needs to select a treatment option for a patient based on the exact kind of cancer the patient might have. However, privacy protection for genome analysis is one of the most important issues in the area of medical genomics. In such a situation, only homomorphic encryption is a desirable technology to be used for this application because it is non-interactive. Privacy-preserving genome sequence search using homomorphic encryption has been a practical challenge because of the scalability issues driven by the depth of computations that need to be supported for privacy-preserving genome sequence search. Comparison and analysis of system designs for such a system are important for us to put them in practical use. Therefore, in this work, we build off of earlier researches for genome sequence search to design, then implement and compare each approach. We particularly focus on the differences in the main calculation time on the server and the data transfer overhead. Our results show that each design has different trade-offs and characteristics.**

Keywords*— *Homomorphic Encryption; Genome Sequence; Secure Search; Privacy; Cloud Computing.

I. Introduction

Ever since the Human Genome Project [1] and the 1000 Genomes Project [2] have begun publishing catalogs of human variation and genotype data, genomic data analytics have found increasingly practical and important use in various fields. Privacy challenges associated with analytics on genomic data have been exacerbated by recent innovations that made it much less expensive to handle genetic information. Furthermore, it is difficult for hospitals or research institutes that have genome databases to publish the complete data because of the privacy issues, and also it is not desirable for researchers to make their work-in-progress work public. Therefore, it is needed to keep both genome database contents and the user's query in private. However, because genomic data is potentially voluminous, making scalability challenges are important when analyzing genetic data, especially when needed to be done in a privacy-preserving manner.

Of particular interest, genomic search applications look for some specific sub-strings, thus driving the need for a system that can conduct privacy-preserving string searches on vast amounts of genome data. Generic cloud computing environments are not feasible to address this need due to security and privacy concerns engendered by multi-tenancy, and the cloud may be managed by unknown and un-trusted individuals. A simple solution to the cloud-based storage of privacy-sensitive genomic information is to use encryption. If this system is built with common symmetric- or public-key encryption, the decryption key would be passed to the cloud to enable analytics, thus creating a privacy concern. Only homomorphic encryption techniques enable non-interactive computation on the data when it is encrypted. Fully Homomorphic Encryption (FHE) supports non-interactive computation on encrypted data. Hence, FHE allows a client to upload a corpus of genomic data to a high-performance off-premise computation environment and then search on that genomic data without leaking its private information to the computation host. However, search operations are considered to be "deep", meaning they are not efficient when running on homomorphically encrypted data.

Prior efforts show some methods that use FHE to protect privacy [3][4]. In these methods, encrypted data is uploaded to a cloud for privacy-preserving non-interactive computation without decryption. Although there have been attempts to accelerate these systems by introducing decentralized computing, such as in [5] as well, the calculation costs on the cloud are still too large to put into practice.

In this paper, we implement the privacy-preserving genome sequences search system in multiple designs based on the previous work [3][4] and compare their performance to explore design trade-offs. In Section II we introduce the motivating application of privacy-preserving genome sequences search. In Section III, we introduce relevant features of FHE techniques. In Section IV, we provide a broad overview of relevant prior work. In Section V, we discuss designs trade-offs and approaches that we build on and explore. In Section VI, we discuss our implementation and experimental settings and then show experimental results. In Section VII, we analyze the results of our experimentation. In Section VIII, we conclude this research and discuss future directions.

 ISBN: 978-1-61208-746-7

II. Genome Sequence Search Application

The goal of an application for the privacy-preserving genome sequences search is for clients to query if there are matches between a query string and the data in a genome database stored on an off-site server [6]. Genomic data are composed of sequences of 4 different kinds of nucleotides – A, G, C, and T –, therefore, we can regard this genome sequences search as a 4-kind character search [5].

A representation of this operation is seen in Figure 1.

Figure 1. Genome sequences search

We assume a secure model of a privacy-preserving genome search system with a cloud environment, where the outsourcing system is implemented in the client-server style. Its representation is shown in Figure 2.

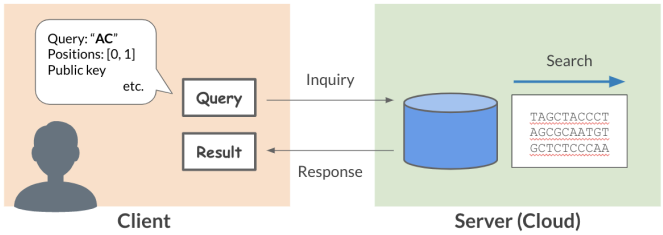

Figure 2. Application overview

A server holds a set of genome sequences data aligned by each sample in a database. This means that it is possible to search every sample in a specific position of the genome sequence. Clients send the inquiry to a server to calculate the matches between the query and the database held by the server. The query sent by the client includes not only the encrypted string that the client wants to search the genome sequence for but also some other parameters such as a public key for calculations and multiple starting points of the search for genome data strings (search positions). By designating multiple positions including dummy ones, clients can hide the actual one the clients use from the server. On receiving an inquiry from a client, the server conducts match searching on the data with FHE calculations, and then transmits the result to the client. The result transmitted by the server indicates whether there are any matches between the query string and genome sequences or not.

III. Fully Homomorphic Encryption (FHE)

As discussed in Section I, we leverage FHE to provide privacy-preserving genome sequences search. As seen respectively in (1) and (2), these homomorphisms are called the Additive Homomorphism (which supports addition over encrypted data), and the Multiplicative Homomorphism (which supports multiplication over encrypted data.)

Additive/Multiplicative Homomorphism

$$Encrypt(m) \oplus Encrypt(n) = Encrypt(m+n) \quad (1)$$
$$Encrypt(m) \otimes Encrypt(n) = Encrypt(m \times n) \quad (2)$$

FHE supports both of these homomorphism properties. By leveraging these properties, users can support the evaluation of polynomial circuits over ciphertexts analogous to how they would support similar circuits evaluated on plaintexts.

FHE was first proposed by Rivest et al. in 1987 [7] but was not known to be feasible until a candidate scheme was discovered by Gentry in 2009 [8]. This first scheme leverages polynomial rings and ideal lattices, and the encrypted text is constructed by encrypted data and random noise to guarantee its difficulty to decrypt without the appropriate secret key. This early scheme was computationally inefficient, for example, the ciphertext of this implementation would be 1 GB on encrypting 1 bit data. There have been tremendous recent strides in developing increasingly more efficient schemes and their implementations. For example, Lu et al. [9] show a scheme that supports a comparison homomorphism in addition to addition and multiplication homomorphisms.

There are still many large challenges with FHE. For example, noise accumulates in ciphertexts when computations are performed on them. As this noise grows, the ciphertexts eventually cannot be decrypted correctly after too many computations are performed. The random noise in ciphertexts grow additively with additive operations and multiplicatively with every multiplication operation. This noise growth would normally limit the size of computations that could be performed with FHE. However, there is a special method called *bootstrapping*, which reduces the noise embedded in a ciphertext, with the drawback that the bootstrapping operations are extremely computationally intensive.

Note that many practical applications of FHE schemes use a limited version of FHE without bootstrapping. The ”reduced” version of FHE is called Somewhat Homomorphic Encryption (SHE or SwHE) [10]. This is the ability to conduct some simple calculations that can be derived with one-time multiplication and multiple times addition, such as the inner product of the vector, distribution, and correlation.

 ISBN: 978-1-61208-746-7

IV. Previous Work

A. PBWT-sec

Several previous attempts have been made to realize practical privacy-preserving genome sequences search. PBWT-sec [6] is an efficient two-party prefix much-counting protocol that combines Additive Homomorphic Encryption (AHE) and an efficient data structure for much searching called Positional-Burrows Wheeler Transform (PBWT) [11]. The server of PBWT-sec has a genome sequences database as PBWT style, that is transformed from an original aligned genome sequences database. In its searching phase, the server access to a look-up vector that is derived from PBWT recursively. This is named Recursive Oblivious Transfer (ROT) [11]. When the query string length is l, ROT consists of l times vector-lookups, which needs l rounds of communication between the client and the server. PBWT-sec also devises the idea that the client passes multiple amounts of search positions, which includes dummy ones, to a server to preserve the privacy of clients with hiding the positions that the client uses. Although AHE can be used as an encryption method, according to this work [6], it is considered that preventing genome data leakage with AHE is difficult because we cannot conduct complex calculations with it.

B. Genome sequences search with FHE

While FHE engenders a much longer computation time than that of AHE, we can extend the PBWT-sec approach to use computation methods that search with wildcards and compute statistics based on the search result by building genome sequences search with FHE.

There are two relevant prior attempts by Ishimaki et al. [3][4]. First, one [3] proposes multi-round privacy-preserving genome sequence searches with FHE based on PBWT-sec [6]. This approach replaces additive homomorphic methods in PBWT-sec with fully homomorphic methods and also introduces the packing technique proposed by Smart et al. [12]. The other [4] propose an efficient approach for one-round search with FHE by introducing bootstrapping and reducing the runtime of the system by optimizing the calculation procedure. These two work use HElib [13] and its BGV implementation as a software library for FHE calculations. The detail of the system design that is proposed by each work is discussed in Section V.

C. FHE scheme comparison for genome sequences search

There is another work we have done for privacy-preserving genome sequences search [14]. In the paper, we implemented multiple designs of genome search systems in two schemes, BFV [15] in PALISADE [16] and BGV [17] in HElib [13], and then compared their calculation time on the server. There is a myriad of options and design trade-offs associated with the application of homomorphic encryption in this domain-driven, not only design trade-offs but also scheme selection, choices in data encoding, even encryption software library.

V. Design and Trade-offs

A. Design 1

First, we introduce the Design 1, proposed by [3], shown in Figure 3 below.

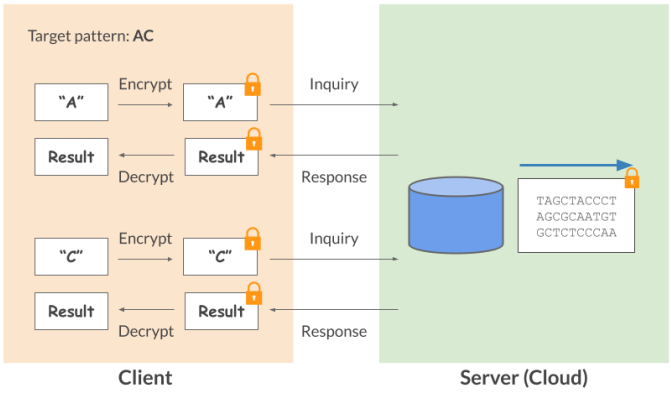

Figure 3. Application Design 1

(1) The client encrypts one character of the query string and then passes the resulting ciphertext to the server with other parameters.
(2) The server then performs FHE computations and then returns the result to the client.
(3) The client decrypts the intermediate result.
(4) The client encrypts the next character of the query using the result and sends it to the server.
(5) Repeat (2)-(4) as many times as the length of the query.

There are two approaches to support the needed depth of computation to avoid incorrect decryption: limiting the depth of FHE computations to keep the noise in ciphertexts less than noise threshold for correct decryption, or adopting bootstrap to reduce noise in ciphertexts. In this design, the server operates over a single character at a time, and thus the client gains the final result by comparing the results for all query characters. This reduces the depth of computation on the server and enables reduced noise to remove the need for bootstrapping. However, computation costs on the clients and communication costs between the client and the server increase as the length of the query increases. Since each communication involves large data transfer, this design is inappropriate for the clients with limited communication resources and requires that the clients both be available and have appropriate computation resources for repeated encryption and decryption.

B. Design 2-1 and Design 2-2

Next, we introduce Design 2-1 and Design 2-2. The server in both Design 2-1 and Design 2-2 supports the whole string search to address the issues Design 1 has. Thus Design 2-1 and Design 2-2 are more appropriate for the client with limited computation power as compared to Design 1. However, the data size of ciphertexts as welll as FHE calculation costs on the server of Design 2-1 and Design 2-2 are much greater than that of Design 1.

 ISBN: 978-1-61208-746-7

There are two general approaches to support the large computation depth needed on the server of Design 2. Design 2-1, proposed by [4], reduces the noise by bootstrapping as shown in Figure 4.

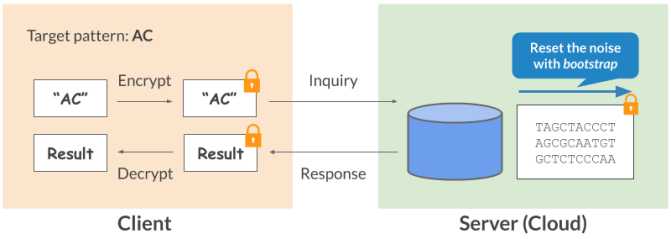

Figure 4. Application Design 2-1

(1) The client encrypts the whole query string and then passes the encrypted query string with supporting parameters to the server.
(2) The server performs FHE computations and reset noise with bootstrap accordingly.
(3) The server transmits the encrypted result to the client.
(4) The client gains a result by decrypting the received data.

The server in Design 2-1 can operate the whole string search by adopting the method called bootstrap. Bootstrap can reset the noise in the ciphertexts while each bootstrapping operation causes expensive overhead. Previous work [4] proposed the approach to minimize the number of bootstrapping with the parameters for a reduced number of calculations.

Alternatively, Design 2-2 sets sufficiently large parameters to ensure correct decryption within a limited (but large) number of operations as shown in Figure 5.

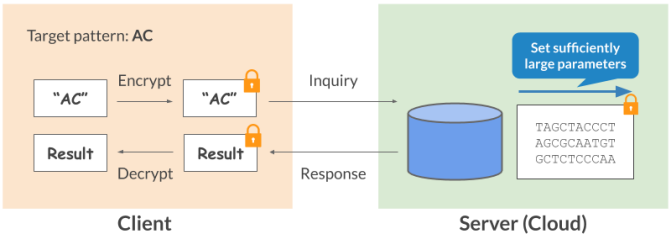

Figure 5. Applicatin Design 2-2

(1) The client encrypts the whole query string and then passes the encrypted query string with supporting parameters to the server.
(2) The server performs FHE computations with large parameters.
(3) The server transmits the encrypted result to the client.
(4) The client gains a result by decrypting the received data.

Design 2-2 is a more naive approach that sets sufficiently large parameters so that no bootstrap is needed. Using the parameters for the larger number of operations deteriorates the performance of all the arithmetic operations, while each bootstrapping operation costs expensive overhead.

VI. Experimentation

Based on previous work, we implement Design 1, Design 2-1 and Design 2-2 discussed in Section V and then compare them on real-world data.

A. Problem settings

The genomic data used for this experiment are Single-Nucleotide Polymorphism (SNP) [18] sequences from the 1,000 Genomes Project [2]. This data provide the representation of where variations from a reference genome are likely to appear, without showing entire genome sequences. In our experimental setting, the number of genomic data samples is 2185 and the number of characters per sample is 10,000. The range of the query length is 1–6, and clients designate just one search position.

B. System overview

We implemented the privacy-preserving genome sequences search system in multiple designs discussed in Section V in C++. As a software library for homomorphic encryption, we adopted HElib [13]. Both systems adopt the Chinese Reminder Theorem (CRT) packing technique by Smart et al. [12] as well. Experiments were conducted on the machines that have the specification shown in Table I and parameters used for these experiments are summarized in Table II.

TABLE I. EXPERIMENTAL ENVIRONMENT

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

TABLE II. PARAMETERS FOR THE EXPERIMENTS

Design	Parameter L
Design 1	8
Design 2-1	23
Design 2-2	9 * (query length)

C. Experimentation Results

We ran each experiment three times and calculated the average of results.

Figures 6-7 show each graph of the average client-to-server and server-to-client data transfer overhead (volume of data) of Design 1, Design 2-1 and Design 2-2 shown in Figures 3-5, based on the length of the query. Figure 8 shows the average execution time on the server of Design 2-1 and Design 2-2 based on the length of the query.

 ISBN: 978-1-61208-746-7

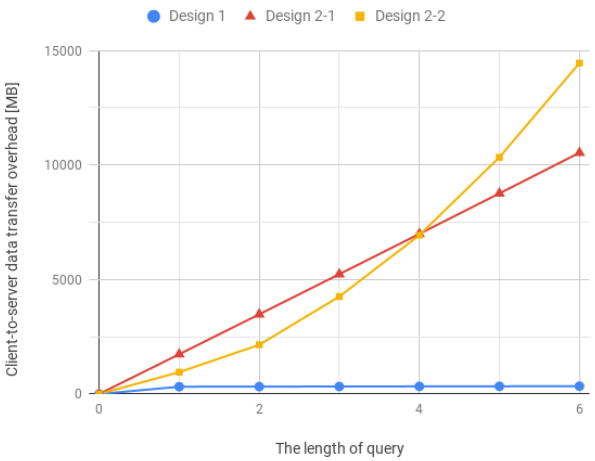

Figure 6. Client-to-server data transfer overhead

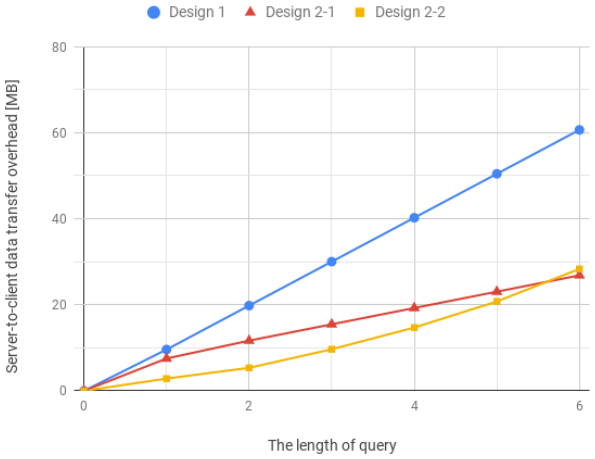

Figure 7. Server-to-client data transfer overhead

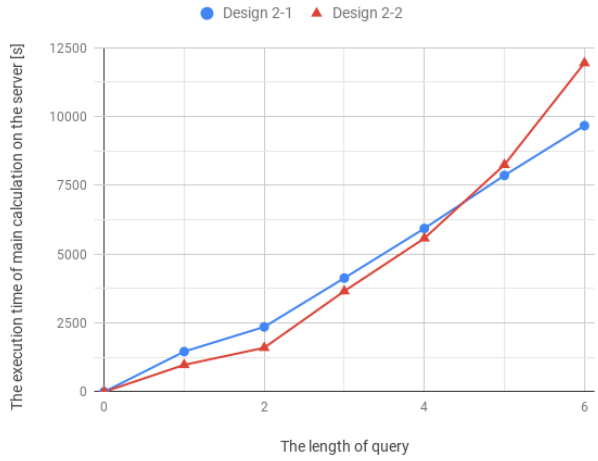

Figure 8. Average execution time of the main calculation on the server of Design 2-1 and Design 2-2 by the length of query

VII. Analysis of Experimental Results

A. Data transfer overhead

First, we compare client-to-server and server-to-client data transfer overhead (volume of data) by each application design. According to the result, Figures 6-7 show the opposite things: the smallest client-to-server data transfer overhead and the largest server-to-client data transfer overhead are those of Design 1.

We can regard server-to-client data transfer overhead as a more important one because it is assumed that the server has plenty of resources while the clients have poor ones, meaning that Design 2 would be more suitable for the client with less computation resource. However, it is needed to evaluate more kinds of values from more points of view to examine the best application design for the client with particular specification; we should compare not only the client-to-server and server-to-client data transfer overheads but also data transfer time between clients and server and the homomorphic calculation time on clients.

B. Execution time

Next, to compare Design 2-1 and Design 2-2 in detail, we compare the execution time of the main calculation on the server. Figure 8 shows the result for this comparison. It can be observed that the main calculation time on the server of Design 2-1 increases linearly while that of Design 2-2 increases in the multiplier.

This result is because of the parameters used for this experimentation shown in Table II. Although we can use the same parameters for FHE calculation in Design 2-1, the parameters for that in Design 2-2 need to get larger as the length of query increases to guarantee correct decryption without using bootstrap nor the noise in the ciphertext of FHE exceeding the threshold. However, it is also indicated that the calculation time on Design 2-2 is faster than that on Design 2-1 with a shorter length of the query. This means that it is better to switch the design to use according to the query and some other parameters.

VIII. Conclusion and Discussion

Comparison and analysis of system designs for systems is important to put them in practical use. Therefore, in this paper, we implemented and compared multiple designs for the client-server style system for privacy-preserving genome sequences search with BGV in HElib based on prior work. Our results show three things: the calculation costs on the clients in Design 1 increases more as the length of query increases, the calculation costs on the server in Design 2 increases more as the length of query increases, and it depends on the length of query and some other parameters that decide which design is the more suitable in Design 2-1 and Design 2-2. As future work, we plan to compare the execution time on the clients and data transfer time with a limited resource of clients, as well as the execution time on the server and data transfer size.

 ISBN: 978-1-61208-746-7

Acknowledgement

This work was partly supported by JST CREST Grant Number JPMJCR1503, Japan.

References

[1] NHGRI. (2019). The human genome project, [Online]. Available: https://www.genome.gov/human-genome-project (visited on 9–2019).

[2] EMBL-EBI. (2018). The international genome sample resource, [Online]. Available: http://www.internationalgenome.org/ (visited on 9–2019).

[3] Y. Ishimaki, K. Shimizu, K. Nuida, and H. Yamana, "Poster: Privacy-preserving string search for genome sequences using fully homomorphic encryption," in *IEEE Symposium on Security and Privacy*, 2016.

[4] Y. Ishimaki, H. Imabayashi, K. Shimizu, and H. Yamana, "Privacy-preserving string search for genome sequences with fhe bootstrapping optimization," in *2016 IEEE International Conference on Big Data (Big Data)*, IEEE, 2016, pp. 3989–3991.

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

[6] K. Shimizu, K. Nuida, and G. Rätsch, "Efficient privacy-preserving string search and an application in genomics," *Bioinformatics*, vol. 32, no. 11, pp. 1652–1661, 2016.

[7] R. L. Rivest, L. Adleman, and M. L. Dertouzos, "On data banks and privacy homomorphisms," *Foundations of secure computation*, vol. 4, no. 11, pp. 169–180, 1978.

[8] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Stoc*, vol. 9, 2009, pp. 169–178.

[9] W. Lu, S. Kawasaki, and J. Sakuma, "Using fully homomorphic encryption for statistical analysis of categorical, ordinal and numerical data," *IACR Cryptology ePrint Archive*, (2016/1163).

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

[11] R. Durbin, "Efficient haplotype matching and storage using the positional burrows–wheeler transform (pbwt)," *Bioinformatics*, vol. 30, no. 9, pp. 1266–1272, 2014.

[12] N. P. Smart and F. Vercauteren, "Fully homomorphic simd operations," *Designs, codes and cryptography*, vol. 71, no. 1, pp. 57–81, 2014.

[13] homenc. (2019). Helib: An implementation of homomorphic encryption, [Online]. Available: https://github.com/homenc/HElib (visited on 9–2019).

[14] Y. Yamada, K. Rohloff, and M. Oguchi, "Homomorphic encryption for privacy-preserving genome sequences search," in *The 3rd IEEE International Workshop on Big Data and IoT Security in Smart Computing (BITS2019)*, 2019.

[15] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *IACR Cryptology ePrint Archive*, vol. 2012, p. 144, 2012.

[16] PALISADE. (2019). Palisade release, [Online]. Available: https://gitlab.com/palisade/palisade-release (visited on 9–2019).

[17] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "Fully homomorphic encryption without bootstrapping," *IACR Cryptology ePrint Archive*, vol. 2011, p. 277, 2011.

[18] NIH. (2019). What are single nucleotide polymorphisms (snps)? - genetics home reference - nih, [Online]. Available: https://ghr.nlm.nih.gov/primer/genomicresearch/snp (visited on 9–2019).

 ISBN: 978-1-61208-746-7