Published in final edited form as:

Comput Methods Programs Biomed. 2018 October ; 165: 129–137. doi:10.1016/j.cmpb.2018.08.007.

Secure Large-Scale Genome Data Storage and Query

Luyao Chen^{a,2,1}, Md Momin Al Aziz^{b,2,1,*}, Noman Mohammed^b, and Xiaoqian Jiang^c

^aHeinz College, Carnegie Mellon University

^bComputer Science, University of Manitoba

^cSchool of Biomedical Informatics, University of Texas Health Science Center at Houston

Abstract

Background and Objective—Cloud computing plays a vital role in big data science with its scalable and cost-efficient architecture. Largescale genome data storage and computations would benefit from using these latest cloud computing infrastructures, to save cost and speedup discoveries. However, due to the privacy and security concerns, data owners are often disinclined to put sensitive data in a public cloud environment without enforcing some protective measures. An ideal solution is to develop secure genome database that supports encrypted data deposition and query.

Methods—Nevertheless, it is a challenging task to make such a system fast and scalable enough to handle real-world demands providing data security as well. In this paper, we propose a novel, secure mechanism to support secure count queries on an open source graph database (Neo4j) and evaluated the performance on a realworld dataset of around 735,317 Single Nucleotide Polymorphisms (SNPs). In particular, we propose a new tree indexing method that offers constant time complexity (proportion to the tree depth), which was the bottleneck of existing approaches.

Results—The proposed method significantly improves the runtime of query execution compared to the existing techniques. It takes less than one minute to execute an arbitrary count query on a dataset of 212 GB, while the best-known algorithm takes around 7 minutes.

Conclusions—The outlined framework and experimental results show the applicability of utilizing graph database for securely storing large-scale genome data in untrusted environment. Furthermore, the crypto-system and security assumptions underlined are much suitable for such use cases which be generalized in future work.

*corresponding author.

¹co-first authors

²Work done during an internship at Department of Biomedical Informatics, University of California San Diego

Publisher's Disclaimer: This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final citable form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.

Statements of Ethical Approval

None to be provided.

Competing Interest

There are no direct conflict of interests of any kind at present, nor any plans for there to be one in the foreseeable future.

Keywords

Secure genome data storage; Graph Database; Secure computation on genome data; Homomorphic Encryption; Genome data storage Neo4j

1. Introduction

Over the past decade, different technical breakthroughs have made genome sequencing more affordable. The next generation sequencing techniques made this growth somewhat exponential as we are starting to observing datasets in volume of Petabytes [1]. This increasing availability of genome data of different individuals gives us an opportunity to zoom into the micro level and analyze the complex correlation or causation. However, this is deeply challenging due to the size of the data, computational complexity, and inherent privacy issues.

As mentioned earlier, the immense size of genome data comes at a price of higher storage space. An economical solution will be leveraging the cost-efficient commercial cloud computing services (*i.e.*, Amazon EC2, Microsoft Azure, or Google Cloud Platform, etc.) to host data and conduct required analysis on demand. For example, Amazon S3 and Azure Storage Services charge only \$0.0208 to store 50 terabytes on a monthly base [2, 3]. More importantly, these cloud services also reduce the operational costs of running large scale experiments on such large-scale data.

Surely the commercial cloud services can provide a cost-effective and efficient solution to the ongoing genome data storage and computation issues. However, the privacy of these records is another notable aspect as public (/unrestricted) access of genome data might lead to re-identification attacks [4], surname recovery [5], facial and voice traits reconstruction [6, 7]. Thus, genome data are highly sensitive because they are irrevocable and have stigmatizing consequences to both the individuals and their family, particularly first-degree relatives [8]. There are some surveys that demonstrate and discuss these privacy and security issues [9, 10].

Due to these concerns and reported vulnerability of the public cloud [11], data custodians are not comfortable in depositing sensitive genome data in a third-party environment (untrusted) without enforcing necessary protection [12]. An ideal approach is to develop a secure genome database, *i.e.*, encrypting the data and providing a security layer on top of the operations interface for safeguarding the data analysis process. Assuming the cloud service provider is semi-honest (honest but curious [13]), and we only want to protect the data from external malicious users, data custodians can run queries on the encrypted data without establishing a complete, trusted relationship.

However, this computation on encrypted data induces a cost on performance as these security primitives are not efficient as their plaintext counterparts. Scalability is another challenge as large memory consumption imposed by these security protocols might hinder the practicability of a realistic system. Thus, in this paper, we look into the balance between privacy and efficiency of the computation of genome data. We consider the count query

operation which is the building block for various statistical analysis on genome data. A count query procedure to obtain the number of individuals satisfying a SQL-like query can be represented as:

```
SELECT count(*) FROM Sequences
      WHERE SNP1='A' AND SNP2='T' AND ...
      AND Disease = Yes
```

Single Nucleotide Polymorphism (SNP) refers to a variation of a single position on a DNA sequence (of a certain individual) such that more than 1% of the population does not carry the same value. Although not all SNPs correspond to disorders, some of them are known to be associated with some diseases. A count query between SNPs and a specific condition is the first step to explore the correlations and serves as the building block for Genome-Wide Association Studies (GWAS).

Contributions

In this paper, we propose a framework that provides better scalability and handles security issues of large-scale computations on genome data outsourced (transferred/stored) to a third party, public cloud server. Furthermore, we utilized a homomorphic cryptographic combined with garbled circuit scheme to ensure the security and tree structure to represent the arbitrary genome data for computational efficiency. The major contributions of the paper can be summarized below:

- We propose a method utilizing graph-based database to store and allow computations on real-world genome data *securely*.
- A novel indexing scheme is proposed on such database to make the secure query operations more efficient.
- We test the proposed approach along with the corresponding indexing scheme on a large-scale genome dataset containing 735, 317 human SNPs (~ 200GB data).
- Experimental results show that it takes less than a minute for a query compared to best-known attempts where it required around 7 minutes [14, 15].

The rest of the paper are organized as follows. Necessary backgrounds are discussed in Section 2. We discuss the proposed methods in Section 3 and show the results in 4. In Section 6 we discuss some of the related work. Finally, we conclude and discuss some future works in Section 7.

2. Preliminaries

In this section, we introduce some of the concepts (related to cryptography and genome data) required in understanding the proposed method.

2.1. Data Representation

In this paper, we consider the *Single Nucleotide Polymorphism* (SNP) of human DNA and its association with specific disease. For example, a mutation in BRCA1/2 genes has been reported to be associated with Breast cancer. A variant in BRCA1 is *rs1799950* is one of 25 SNPs to express an increased risk for breast cancer [16]. We considered such a SNP dataset with has a specific disease association. The data is represented in Table 1.

2.2. Graph Database

Graph database uses different interconnected graph compositions to represent the data. In contrast to relational (traditional) database, graph database considers data points as the nodes and the relation between them as edges. This approach has proved much useful [17, 18] in different literature and use cases as most of the relational data can be represented as a hierarchical data where one record is closely related to another. Graph database consists of nodes and edges where the nodes are interconnected with edges. Furthermore, there might be directional edges defining the connectivity of the nodes, though for simplicity we will only consider the non-directional edges throughout the rest of the paper. Regardless of the directions, the edges usually represent the relation between the nodes. In Figure 1 we depict the difference between a relational and graph database.

Formally, in a graph database (compared to relational tables), there are *relationships* which connect the *entities*. These entities can have specific properties. The relationships commonly described by verbs, for example, a patient ‘get’ certain conditions or a patient ‘has’ many SNPs. A relationship also has properties, for instance, the property ‘has’ describes the detail data of SNP.

2.3. Homomorphic Encryption

Homomorphic Encryption (HE) is an encryption scheme which allows computations under encryption. For example, consider two numeric values 2, 3 and the resulting homomorphic encryption are two random numbers $E(2)$ and $E(3)$. The result of $E(2) + E(3)$ will be the same as $E(5)$.

In this work, we utilized Paillier encryption [19] which has this additive property. However, there are other HE schemes with additive and/or multiplicative functionality. Regardless, we only need the additive property for our method and opted for this simple HE scheme.

2.4. Garbled Circuit

In 1986, Yao proposed Garbled Circuit (GC) which establishes a two-party protocol which allows the secure execution of an arbitrary Boolean function $f(x,y)$ against semi-honest adversaries [20]. Here, x and y are two inputs from two individual parties, and they are kept secret from each other while the output of $f(x, y)$ is disclosed. For example, in the millionaire problem, Alice and Bob want to know who has more money. They engage in a GC protocol where x and y are their net worth respectively. The output will be a Boolean value representing $f(x, y) = x > y$. If the value is one then $x > y$ (denotes that Alice is richer) and vice versa.

2.5. System Architecture Overview

We consider four different parties involved in our proposed architecture:

1. *Data Owners*: Data owners are the parties who sequence or own human genomic data. They have the proprietary rights over the data. Due to their technical limitation or the data aggregation requirements, they do not share the data directly. Instead, they hand over (or outsource) the data to the certified institutions.
2. *Certified Institution*: The certified institution is the trusted entity who generates and manages the cryptographic keys and responsible for the security of the proposed solution. We assume that a government organization such as NIH can play the role of a certified institution.
3. *Cloud Service Provider*: Cloud is responsible for storing the encrypted data and executing different queries on the encrypted data. We assume that the cloud service provider is a semi-honest entity and it only receives the public key. Hence, the cloud is unable to decrypt the encrypted data or the query.
4. *Researchers*: Researchers gain access to the query system from the Certified Institution. They acquire the public key to encrypt their query data and private key to decrypt the corresponding results.

Our proposed architecture is shown in Figure 2. We briefly overview the major steps of the architecture:

- *Key distribution*: The certified Institution sends the public key to the Cloud and distributes the public and private keys to authorized researchers.
- *Data processing*: This task is done by the certified institution (CI) prior to sending the encrypted data to the cloud. Initially, the CI collects the data from different data owners. Then, CI builds a count tree on the aggregated data and uses the count tree to build the index. Finally, then on the encrypted index. This operation is performed by executing a garbled circuit between the Cloud and researchers. Finally, the Cloud sends the encrypted query result to the researchers who decrypt using the private key. it encrypts the tree and data before sending it to the cloud.
- *Query execution*: The cloud receives an encrypted query from researches and executes them on the encrypted index. This operation is performed by executing a garbled circuit between the Cloud and researchers. Finally, the Cloud sends the encrypted query result to the researchers who decrypt using the private key.

3. Methods

In this section, we describe the techniques applied to represent the genome data in a graph database. Furthermore, we explain the secure execution of count query using two cryptographic primitives (HE and GC) described in the earlier section.

3.1. Data Preprocessing

Initially, we consider the data to be in a raw tabular format similar to Table 1 and stored in a text file. In some earlier attempts [15, 21], such row-wise data were preprocessed to a relational table and stored in a SQL database. In this work, we incorporate a graph database to store such data and essentially convert the relational tables into a tree structure. This approach is more realistic as we can model the data into three entities: a) patients, b) conditions (disease), and c) SNPs. Then these entities are connected with relationships such as a specific patient has a particular condition and several SNPs.

3.2. Counting Tree Construction

We generate the tree containing all the SNPs and the patients from a comma-separated values (CSV) file (similar to Table 1). We outline the algorithm to import the data in Algorithm 1. Initially, we generate an empty tree node and mark it as the root. Then for all SNPs from each patient (row-wise in Table 1) are linked sequentially in the tree. The first SNPs will be linked back to the root node (Figure 3a).

Subsequently, for each level of the tree, we group the nodes by their values (nucleotide values 'A/T/G/C') and keep the unique ones. Thus, the resulting tree will only contain unique nodes on a particular level, along with the number of occurrences. For example, if the nucleotide 'A' appears 3 times in level 1 of the tree, the aggregated node will have 3 as the count value. Figure 3a shows the initialization of the tree where we create all the nodes according to the CSV file (Table 1). Then, the nodes are aggregated only storing the unique SNP values in Figure 3b. The algorithm for creating this Counting Tree is provided in Algorithm 1.

This process is also much simpler than the earlier work [15] where the authors opted for processing the data row wise. Here, we can utilize batch processing capabilities of the database management system and reorder the tree afterwards.

The purpose of reordering the sequence of the counting tree is to further reduce the tree size. We sort the SNPs by their Shannon entropy so that those common SNPs will appear in the higher levels of the tree.

3.3. Indexing the Counting Tree

Another significant contribution in this work is indexing the tree structure comparing to the earlier work [15]. In our search algorithm, one of the important features is to confirm the linkage between a parent node and his children. The fundamental tree search functions result in a logarithmic runtime without any indexing [22], while the runtime in our graphic database is linear to the depth (/level). Thus, experimental results (Figure 6) show us that on a million depth tree constructed from genome data (according to Algorithm 1) takes around 10 minutes from root to leaf nodes (empirically). However, we can search the desired nodes (SNPs) in a linear time with the proposed indexing scheme.

Our indexing scheme puts *position tags* and stores the corresponding *range information* in the nodes along with the other data (nucleotide value, count of patients). Initially, we take all the nodes residing at the same level (siblings) and number them sequentially. This is shown

next to the nucleotide value in Figure 4. Then, the range of the child nodes are added to the node's data. It is noteworthy that this range information inherits all the position tags of its underlying children (example below).

Figure 4 and 5 details the corresponding steps which we describe in details here. The process works in two steps: First, we traverse the tree level-wise and assign an incremental number to each node. For example, in level 1 we assign 0 and 1 to the two nodes (A, T) available. Sequentially, at level 2, we assign 0, 1 and 2 to the nodes T, C and C respectively. Thus, we label each of the nodes of each level of the constructed tree.

In the second step, we start from the last level which denotes the n th SNPs of each patient records. These are the leaf nodes of the tree and do not have any children nodes. We assign the range of nodes (position tags) it connects as children and keeps it along the SNP value, count and the position as mentioned above. For these leaf nodes, the range will be its sequence number. Figure 5 shows this operation in detail where the leaf nodes contain their value as a range considering they do not have any child. However, the parents include the range of their children positions. Thus, the root node's range has the whole range of the tree which is (0 to 3).

During the validation, we only need to compare the ranges of parent and child nodes. If the range of the parent node covers the child's, then they are connected. For example, node A in level 1 has a range of [0, 2], so it has connectivity with the child nodes with position 0 to 2. Thus even the leaf nodes that belong to this range [0, 2] are connected to node A in level 1. However, any node having a position not included in the range [0, 2] are not connected. For example, the leaf node G with position 3 is not covered by A's range in level 1 ([0, 2]). Thus, this denotes that (leaf) node G is not connected with A (level 1).

3.4. Encryption of the Tree

The SNP nucleotide value and count will then be encrypted to protect the privacy of the data. We use the additive homomorphic encryption scheme, Paillier to encrypt the nucleotide and count values of each SNPs. However, before encrypting the value of the nucleotides we utilize a numerical encoding for each value A,C,G,T to 0, 1, 2, 3 respectively. Thus, an encryption, $E(A)$ will be stored as $E(0)$ in the counting tree. This encoding scheme will also be public as the researcher will know the mapping of $\{A,C,G, T\} = \{0, 1, 2, 3\}$.

As the applied cryptographic scheme (Paillier [19]) produces a randomized ciphertext, the ciphertexts will be indistinguishable even with same numeric values for the SNPs. For example, encryption of 'A' (or 0) will be different each time and seemingly random. Thus, an adversary cannot distinguish between two encryption of the same value (known as semantic security [19]).

3.5. Search Operation

In our framework, the search operation is based on queries like 'How many patients are there with $SNP1=A$ and $SNP2=C$...' from a particular dataset containing a specific disease. In our scheme, the cloud server has the public key only where the researcher has the private key as well. We utilize the tree structure (and index) mentioned above and encrypt the data of the

query (with encoding) accordingly. Our proposed method uses reference SNP IDs (*rs ids*) [23], which is equivalent to chromosome and position in the following example.

Initially, the researcher encodes his/her query parameters such as $SNP1 = 0$, $SNP2 = 1$. Then it encrypts them as $SNP1_A = E(0)$, $SNP2_C = E(1)$ and send to the cloud server as:

```
SELECT count(*) FROM Sequences
      WHERE SNP1=E(0) AND SNP2=E(1) AND ...
      AND Disease = E(1)
```

Here, the presence of the disease is also encrypted as a boolean value of 0 or 1. The cloud server separates the incoming query parameters (i.e., $SNP1_A = E(0)$ and $SNP2_C = E(1)$...) and sort them according to the tree order (from Section 3.3).

For example, based on the ordered tree Figure 5, SNP2 is the child of SNP1, the array would be queried in the order of $[SNP1_A = E(0), SNP2_C = E(1)]$. For each of these query SNP positions, we search along the tree with the assistance of the index created by position tags and ranges. For $SNP1$, let us assume it is positioned on the first level of Figure 5, which has two nodes A and T. The cloud service provider generates two random values $r1$ and $r2$, which are added to the SNP values of A and T, $SNP1_A^* = E(0 + r1)$ and $SNP1_T^* = E(3 + r2)$. These values are returned to the researcher, who subtracts its encrypted $SNP1_A$ and retrieves two random numbers $r01 = decrypt(SNP1_A^* - SNP1_A)$ and $r02 = decrypt(SNP1_T^* - SNP1_A)$.

A Garbled Circuit protocol is then executed to check whether $r01 = r1$ or $r02 = r2$. Because only $r1 = r01$ is true in this case, the cloud server only proceeds further on the left side and checks the branches connected to $SNP1 = A$. Suppose, $SNP2$ is positioned on Level n (due to the sorting of SNPs), we only need to check the three C nodes under the branch of node A in Level 1 (as their position ranges are falling between 0 and 2, which follows the range of children for node A). There is no need to check on the other node with SNP value G at Level n , which has a position range of (3,3), outside the child range of node A at Level 1.

The same verification procedure for SNP1 is repeated for SNP2 and the counts on the satisfying nodes (with SNP values equal C) are summed up at the cloud service provider to be $E(3)$ because there are three C nodes under the branch of A (at Level 1) falling between 0 and 2. The final layer is about the disease/phenotype diabetes, which has a binary value (yes/no). If 2 out of the 3 C nodes from Level n have 1's (1 means positive), the final count will be $E(2)$. This encrypted value is returned to the researcher, who gets a final count of 2.

4. Results

For the experiments, we used a realistic and largescale genome dataset from PGP [24]. The dataset had 173 patients, each with 736,317 (~0.75 million) SNPs.

We utilized the cloud services from Amazon AWS *m4.xlarge* instances (4 CPUs, 16GB memory, 500GB disk space) to store and perform the required computations. Furthermore,

for comparison with the earlier works [14, 15], we executed these algorithm with 7 patients and 736K SNPs in the same environment. Unfortunately, the programs ended with ‘Stack Overflow’ error during the early phase of encryption process. It indicated that the encrypted data is unduly excessive to be handled in the main memory.

In Table 2, we show the running times of the different Phase.

The generated tree from the aforementioned dataset generated 120 million nodes and required *223.41 GB of disk space*. We used Neo4j as the graph database on a Linux system. Table 2 indicates that the most time-consuming task was the encryption of the contents of the nodes. Hence, we utilized a multi-threaded architecture where multiple nodes were encrypted at the same time due to the non-atomic nature of the process. Furthermore, this process can be made faster using clustered programs on several cloud servers.

Table 3 scrutinizes the space requirements of different component of the Counting Tree on the database. As the content of the nodes (SNP values) are encrypted, it takes the most space (*String Store*) of 170 GB. This can be reduced with different encryption scheme which is not under consideration in this paper.

In Figure 6, we show the execution time for searching one node in different number of SNPs available in the database. We selected leaf nodes to search as this would result the time required to traverse the whole tree considering the depth (/SNPs). Evidently, the execution time of the count query increases with more SNPs, though it takes only 410 seconds (6minutes) to search in 7 million SNPs. For a query inside the million depth (search SNP in 1M), it takes only 31 seconds.

In Figure 7, we depict the effect of the query size on any given query. In other words, we experimented with different number of SNPs on the query sequence (*50, 100, 200, 500*) and analyze the execution time of retrieving the results. Furthermore, the effect of caching on the graph database was considered as well. We evaluated three scenarios:

1. *ColdDB*: Execution of a query with no caching
2. *HotDB*: Execution of multiple queries with *same* SNPs (full caching)
3. *WarmDB*: Execution of multiple queries with *random* SNPs (Figure 7)

Table 4 shows the effect of the aforementioned three scenarios where parsing a query depends on the number of SNPs and takes a significant amount of time. Caching effects are also available in this result where a fully cached query is returned much faster than the other two.

One critical implication of the proposed approach is the increment of the nodes related to the number of patients and SNPs. In Figure 8 we depict the number of nodes required for storing total 735,317 SNPs and different number of patients. It is apparent from the figure that the number of the nodes (/storage overhead) are not quite linear to the number of patients. For example, the system required 800,074 nodes for 5 patients where for 10 patients it needed 1,512,961 nodes. Though, in worst case, the expansion can be $2^{735,317}$ (if we see every variant of bi-allelic SNP), in our case we only observed 8,283,083 nodes after constructing

the full tree for 173 patients. This is much smaller than the worst-case scenario as the ratio of the bi-allelic SNPs follow beta distribution [25].

5. Discussion

Limitations:

Regardless of the encrypted sensitive data or the node values, we are not immune to these security leakages:

1. *Search pattern of the researcher:* Since the tree traversal depends on the GC outputs and researcher's query input, the corresponding path will be revealed to the Cloud. It is important to analyze this leakage as even with this search pattern the cloud will not acquire the sensitive information unless it colludes with the researcher. Furthermore, as all the nucleotide values are encrypted to random ciphertexts, the cloud server cannot infer any information about them. ObliviousRAM [26, 27] concepts can be used to mitigate this issue which will add additional computational complexity.
2. *Dishonest researcher:* In this paper, we do not consider malicious researchers as they have the private key which decrypts the results. We can overcome this at an additional cost of involving the data owner on every decryption. This will incur further communication and computational cost which will be deemed inefficient.

Future Work:

The principal direction for extension will be utilizing the proposed framework to answer complex queries. Though count queries are the building blocks of different statistical analysis (i.e., GWAS), different aggregation functions might also be useful in many cases [28]. The other key area of interest might be performing different machine learning algorithms.

Regarding the crypto primitives, instead of Paillier [19], we can analyze recent homomorphic schemes[29]. This will reduce the ciphertext size and speed-up the encryption time which seems to be a performance overhead.

6. Related Work

Our proposed methods offers significant modifications to Hasan *et al.* [15] where the authors proposed a solution for secure count query on encrypted genomic data using an Index tree. Their Index tree is a subset of our Counting tree but it lacks the indexing scheme proposed in Section 3.3. Their encryption and security models are similar as both of these works are provably secure under the semi-honest trust model [30]. However, our storage scheme has a significant difference as Hasan *et al.* solely relied on volatile memory. Previously, Hasan *et al.* 's method was only tested in a small database of 300 SNPs. So, we also tested on a large number of SNPs to see the scalability of methods. As a result, Hasan *et al.* 's method was not practical for real size genome data (e.g., 736, 317 SNPs in our setting) and their program ended with `Stack Overflow` error during the early phase of encryption process, while ours

took less than 50 seconds for a query consisting of 100 SNPs (on the same 16G RAM environment).

Similar problem of secure outsourcing and count query execution has been proposed by Ghasemi *et al.* [31], Canim *et al.* [32] and Kantarcioglu *et al.* [33] in 2016, 2012 and 2008 respectively. In most of these works, data were kept encrypted though un-indexed and tested in smaller datasets. However, in reality, providing security and efficiency in a realistic size of genomic data is much harder and reflected in our work. The prior works are summarized in Table 5.

7. Conclusion

In this paper, we demonstrate a realistic use case of secure genome data storage and retrieval application using graph database. Our new mechanisms are more scalable compared to the previous work due to the proposed indexing schemes. A demonstration responding arbitrary queries on different number of SNPs from ~ 750k SNPs (per person) within one minute shows the feasibility of our methods. However, the encryption mechanism (offline) is a major bottleneck of the scheme considering frequent database updates. This can be replaced by the recent state of the art HE mechanisms [34, 29] to improve efficiency.

Acknowledgement

Funding

This work was supported in part by the National Institute of Health (NIH) under award number U54HL108460, U01TR002062, R01GM124111, NSERC Discovery Grants (RGPIN-2015-04147) and University Research Grants Program (URGP) from the University of Manitoba.

References

- [1]. Cook CE, Bergman MT, Finn RD, Cochrane G, Birney E, Apweiler R, The european bioinformatics institute in 2016: Data growth and integration, *Nucleic Acids Res* 44 (D1) (2016) D20–6. [PubMed: 26673705]
- [2]. Azure storage services, <https://azure.microsoft.com/en-us/pricing/details/storage/blobs/>, accessed: 2017-12-13.
- [3]. Cloud storage pricing, <https://aws.amazon.com/s3/pricing/>, accessed: 2017-12-13.
- [4]. Lin Z, Owen AB, Altman RB, Genomic research and human subject privacy, *Science* 305 (5681) (2004) 183. [PubMed: 15247459]
- [5]. Gymrek M, McGuire AL, Golan D, Halperin E, Erlich Y, Identifying personal genomes by surname inference, *Science* 339 (6117) (2013) 321–324. [PubMed: 23329047]
- [6]. Claes P, Liberton DK, Daniels K, Rosana KM, Quillen EE, Pearson LN, McEvoy B, Bauchet M, Zaidi AA, Yao W, Others, Modeling 3D facial shape from DNA, *PLoS Genet* 10 (3) (2014) e1004224. [PubMed: 24651127]
- [7]. Lippert C, Sabatini R, Maher MC, Kang EY, Lee S, Arikian O, Harley A, Bernal A, Garst P, Lavrenko V, Yocum K, Wong T, Zhu M, Yang W-Y, Chang C, Lu T, Lee CWH, Hicks B, Ramakrishnan S, Tang H, Xie C, Piper J, Brewerton S, Turpaz Y, Telenti A, Roby RK, Och FJ, Venter JC, Identification of individuals by trait prediction using whole-genome sequencing data, *Proceedings of the National Academy of Sciences* 114 (38) (2017) 10166–10171.
- [8]. The privacy conundrum and genomic research: Re-Identification and other concerns, <http://www.healthaffairs.org/doi/10.1377/hblog20130911.034137/full/>, accessed: 2017-11-13.

- [9]. Naveed M, Ayday E, Clayton EW, Fellay J, Gunter CA, Hubaux J-P, Malin BA, Wang X, Privacy in the genomic era, *ACM Computing Surveys (CSUR)* 48 (1) (2015) 6.
- [10]. Aziz MMA, Sadat MN, Alhadidi D, Wang S, Jiang X, Brown CL, Mohammed N, Privacy-preserving techniques of genomic data survey, *Briefings in Bioinformatics* (2017) 10.1093/bib/bbx139. URL +10.1093/bib/bbx139
- [11]. 14 MEEELLION verizon subscribers' details leak from crappily configured AWS S3 data store, https://www.theregister.co.uk/2017/07/12/14m_verizon_customers_details_out/, accessed: 2017-11-13.
- [12]. Alomari Muhammad Ebtesam A, A survey of security issues for data sharing over untrusted cloud, *Journal of Emerging Trends in Computing and Information Sciences* 5 (8) (2014) 609–619.
- [13]. Liu D, Efficient processing of encrypted data in Honest-but- Curious clouds, in: 2016 IEEE 9th International Conference on Cloud Computing (CLOUD), 2016, pp. 970–974.
- [14]. Hasan Z, Mahdi MSR, Mohammed N, Secure count query on encrypted genomic data: A survey, *IEEE Internet Computing*.
- [15]. Hasan MZ, Mahdi MSR, Sadat MN, Mohammed N, Secure count query on encrypted genomic data, *Journal of biomedical informatics* 81 (2018) 41–52. [PubMed: 29550393]
- [16]. Johnson N, Fletcher O, Palles C, Rudd M, Webb E, Sellick G, dos Santos Silva I, McCormack V, Gibson L, Fraser A, et al., Counting potentially functional variants in *brca1*, *brca2* and *atm* predicts breast cancer susceptibility, *Human molecular genetics* 16 (9) (2007) 1051–1057. [PubMed: 17341484]
- [17]. Miller JJ, Graph database applications and concepts with neo4j, in: *Proceedings of the Southern Association for Information Systems Conference*, Atlanta, GA, USA, Vol. 2324, 2013, p. 36.
- [18]. Bollacker K, Evans C, Paritosh P, Sturge T, Taylor J, Freebase: a collaboratively created graph database for structuring human knowledge, in: *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, AcM, 2008, pp. 1247–1250.
- [19]. Paillier P, Public-key cryptosystems based on composite degree residuosity classes, in: *Advances in cryptology, EUROCRYPT*, Springer, 1999, pp. 223–238.
- [20]. Yao AC-C, Protocols for secure computations, in: *FOCS*, Vol. 82, 1982, pp. 160–164.
- [21]. Al Aziz MM, Hasan MZ, Mohammed N, Alhadidi D, Secure and efficient multiparty computation on genomic data, in: *Proceedings of the 20th International Database Engineering and Applications Symposium, IDEAS '16*, ACM, New York, NY, USA, 2016, pp. 278–283. doi: 10.1145/2938503.2938507. URL <http://doi.acm.org/10.1145/2938503.2938507>
- [22]. Aho AV, Hopcroft JE, The design and analysis of computer algorithms, Pearson Education India, 1974.
- [23]. Kitts A, Sherry S, The single nucleotide polymorphism database (dbSNP) of nucleotide sequence variation, *The NCBI Handbook* McEntyre J, Ostell J, eds. Bethesda, MD: US National Center for Biotechnology Information.
- [24]. Church GM, The personal genome project, *Molecular Systems Biology* 1 (1). arXiv:<http://msb.embopress.org/content/1/1/2005.0030.full.pdf>, doi:10.1038/msb4100040. URL <http://msb.embopress.org/content/1/1/2005.0030>
- [25]. Fumagalli M, Vieira FG, Korneliussen TS, Linderroth T, Huerta-S E´anchez A Albrechtsen R Nielsen, Quantifying population genetic differentiation from next-generation sequencing data, *Genetics* 195 (3) (2013) 979–992. [PubMed: 23979584]
- [26]. Gentry C, Goldman KA, Halevi S, Julta C, Raykova M, Wichs D, Optimizing oram and using it efficiently for secure computation, in: *International Symposium on Privacy Enhancing Technologies Symposium*, Springer, 2013, pp. 1–18.
- [27]. Stefanov E, Van Dijk M, Shi E, Fletcher C, Ren L, Yu X, Devadas S, Path oram: an extremely simple oblivious ram protocol, in: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, ACM, 2013, pp. 299–310.
- [28]. Stephens ZD, Lee SY, Faghri F, Campbell RH, Zhai C, Efron MJ, Iyer R, Schatz MC, Sinha S, Robinson GE, Big data: astronomical or genetical?, *PLoS biology* 13 (7) (2015) e1002195. [PubMed: 26151137]

- [29]. Chillotti I, Gama N, Georgieva M, Izabachène M, Improving tfhe: faster packed homomorphic operations and efficient circuit bootstrapping, Tech. rep., IACR Cryptology ePrint Archive 2017, 430 (2017).
- [30]. Pinkas B, Cryptographic techniques for privacy-preserving data mining, SIGKDD Explor. Newsl 4 (2) (2002) 12–19. doi: 10.1145/772862.772865. URL <http://doi.acm.org/10.1145/772862.772865>
- [31]. Ghasemi R, Aziz MMA, Mohammed N, Dehkordi MH, Jiang X, Private and efficient query processing on outsourced genomic databases, IEEE Journal of Biomedical and Health Informatics 21 (5) (2017) 1466–1472. doi:10.1109/JBHI.2016.2625299. [PubMed: 27834660]
- [32]. Canim M, Kantarcioglu M, Malin B, Secure management of biomedical data with cryptographic hardware, IEEE Transactions on Information Technology in Biomedicine 16 (1) (2012) 166–175. [PubMed: 22010157]
- [33]. Kantarcioglu M, Jiang W, Liu Y, Malin B, A cryptographic approach to securely share and query genomic sequences, IEEE Transactions on information technology in biomedicine 12 (5) (2008) 606–617. [PubMed: 18779075]
- [34]. Fan J, Vercauteren F, Somewhat practical fully homomorphic encryption., IACR Cryptology ePrint Archive 2012 (2012) 144.

Highlights

- We propose a method utilizing graph database system to store and allow computations on real-world genome dataset in a privacy preserving manner.
- A novel indexing scheme is proposed on such database to make the secure query operations more efficient.
- We test the proposed approach along with the corresponding indexing scheme on a large-scale genome dataset containing 735; 317 human SNPs (~ 200GB data).
- Experimental results show that it takes less than a minute for a query compared to best-known attempt where it required around 7 minutes.

Figure 1:
Representation of relational and graph database

Figure 2:
System Architecture and Process Flows

Figure 3:

Building the tree from genomic sequence according to Algorithm. The numbers under SNP values in (b) are the counts

Figure 4: Setting position tags on each level and their nodes. The numbers in the parenthesis are sequential labels (position tags) of each layer to be used for generating node ranges for quick indexing

Figure 5:

Range of position tags from underlying child in each nodes The range of each node is the union of ranges of its children

Figure 6:
Execution time (seconds) for searching *one* leaf node on different number of SNPs in the Counting Tree

Figure7:
Execution time (seconds) for searching different number of SNPs (randomly selected) in the Counting Tree

Figure 8:
Increment of the number of nodes along with number of patients {5,15,50,173}

Data: CSV file of SNP data

Result: Counting Tree in graph database

```

1 Create empty root node
2 foreach patient do
3   Load SNP sequence with default quantity 1
4   Link SNP sequence using relationship 'Next'
5   Link the first SNP in the sequence to root
   node
   /* a tree generated but has all nodes with
   quantity 1 */
6 for  $i \leftarrow 1$  to  $h$  do
   //  $h$  is the height of the tree
7    $N \leftarrow$  Group SNP data of level  $i$  by the same
   SNP value
8   foreach  $N$  where  $\text{sum}(N) > 1$  do
9     Create a new node  $n$  with SNP value and
     count Link node  $n$  to the parent level
     delete the old nodes  $N$  Link those child
     nodes whose parent in  $N$  to new node  $n$ 

```

Algorithm 1:

Generate a Counting Tree structure from human genome data

Data: Search string, S (formatted as SNP1=A and SNP2=T ...)

Result: The encrypted count of patients that meet the query string

```

1 Function SearchTree ( $S$ )
2    $Q \leftarrow$  Parse the search string,  $S$  //  $Q$  will be
   an array representing the query
3   Sort  $Q$  by the depth of the SNP according to
   the Counting Tree
4    $result \leftarrow 0$ ,  $range \leftarrow [0, \infty]$ 
5   foreach  $SNP_i$  in  $Q$  do
6      $N \leftarrow$  Nodes in  $SNP_i$  // traversal is done
   using the position tags and ranges
7      $Q_{SNP} \leftarrow$  the encrypted value of the SNP
   nucleotide from the query
8      $tmp \leftarrow 0$ 
9     foreach node  $n$  in  $N$  do
10       $currentRange \leftarrow$  range stored in  $n$ 
11      if  $currentRange$  is between  $range$  then
12         $t_n \leftarrow$  encrypted nucleotide
   encoding +  $r_n$  // random number  $r_n$ 
13         $res \leftarrow r_n == t_n - Q_{SNP}$  /* checking
   done with GC after sending  $t_n$  to
   researcher */
14        if  $res = 1$  then
15           $tmp \leftarrow 1$ 
16           $result \leftarrow$  the encrypted count
   value stored in  $n$ 
17           $range \leftarrow currentRange$ 
18      if  $tmp = 0$  then
19        return  $E(0)$  // no matches were found
20 return  $result$ 

```

Algorithm 2:

Searching for number of patients having specific SNP values

Table 1:

Considered genomic data containing multiple patients and corresponding SNPs

Patient	Genomic sequence					Phenotype
	SNP_1	SNP_2	SNP_3	...	SNP_5	Disease
1	A	T	G		C	YES
2	T	C	C		G	NO
3	A	T	C		C	NO
4	A	C	C		C	YES

Table 2:

Operations and their required time

<i>Operation</i>	<i>Time</i>
Load raw data and preprocessing	3 hours
Building the tree structure	8 hours
Adding position tag and range values	8 hours
Encryption of the tree nodes	5.5 days

Table 3:

Size of different elements of the Counting Tree in the Neo4J database

<i>Store Sizes</i>	<i>Size (GB)</i>
Node store	1.80
Property store	27.97
Relationship Store	12.13
String Store	170.37
Total Size	212.27

Author Manuscript

Author Manuscript

Author Manuscript

Author Manuscript

Table 4:

Relationship of the execution time with query size on different scenarios

<i>Scenarios</i>	SNPs in the Query			
	50	100	200	500
Parse Query	51	95	277	519
ColdDB	86	631	998	2090
HotDB	24	33	74	33
WarmDB	35	47	87	270

Table 5:

Comparison of related works on Secure Count Query chronologically. It is noteworthy that we experimented with a real world dataset and our scheme is invariant to the number of records

Authors	Year	Method	SNP _s	Time(s)
Kantarcioglu <i>et al.</i> [33]	HE	2008	40	6900
Canim <i>et al.</i> [32]	Hardware	2012	50	600
Hasan <i>et al.</i> [31]	HE,GC	2016	300	6
Ghasemi <i>et al.</i> [31]	HE	2016	50	90
Our Work	HE,GC	2017	736,317	40