

Preserving Consumer DNA Privacy For Finding Relatives In A Malicious Two Party Computation

by

Syed Zeeshan Ahmed

**Bachelor of Technology in Computer Science and Engineering,
Amity University, 2012**

**A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF**

Master of Computer Science

In the Graduate Academic Unit of Computer Science

Supervisor(s): Owen Kaser, PhD, Computer Science
Daniel Lemire, PhD, Computer Science
Examining Board: Rongxing Lu, PhD, Computer Science, Chair
Christopher Baker, PhD, Computer Science
Joanna Everitt, PhD, Political Science

This thesis is accepted by the
Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

August, 2020

© Syed Zeeshan Ahmed, 2020

Abstract

The evolution of consumer DNA analysis and testing has led consumers to face data privacy risks, as their data are often uploaded into the cloud and made publicly available. By various means, an adversary can obtain the consumer DNA data and perform activities that can harm a consumer. A certain section of consumers are willing to share their data and compromise on their data privacy. However, there exists another segment of consumers who do not want to undermine their data privacy. These consumers are unwilling to upload their sensitive data into the cloud, and they are only interested to find out whether they are related to another individual. In this research, we have created a platform where any two individuals can verify if they are related to each other without uploading or sharing their raw DNA data to each other or to the cloud. We have developed two techniques to verify individual relatedness while keeping their data private. The first approach is based on secure hash algorithms, while the other approach involves the garbled-circuit technique to ensure data privacy. We have implemented matching algorithms and opposite-homozygotes techniques to solve the genealogical DNA match-

ing problem to verify relationships. We have compared our approaches based on security assurance, running time and the correctness of the results. The experimental results show that the garbled-circuit approach is better overall than the hashing-based approach. The hashing-based approach takes about 7.5 seconds to execute the entire operation, whereas the garbled-circuit approach takes approximately 3 minutes to execute the operation. In general, an individual will run this software only a few times in their lifetime, so the running time of the garbled-circuit approach is acceptable.

Acknowledgements

This research was conducted at the University of New Brunswick. I am grateful to my supervisors, Dr. Owen Kaser and Dr. Daniel Lemire, for their continued feedback and insights, without their support I would not have been able to complete this research. I would also like to acknowledge the funding support of the Natural Sciences and Engineering Research Council of Canada (NSERC), grant number 2017-03910.

Table of Contents

Abstract	ii
Acknowledgments	iv
Table of Contents	v
List of Tables	x
List of Figures	xii
Abbreviations	xiv
List of Algorithms	xv
1 Introduction	1
2 Background	7
2.1 DNA	7
2.2 Genealogical Tests	9
2.2.1 Autosomal Testing	10
2.3 Genotyping	10

2.4	DNA-Matching Algorithm	11
2.4.1	Opposite- Homozygote Technique	13
2.4.2	How Is DNA Matched?	15
2.5	centiMorgan	18
2.6	SNP Reporting	20
3	Computational Division of DNA	22
3.1	Frames	22
3.1.1	Overlapping Frames	24
3.2	Start Position of centiMorgans	26
3.3	Genotyping Errors	29
3.4	Data Preprocessing and Cleaning	32
4	Data-Privacy Techniques	36
4.1	Data-Privacy Issues	36
4.2	Secure Two-Party Computation	37
4.2.1	Garbled Circuits	38
4.2.1.1	Oblivious Transfer	38
4.2.1.2	Garbled-Circuit Protocol	39
4.2.1.3	Garbling	40
4.2.1.4	Garbled Outputs	40
4.2.1.5	Evaluation	42
4.2.2	Authenticated Garbled Circuit	43
4.3	Secure Hashing Algorithm	44

4.3.1	SHA-256	45
5	Data Handling	47
5.1	Data Communication	47
5.1.1	Server-Client Architecture	48
5.2	Data Parsing	49
5.3	Finding and Sharing Common SNPs	49
5.4	Summary	50
6	Hashing-Based Approach	52
6.1	Main Steps of HBA	52
6.2	Generating the Nonce	53
6.3	Frame Division	54
6.3.1	Handling Genotyping Errors	56
6.4	Hashing	58
6.5	DNA Matching	61
6.6	Reporting Results	62
6.7	Data-Privacy Evaluation	62
6.7.1	Replay Attacks	65
7	Garbled-Circuit Approach	67
7.1	Main Steps of GCA	67
7.2	Pre-processing Phase	68
7.3	Encoding	69

7.4	Boolean Circuits	70
7.4.1	SNP-Matching Circuit	71
7.4.2	Frame Division	73
7.4.3	Tree-of-Adders Circuit	74
7.4.4	Handling Genotyping Errors	76
7.4.5	Less-Than- T_2 Circuit	76
7.4.6	Circuit Size	79
7.5	Garbled-Circuit Evaluation	86
7.5.1	Grouping of Frames	86
7.5.2	Reporting Results	87
7.6	Data-Privacy Evaluation	88
8	Experimental Setup and Results	90
8.1	Build and Test Setup	90
8.2	Data Sets	91
8.3	Test Scenarios	92
8.4	Data Characteristics	93
8.5	Common SNPs	94
8.6	Frames and centiMorgan Distribution	96
8.7	Relatedness	99
8.7.1	Garbled-Circuit Approach	101
8.7.2	Hashing-Based Approach	101
8.8	False Negatives in HBA	103

8.9	Running Times	104
8.9.1	Hashing-Based Approach	104
8.9.2	Garbled-Circuit Approach	105
8.10	Running Times Over Network	106
9	Related Work	108
9.1	Computations on Haplotypes	109
9.2	Computations on STRs	111
9.3	Computations on Genotypes	112
9.4	Other Works	113
10	Conclusions and Future Work	114
	Bibliography	127
A	Data-sets Used in Experiments	128
	Vita	

List of Tables

2.1	Ranges of IBD segments in cM based on degree of relationship.	19
3.1	Starting SNP positions of 0 to 3 cM along each chromosome .	28
4.1	Garbled circuit: AND gate truth table	41
5.1	Input fields	49
7.1	Encoding of alleles into their three-bit binary equivalents . . .	70
7.2	Truth table of SNP-matching circuit	72
7.3	Boolean expression for less-than- T_2 circuit	79
7.4	Total number of individual gates used by SNP-matching circuits in a frame-matching circuit	81
7.5	Total number of individual gates used by half-adder circuits in a frame-matching circuit	81
7.6	Total number of individual gates used by full-adder circuits in a frame-matching circuit	83
7.7	Total number of individual gates used in a frame-matching circuit containing 2^n SNPs	85

8.1	Data-set descriptions	92
8.2	Percentage of shared common SNPs	96
8.3	Number of related cases in RTASA	100
8.4	GCA relatedness results for RTASA cases	101
8.5	HBA relatedness results for RTASA cases (<code>cMperFrame = 5 cM</code>)	102
8.6	HBA relatedness results for RTASA cases (<code>cMperFrame = 25 cM</code>)	103
8.7	GCA running times for different cases	106
A.1	Ancestry family data sets	128
A.2	23AndMe data sets	129
A.3	Ancestry data sets	129
A.4	FTDNA data sets	130

List of Figures

2.1	A pictorial view of DNA	8
2.2	Sample of genotyped data	12
2.3	IBD _{half} inference	14
2.4	Rules for DNA matching.	16
2.5	A sample DNA matching segment.	17
2.6	Segments of DNA shared within a family	18
3.1	centiMorgan division in chromosome 1	23
3.2	Division of chromosome 1 by T_1 sized frames.	24
3.3	Overlapping frames	25
3.4	Chromosome 1 Rutgers map files snippet	27
3.5	centiMorgan data-set	29
3.6	Data in the format of 23AndMe.com	32
3.7	Data in the format of AncestryDNA	33
4.1	Garbled outputs	41
4.2	SHA-256 hash value generator	45
5.1	Server-client architecture	48

5.2	Finding common SNPs for each chromosome	50
6.1	Dividing frames into two strings	57
7.1	SNP-matching circuit	71
7.2	Adder tree for a frame with 256 SNPs	74
7.3	Half-adder circuit	75
7.4	3-bit full-adder circuit	77
7.5	High-level circuit depiction of a frame-matching circuit for 2^n SNPs.	80
8.1	Data characteristics of test user's DNA data	95
8.2	Frames and centiMorgan distribution for TASA and TADA test cases for GCA	97
8.3	Frames and centiMorgan distribution for TASA and TADA test cases for HBA with frame-size = 5 cM	98
8.4	Excluded frames distribution of HBA for TSRG and TDRG test cases with cMperframe size of 5 cM and 25 cM	99
8.5	Frames and centiMorgan distribution for TASA and TADA test cases for HBA with cMperFrame = 25 cM	100
8.6	Running times for HBA	105
8.7	Running times for GCA	107

List of Symbols, Nomenclature or Abbreviations

<i>SHA</i>	Secure Hash Algorithm
<i>DNA</i>	Deoxyribonucleic Acid
<i>GC</i>	Garbled Circuit
<i>GCA</i>	Garbled Circuit Approach
<i>HBA</i>	Hashing Based Approach
<i>2PC</i>	Two Party Computation

List of Algorithms

1	Computing first centiMorgan positions for each chromosome file	30
2	Garbled-circuit protocol	39
3	Finding common SNPs	51
4	Nonce generation	55
5	Checksum exchange	60
6	Hash exchange	61
7	DNA matching algorithm in HBA	63
8	Generate Boolean expression for less-than- T_2 circuit	78

Chapter 1

Introduction

Genetic information contained in DNA can support or cast doubt on a lineage. DNA analysis of two individuals can reveal the degree of relationship between them if they share a common ancestor. According to MIT Technology Review estimates, as many as 26 million people have had their DNA data added to the commercial health and ancestry databases [47]. The number of tests recorded in 2018 was more than the number of tests recorded in all previous years combined.

The emergence of direct-to-consumer genetic testing has empowered consumers to discover their ethnic heritage, family tree, health susceptibility, personality traits, genetic constituents, etc. A consumer can acquire their DNA data by following two simple steps: registration and payment for obtaining the services and supplying the testing company with a sample of saliva (in most of the cases).

Although most of the companies focus on ethnic heritage and ancestry, some companies like 23andMe¹ also provide insights into health predispositions as well [26]. They test certain sections of the DNA and report if a person is more likely to develop diseases like type-2 diabetes, Parkinson's disease, Alzheimer's disease, breast cancer etc. They provide services like ancestry, health, traits, etc. as part of packages as well. So, people often buy such packages and consent to test and share their sensitive private data.

Suppose Alice is a customer of 23andMe who has recently taken a genealogical DNA test. She gets her genetic material analyzed by a micro-array that samples her DNA data. The results of this test is then fed into the company's database. A DNA-matching algorithm is run to match Alice's DNA with its existing members' DNA data. Based on the matches, the testing company provides her with a ranked list of individuals. The company also allows Alice to download her raw DNA data.

DNA data can reveal associations with known medical conditions (for example Alzheimer's disease), so publishing this information has certain risks [44]. Suppose Alice's test results has indications of some medical condition that she has or may have in future. This kind of information can be used to discriminate against her by a health insurance or a life insurance company. For genealogy-oriented tests, companies intentionally do not test certain sections that have known medical implications. In spite of the omission of the aforementioned sections, it is probable that certain tested sections, which

¹<https://www.23andme.com>

are not known to have medical implications right now, may have medical associations discovered in the future.

GEDMatch² is an open-data personal genomics database and genealogy website. The members of this community get their DNA tested at some other company and upload the results in GEDMatch's database. By uploading their DNA data into GEDMatch's database they can find relatives who have also uploaded their data in GEDMatch's database. GEDMatch also allows matches between people who have tested at different testing companies. The service provided by GEDMatch is free of cost but it poses a greater data security risk as well. According to a study by a team of researchers at the University of Washington, GEDMatch is open to multiple kinds of security risks [37]. The researchers created a research account and uploaded some experimental profiles that they obtained from multiple databases of anonymous users. With these profiles they did one-to-one comparisons with a target profile and gathered 92% of the target's sequences with about 98% accuracy. Similar kinds of attacks and data-privacy issues in GEDMatch and other direct-to-consumer testing companies are also highlighted in the works published by Edge et al. [14] and Ney et al. [40].

In 2019, GEDMatch was sold to Verogen [52], a forensic genetics firm in San Diego. Despite the fact that Verogen has promised to increase the security features of GEDMatch, many users are deleting their accounts. This is due to the news that in the past law enforcement agencies have used GED-

²<https://www.gedmatch.com>

Match to solve criminal cases. Verogen intends to make money by providing access to the database and tools for DNA analysis, so it may be a matter of time until users realise their data-privacy risks and opt out of the system.

In 2018 and 2019, American law-enforcement agencies, including the FBI, used GEDMatch and Family Tree DNA (FTDNA) to upload the DNA samples collected from crime scenes [20]. Specifically, in the *Golden Gate Killer* case, the California law enforcement agency identified 10 to 20 distant relatives of the killer after uploading the DNA profile to GEDMatch [28]. With the help of a genealogist, the investigators built up a large family tree and were able to identify former police officer Joseph James DeAngelo as a suspect. Later on the DNA profile of the killer, collected from the scene, and that of DeAngelo were matched and he was subsequently arrested in 2018. The justice served comes at a cost of legal precedent and raises questions about ethical concerns on the unprecedented use of genetic genealogy.

Similar approaches could be followed by a rogue or a repressive government to identify and punish a dissident who had left traces of DNA evidence for doing activities that the government disapproves of. Therefore, due to these risks, certain individuals may be unwilling to share or engage in any action that may publish their DNA data to corporations, governments or to individuals who may harvest these data on behalf of these entities.

All these security breaches and issues have hindered data privacy for the consumers. It is imperative that a solution should be created where users can find relatives without having to compromise on their DNA data privacy. In

this research, we aimed to provide a software implementation where individuals can verify their relatedness while ensuring data privacy. One key factor in a two-party computation scenario is to acknowledge the legitimacy of the data provided by the involved users. We can not control whether either party uses legitimate data, so we cannot stop a relative from pretending not to be related. But, we will make protocols that will stop someone from appearing to be related when they are not, assuming that they cannot find the DNA of an actual relative. These type of cases will lead to replay attacks, which we will handle. We also assume actual relatives are not malicious. Given this privacy setting, we have implemented two different software approaches that help the users to keep their data private:

- Hashing-Based Approach
- Garbled-Circuit Approach

The objective of our work is to define, design and implement our intended approaches and then compare them in terms of security, execution times and correctness of the results (false negatives and positives).

The rest of the thesis is structured as follows:

- *Chapter 2* provides background on the genetic-genealogy aspect of the thesis.
- *Chapter 3* deals with the implementations of the topics discussed in the previous chapter.

- *Chapter 4* provides background on the data privacy aspect of the thesis.
- *Chapter 5* provides an in-depth information regarding data preprocessing steps that need to be done before data-privacy techniques are applied.
- *Chapter 6* explains the hashing-based-approach.
- *Chapter 7* explains the garbled-circuit-approach.
- *Chapter 8* describes the experimental setup and explains the results.
- *Chapter 9* describes a few works related to our thesis in brief.
- *Chapter 10* concludes with the results and provides directions for future work.

Chapter 2

Background

This chapter explores the genealogical aspect of the thesis. The constituents of DNA are discussed in brief to establish the concepts needed to understand the thesis. This chapter also provides information on how the genealogical tests are performed, how DNA matches between two individuals, explains SNPs and finally it establishes the DNA-matching algorithm that is used later on in the thesis.

2.1 DNA

DNA (deoxyribonucleic acid) is made up of molecules called nucleotides, and these nucleotide molecules contain four types of nitrogen bases: adenine (A), thymine (T), guanine (G) and cytosine (C). DNA is coupled as thread-like structures called chromosomes. A human cell contains 23 pairs of chromo-

somes, for a total of 46 chromosomes. The first 22 pairs of these chromosome are called autosomes and these are present in both males and females. The twenty third chromosome is known as the sex chromosome and it differs in both males and females. Females have two copies of the X chromosome, whereas males have one copy of each of the X chromosome and the Y chromosome [1]. Fig 2.1 shows a pictorial view of a double-stranded DNA molecule with *A*, *C*, *T* and *G* nucleotides.

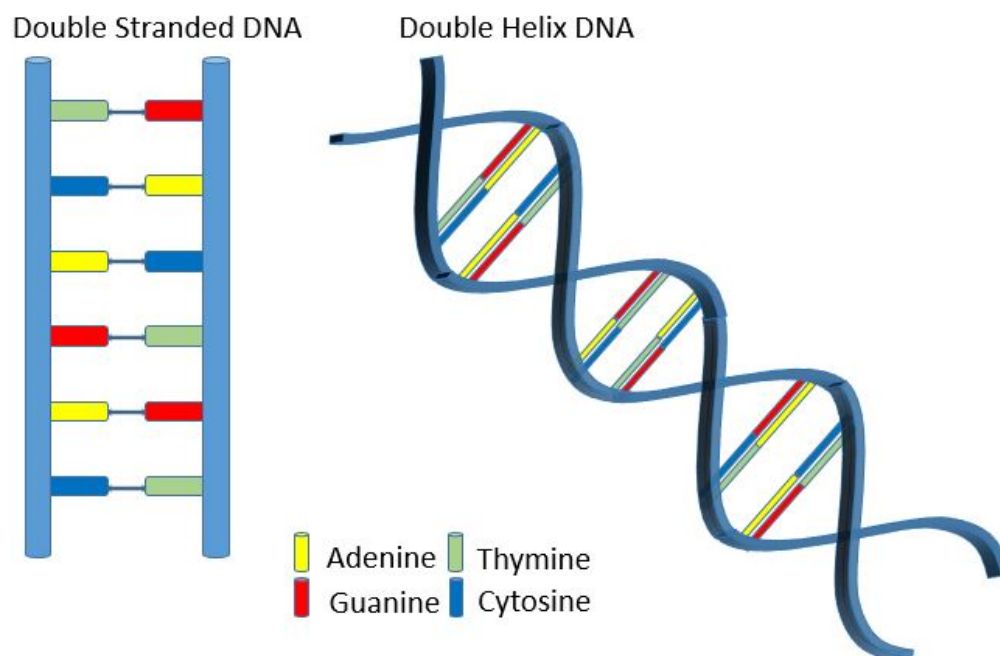


Figure 2.1: A pictorial view of DNA

These nucleotides *A*, *C*, *T* and *G* are present at base locations in a human genome and their order determines the genetic code. Also, some variations exist in the nucleotides at certain positions in a genome; e.g., at a particular

location, nucleotide A is present for most of the population, but in some significant minority of individuals, nucleotide T is present. If more than 1% of a population does not carry the same nucleotide at a specific position in the DNA sequence, then this variation can be classified as a SNP (single nucleotide polymorphism) [38]. The two possible nucleotides A and T are known as *alleles* for that position. DNA is passed from parents to child, so a child inherits the alleles from their parents¹. Thus, an individual will match with their siblings, parents, grandparents, aunts, uncles and cousins at some of these SNPs. However, these matches will be less common among distant cousins. This genetic information and the analysis of SNPs can be used to trace an individual's ancestry and their lineages.

2.2 Genealogical Tests

Genealogical tests are DNA-based tests which inspect specific locations in a genome to study and examine an individual's ancestry. Generally, there are three types of genealogical DNA tests performed:

- *Autosomal and X-DNA*: The autosomal and X-DNA tests examines the chromosome pairs 1-22 that are inherited from both the parents. It also examines the X-chromosome, which follows a complicated hereditary pattern.
- *Y-DNA*: A Y-DNA test examines the Y-chromosome which is passed

¹except for mutations, which are rare

from a father to his son, so this test is only taken by males and can be used to explore the paternal ancestry.

- *mtDNA*: A mtDNA test looks at the mitochondrial chromosome, which is passed from a mother to a child, so it can be used to explore the maternal ancestry.

2.2.1 Autosomal Testing

The autosomal test is widely used by companies for consumer DNA tests to establish relationship between individuals. Some of the companies offering autosomal tests include the following: MyHeritage², Ancestry.com³, FamilyTree DNA (FTDNA)⁴ and 23andMe.

Any two individuals who share DNA in significant regions must certainly share common ancestry. Generally, longer shared regions between individuals mean closer relationship between them. So, siblings have longer matching DNA regions than cousins. The focal point of our work is DNA matching between two individuals using autosomal DNA testing.

2.3 Genotyping

Genotyping is the process through which an individual can find the constituents of their genes and DNA sequences as it reveals the alleles at the

²<https://www.myheritage.com>

³<https://www.ancestry.com>

⁴<https://www.familytreedna.com>

selected SNP locations in the genome. Figure 2.2 shows an example of a genotyped data of an individual. At a particular location in a genome the individual will have one allele inherited from their mother and one allele inherited from their father. If the alleles match, the person is *homozygous* at that location. If the alleles do not match at a location, then it is known as *heterozygous*. In Figure 2.2, the location 720117 is homozygous with value *AA* and the location 836671 is heterozygous with value *CT*. Without any additional information and expensive DNA analysis process, we cannot for certain determine which allele is inherited from which parent, i.e., we cannot determine whether *C* in the genotype *CT* came from the father or the mother.

There are other techniques like sequencing, haplotype estimation, phasing, etc. that can be used to find the constituents of DNA and used for determining ethnicity and ancestry. The direct-to-consumer services typically use genotyping to ascertain the DNA constituents as it is comparatively cheaper and easy to understand. Since our work deals with the direct-to-consumer DNA testing data, we will focus only on genotyping.

2.4 DNA-Matching Algorithm

A snippet from the sample genotyped data is shown in Figure 2.2. In the figure, column *RSID* (reference SNP cluster id) is a unique label (rs followed by a number) which is used by researchers to signify a specific SNP. The column

RSID	CHROMOSOME	POSITION	Genotype
rs4477212	1	720117	AA
rs3094315	1	752566	AA
rs3131972	1	752721	GG
rs12124819	1	776546	AA
rs11240777	1	798959	GG
rs6681049	1	800007	CC
rs4970383	1	836671	CT
rs4475691	1	846808	TC
rs7537756	1	854250	AG
rs13302982	1	861808	GG
rs1110052	1	873558	TT

Figure 2.2: Sample of genotyped data

CHROMOSOME identifies the position of the SNP in a particular chromosome, and the values range from 1-22. The column *POSITION* represents the presumed location of the SNP from the starting point of the chromosome. The column *GENOTYPE* reveals the values of the alleles at that particular position. A *base-pair* is a combination of two nucleotides at a location along the chromosome represented by the alleles an individual receives from their parents, e.g. “AT”, “CG”, etc.

A DNA-matching algorithm inspects the constituents of the DNA and finds matching sequences between two individuals. Two individuals who share long matching segments are almost certainly closely related. A DNA data sequence can be visualized as a string of genotypes. A string *AAAAGGAAGGCCCTTCAGGGTT* between location 720117 and 873558 can be formed from Figure 2.2.

Pattern-matching algorithms [7, 54] and some bio-informatics tools [2] use the data in its raw form to extract matching sequences. Typically a matching

algorithm traverses through the entire string and looks for matches. So, a general pattern-matching algorithm will not be a suitable solution for our work as the intention is to hide the raw data to ensure data privacy. In our thesis, we need an algorithm that can do pattern matching without exposing a portion of raw data.

2.4.1 Opposite- Homozygote Technique

One of the naive approaches to determine genetic relatedness between two individuals could be done in the following 3 steps: at the first step, find all the common SNPs between both users, at the second step match alleles for all the locations and at the third step count all the matching alleles. Based on the percentage of matching alleles, relationship can be inferred. This approach is not used by the major test companies as it cannot easily distinguish between relatives and members of the same general population, e.g. two Swedes might share more alleles than either would share with an actual close cousin of theirs with mostly Indian ancestry. Instead, most direct-to-consumer testing companies use the “Opposite-Homozygotes” technique that aims to find matching segments rather than counting matching alleles.

The autosomal tests detect long regions on 22 pairs of autosomes where two individuals may share identical sequences. The presence of identical long sequences is suggestive of a fairly recent common ancestor shared between individuals and can determine relationships between them.

In genetic genealogy, a DNA *segment* is defined as the contiguous collection

of alleles segregated between a start and an end location in a particular chromosome. A DNA segment is identical by state (IBS) if two individuals have identical alleles in the SNPs whose locations fall within the segment's boundary. An IBS segment is identical by descent (IBD) if two individuals inherited the alleles in the segment from the same shared ancestor. To infer an IBD segment using genotyped data (where each SNP has two alleles), each pair of genomes should be searched for long segments of alleles that are separated by *opposite homozygotes* [23]. Opposite homozygotes occur at a location in the genome where two individuals are homozygous and their alleles are different from one another. The pair AA-GG and the pair CC-TT are two examples of opposite homozygotes. In between opposite homozygotes, IBS segments are demonstrated by same homozygotes such as "AA-AA" and *half-identical* SNPs such as "AC-CC" [51] (where one allele from an individual should match at least one allele in the other individual).

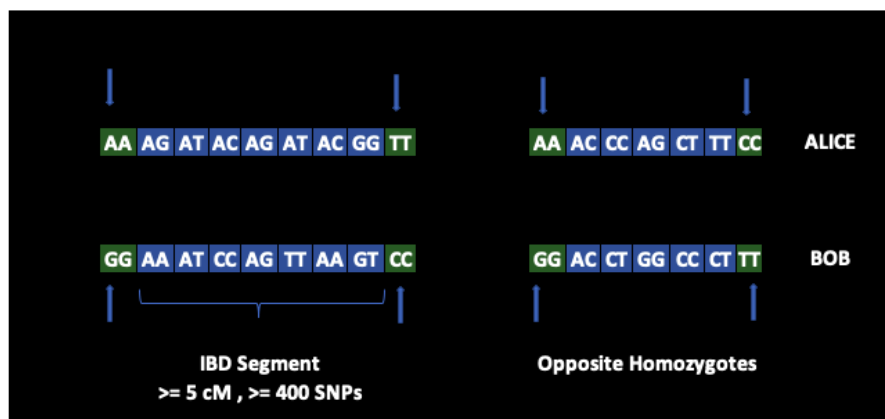


Figure 2.3: IBD_{half} inference

A heterozygous AG will be half-identical to the genotypes AA or GG. A homozygous AA will be a match with AA, likewise CC with CC and so on [51]. The opposite homozygotes appear often to make it difficult to obtain a long match, unless the two individuals are IBD [51], i.e., the length of an IBS segment is long enough to infer it is an IBD segment. IBD_{half} is defined as the sum of the lengths of genomic segments where two individual are IBD [23]. Figure 2.3 sketches how IBD_{half} can be determined within the boundaries of opposite homozygotes. IBD_{half} segments are determined from the genomic data where a series of alleles were IBD within a genome. An IBD segment is indicated in blue and opposite homozygotes separate the IBD segments. The IBD segment should be minimally 5cM and should contain more than 400 SNPs.

In this research, we have implemented the opposite-homozygotes technique for our DNA-matching algorithm to find the matching segments. The details of the implementations are further explained in Section 3.1 and Chapters 6 and 7.

2.4.2 How Is DNA Matched?

The rules and logic for allelic matching between individuals are described in Figure 2.4. For homozygous cases, it shows how a homozygote matches with a homozygote and other heterozygotes. For heterozygotes, we know that most human SNPs are biallelic, i.e., SNPs have a combination of only two alleles (triallelic SNPs are also possible but we ignore them in our com-

putation as they are very low in number). The lower figure shows how a biallelic SNP matches with other homozygote and heterozygote alleles. So, a heterozygote matches with everything that is possible, e.g., a heterozygote ‘AC’ with alleles ‘A’ and ‘C’ will match with ‘AA’, ‘CC’ and ‘AC’.

Homozygote Alleles	Will match to			
AA	AA	AC	AG	AT
CC	AC	CC	CG	CT
GG	AG	CG	GG	GT
TT	AT	CT	GT	TT

Heterozygote Alleles	Alleles in a Biallelic SNP	Will match to		
AC	A, C	AA	CC	AC
AG	A, G	AA	GG	AG
AT	A, T	AA	TT	AT
CG	C, G	CC	GG	CG
CT	C, T	CC	TT	CT
GT	G, T	GG	TT	GT

Figure 2.4: Rules for DNA matching.

Let us consider another example from Figure 2.5, where we have two individuals, Alice and Bob, whose matching segments are distinguished by opposite homozygotes. The first segment from Alice ‘AC-AA-AG-AA-AA-TT-AC’ (contiguous alleles are separated by ‘-’ for clear understanding) matches the first segment from Bob ‘CC-AA-GG-AA-AA-TT-AC’. It shows how half-identical ‘AC’ and ‘AG’ match with ‘CC’ and ‘GG’ respectively. The second segment also contains two half-identicals ‘AC’ with ‘CC’ and ‘CC’ with ‘AC’. So, our DNA-matching problem is different from traditional pattern-matching problems as our strings have same length, are symmetric and we

seek long sub-string matches rather matching all of a pattern. Moreover, both strings (not just a “pattern” string) contain single-character wildcards in the form of heterozygotes. Therefore, we have to keep an account of half identicals, e.g. where ‘AA’ matches with ‘AC’.

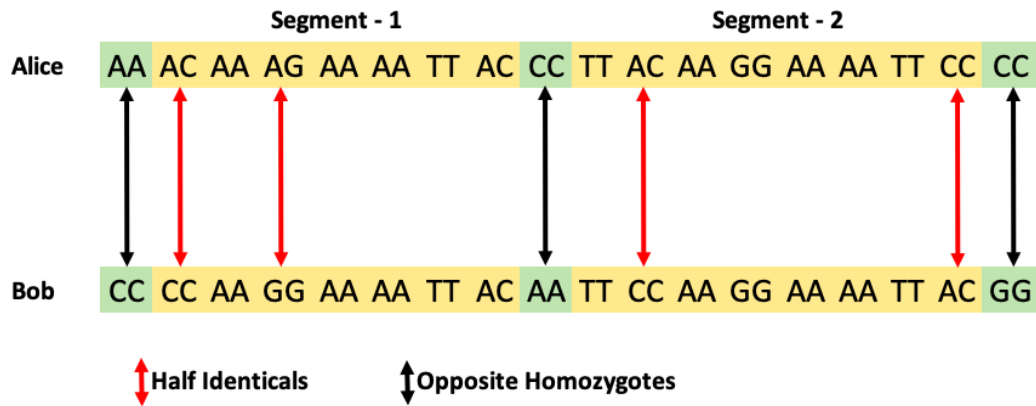


Figure 2.5: A sample DNA matching segment.

Figure 2.6 demonstrates how a DNA sequence is inherited from grandparents to grandchildren. A segment of alleles *ACTC* indicated in orange is inherited by Mary and Richard from their parent John. The same segment, *ACTC*, is found in Mary and Richard’s IBD segment (indicated in green) between opposite homozygotes. The segment *ACTC* is reduced to just *ACT* when it is inherited by Alice and Bob via their respective parent Mary and Richard. Only a part of the segment is shared by Alice and Bob rather than the entire segment from John. It is because of recombination that IBD segments shared by two close relatives will tend to be longer than IBD segments shared by two distant relatives.

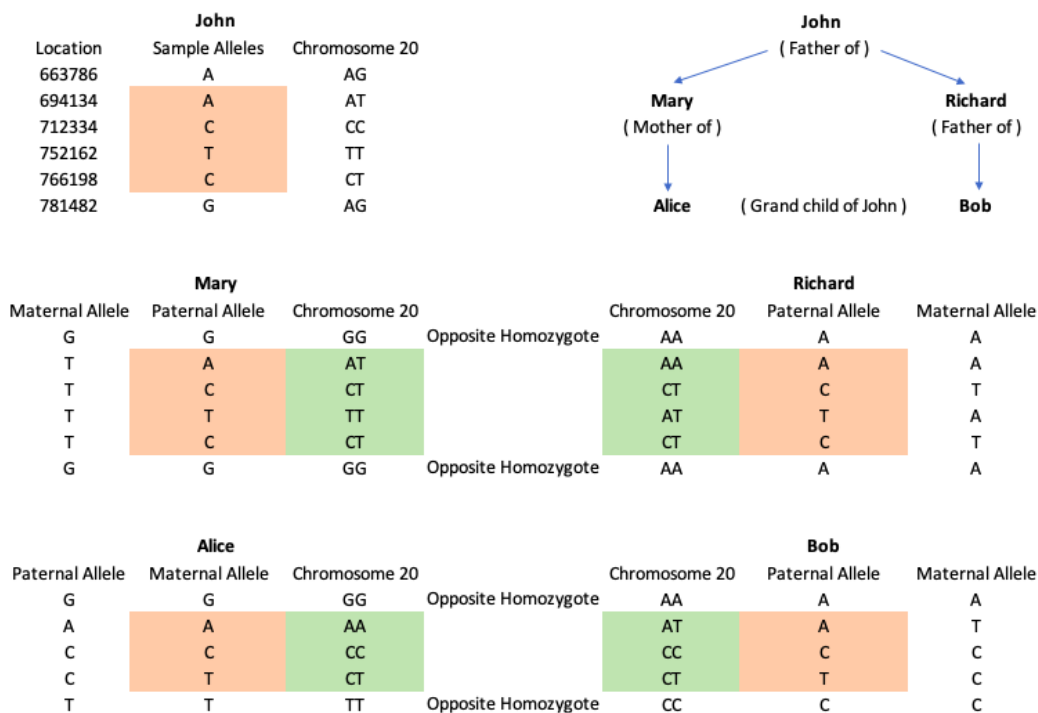


Figure 2.6: Segments of DNA shared within a family

2.5 centiMorgan

A centiMorgan (cM) is a unit in genetic genealogy that is used to measure segment length. It is used to deduce the distance along a particular chromosome. One centiMorgan, on average, roughly corresponds to 1 million base pairs, but the number of base pairs for 1 cM varies along the chromosome as it depends on recombination and other factors [42]. It is defined as a length where the segment would have 0.01 expected recombination events when the segment is being passed to the next generation. The centiMorgan distance does not denote a true physical distance. The total centiMorgans in a re-

ported genotyped data file that we are working with is between 3400 – 3600. Approximately 500,000 SNP locations are used for matching DNA sequences. Hence, we can assume that each centiMorgan will be approximately equivalent to 500000/3500 or about 140 SNPs. We know that centiMorgan are not uniformly divided along a chromosome; according to our experiments discussed in Chapter 8 the number of SNPs contained in a centiMorgan ranges between 50 to 10000 SNPs.

Table 2.1: Ranges of IBD segments in cM based on degree of relationship.

RELATIONSHIP	IBD _{half} RANGE	EXPECTED	SEGMENTS
Parent/Child	3539 – 3748 cM		23 – 29
First cousins	548 – 1139 cM	888 cM	17 – 32
Second cousins	86 – 426 cM	222 cM	10 – 18
Third cousins	16 – 111 cM	55.4 cM	2 – 6
Fourth cousins	0 – 54 cM	13.8 cM	0 – 2

The threshold of long matches to contribute toward IBD_{half} should be minimally 5cM and should contain more than 400 SNPs [23]. Table 2.1 indicates the ranges of IBD segments in cM to infer relationship between two individuals, and the information in the table has been utilised from the ISOGG. For two individuals to be siblings, the matching segments should be approximately 2550 cM [12]; almost 75% of the genome should match ⁵. To conclude a relationship between first cousins, the matching segments should range between 548-1139 cM and the number of shared segments must fall in the range

⁵https://isogg.org/wiki/Identical_by_descent

of 17-32 . As the degree of relationship increases, IBD_{half} decreases [23].

2.6 SNP Reporting

The number of reported SNPs depend on the testing agency, the gender of the individual, etc. So the number of SNPs that we will investigate in our research varies by the individual.

Every direct-to-consumer DNA testing agency uses their own SNP micro-array chip and a human reference genome to determine the set of SNPs that are reported. SNP micro-array chips generate a human genome map which contains data about SNP in each chromosome [34]. The data include details about the SNP identifier (*rsid_id*), chromosome it belongs to and the value of the alleles present.

The human reference genome used by direct-to-consumer agencies has also evolved over time as there are different builds of the genome that have been used by agencies [35]. The current version of human reference genome used is NCBI version 38, patch 13 [18]. However, the results from versions 36 and 37 are still widely used. In our experiments, most of the human genome data sets are built from version 37.

A testing agency obtains the set of SNPs that will be reported from micro-array chips and maps the SNP using the human reference genome. The human reference genome places the reported SNP in its chromosome and also determines its position from the start of the chromosome.

Two individuals will report almost identical sets of SNPs if they are tested at the same agency (using the same micro-array chips and human reference genome). However, two individuals will report non-identical sets of SNPs if they are reported by different agencies using different SNP micro-array chips and builds of the human reference genome.

DNA matching of two parties requires both parties to have a large number of common SNPs between them. Both the parties will verify each common SNP location to determine whether they match for that particular SNP by comparing the genotypes.

Chapter 3

Computational Division of DNA

In this chapter, we discuss the computational division of chromosomes into frames and its related implementations. The division of chromosomes plays a significant role in the latter chapters and is referenced often. This chapter also explains the technical details of how we transformed the centiMorgans data set to find the starting positions of each centiMorgan along each chromosomes.

3.1 Frames

Although the term ‘frame’ has been used in various contexts in computer science, in this thesis we have used the term *frame* to divide the SNPs contained

in each chromosomes. According to the opposite-homozygotes technique, as discussed in Section 2.4.1, we need to find long IBD segments separated by opposite homozygotes. Each IBD segment should contain at least 5cM and 400 SNPs as has been established in Section 2.5. So to find IBD segments, we will divide the chromosomes into frames of size 5 cM each and check for matching segments in each 5 cM-sized frame. Therefore, if we have a matching frame then it corresponds to a matching IBD segment.



Figure 3.1: centiMorgan division in chromosome 1

Figure 3.1 illustrates an approximate division of centiMorgans in Chromosome 1 without the framing division. In the figure, the chromosome data starts from the first SNP reported and the base centiMorgan position is ‘0’. For the sake of simplicity, the division of each centiMorgan in the current and following figures in the thesis is shown as uniform. However, the number of SNPs and base pairs in each centiMorgan (and hence each frame) varies. After the implementation of frames, chromosome 1 DNA data can be perceived from Figure 3.2. This division offers a set of frames as $\{[0, 5), [5, 10), [10, 15), [15, 20), \dots\}$. The objective of implementing frames is to find IBD segments with a minimum size of 5 cM; any segment less than 5 cM is not an IBD segment and can be ignored. IBD segments greater than 5 cM can

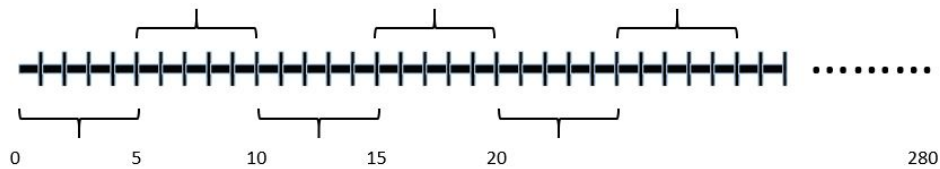


Figure 3.2: Division of chromosome 1 by T_1 sized frames.

also be determined using this division and it is explained in the following paragraphs of this chapter.

Most of the time, the matching segments may not completely lie within the boundaries of a frame. Suppose we have a matching segment that exists between 6 cM and 10 cM: it would go undetected. So, we have to create some kind of overlapping regions such that these regions do not go unnoticed.

3.1.1 Overlapping Frames

To create overlapping regions, we will create the next frame from the starting position of next centiMorgan. With the overlapping frames, we can generate a set of frames as $\{[0, 5), [1, 6), [2, 7), [3, 8), [4, 9), [5, 10), [6, 11), [7, 12) \dots\}$. Overlapping frames can be visualized from Figure 3.3. In our setting, all the frames start from the first location at every centiMorgan. If there exists an IBD segment of 5 cM from the beginning of any centiMorgan, it can be detected using the overlap division. An IBD segment of 7 cM between 2 to 9 cM, one that starts from the first position of 2 cM and ends at the last position of 8 cM, corresponds to the matching set of three frames, i.e.

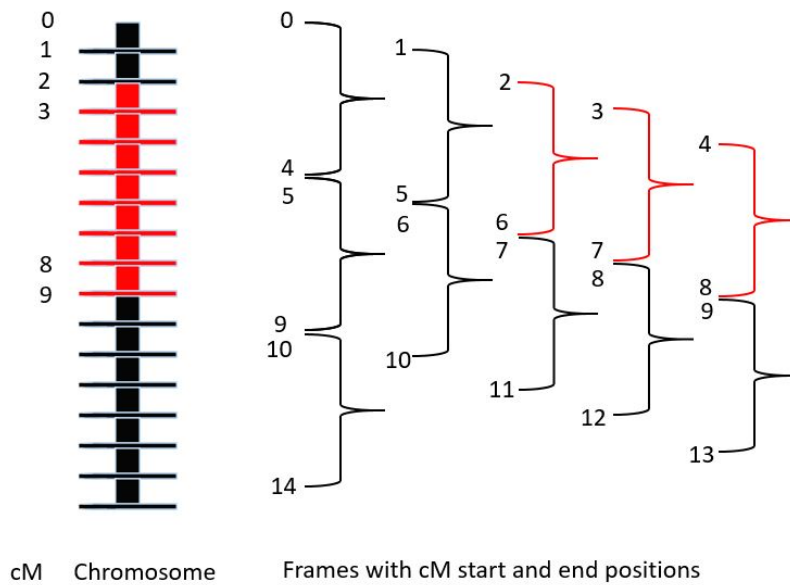


Figure 3.3: Overlapping frames

$\{[2, 7), [3, 8) \text{ and } [4, 9)\}$. Figure 3.3 demonstrates how a 7 cM IBD segment between 2 to 8 cM is determined by the set of $\{[2, 7), [3, 8) \text{ and } [4, 9)\}$ frames, and the matching segment in the chromosome data has been marked red from the beginning of 2 cM to the end of 8 cM. The matching frames are also designated by color red.

In this frame division, we start our frames exclusively from the first position of each chromosome. If there exists an IBD segment of length 5.5 cM between 2.9 cM and 8.4 cM, then an IBD segment of 5 cM will be reported with the frame $[3, 8)$. Supposing there exists an IBD segment of exactly 5 cM between 1.2 cM to 6.2 cM, this entire IBD segment will be missed as our division does not involve generation of frames in between any centiMorgans. Often, IBD segments for two related individuals span much more than 5 cM, so the

limitation of starting a frame from the first position would not yield too many missed IBD segments.

3.2 Start Position of centiMorgans

The first step for creating a frame needs information about the positions of the centiMorgans. So it is imperative to find the first SNP position of each centiMorgan at each chromosome. The centiMorgan data are not reported by any genealogical testing agency, and this can be verified by looking at the sample genotyped data in Figure 2.2. In order to build the centiMorgan data, we need to find a data source and build an application that can determine the data we need for our thesis.

For our thesis, we have used the Rutgers Map Files [35] to build our centiMorgan data. The data set contains details of each SNP along each chromosome. There are 22 data files, one for each chromosome. A snippet from the chromosome 1 data file is shown in Figure 3.4. We have trimmed down the number of columns from the original data file to just 4 columns for better perception. The first column, *Marker_name*, references the rsid or the SNP ids for each SNP in the chromosome. The second column, *Type*, signifies the type of base pair location (STS or SNP), but we are concerned with SNP only. The third column, *Build37-start-physical-position*, gives the position of the SNP from the beginning of the chromosome 1. The fourth column, *Sex-averaged-start-map-position*, gives the centiMorgan value for that partic-

ular SNP. Roughly each data file contains 700,000 to 1,500,000 rows.

Marker_name	Type	Build37_start_physical_position	Sex_averaged_start_map_position
D1S2217	STS	797159	0
rs148673408	SNP	797208	0.000055784
rs181840775	SNP	797211	0.000059199
rs186952811	SNP	797225	0.000075137
rs76631953	SNP	797281	0.000138891
rs111739932	SNP	797325	0.000188983
rs149025072	SNP	797377	0.000248182
rs58013264	SNP	797440	0.000319905
rs200972937	SNP	797474	0.000358612
rs116442413	SNP	797506	0.000395042

Figure 3.4: Chromosome 1 Rutgers map files snippet

We parsed each data file to compute the starting SNP positions of each centiMorgan along each chromosome. The details of our computation is shown in Algorithm 1. The output of the algorithm gives us a 2-D array where the first column represent the chromosome it belongs to, and so it ranges from 1 to 22. The second column is an array of integers which contains the list of starting positions of each centiMorgan. The index of the array of integers determines the centiMorgan value and the value at the index is the starting SNP position for that particular centiMorgan. A small portion of output with the starting SNP position of the first 4 centiMorgans along each chromosome is tabulated in Table 3.1. A snippet from the actual data set that the algorithm generates is shown in Figure 3.5. This data set is utilised by both the approaches of our software implementation. This data-extraction algorithm is executed only once to create the data set, and its output has been inserted into the source code of other programs. So, the software built for

verifying relationship using DNA data while ensuring data privacy, described in Chapters 6 and 7, does not require execution of this algorithm.

Table 3.1: Starting SNP positions of 0 to 3 cM along each chromosome

Start SNP Position for cMs Chromosome	0	1	2	3
1	0	1131729	1418789	1705371
2	0	885649	1604531	2251840
3	0	709556	1158586	1536267
4	0	1365417	2224331	2907731
5	0	521242	884503	1213598
6	0	497198	772527	1051246
7	0	1052871	1836669	2535743
8	0	949443	1395789	1823152
9	0	679119	1011757	1329113
10	0	520164	930647	1444301
11	0	545597	847884	1155534
12	0	738274	1143228	1531028
13	0	20841237	21454177	21907959
14	0	20771918	20992484	21180482
15	0	23076235	23333893	23636813
16	0	373301	622325	887933
17	0	284012	478595	686392
18	0	412629	672204	997498
19	0	1398143	1648775	1890208
20	0	292926	473572	657014
21	0	15017120	15289230	15573653
22	0	17230886	17378089	17529780

```

cM = new int[23][];
cM[0] = null;
cM[1] = new int [] {0,1131729,1418789,1705371,2003293,2296659,2601009,2917698,3229048,3551709,3995287,4219407, ....};
cM[2] = new int [] {0,885649,1604531,2251840,2895684,3378023,3791034,4111106,4463078,4808381,5165481,5551324, ....};
cM[3] = new int [] {0,709556,1158586,1536267,1879573,2224996,2458778,2713403,2951741,3162247,3378010,3623944, ....};
cM[4] = new int [] {0,1365417,2224331,2907731,3519917,4061491,4531684,4950579,5325130,5651445,5946064,6238460, ....};
cM[5] = new int [] {0,521242,884503,1213598,1532746,1838201,2130925,2396771,2645796,2982369,3306358,3700168, ....};
cM[6] = new int [] {0,497198,772527,1051246,1344244,1639428,1947767,2271652,2604477,2990739,3364512,3710727, ....};
cM[7] = new int [] {0,1052871,1836669,2535743,3181526,3755014,4262680,4746869,5279433,5764469,6357283,6963887, ....};
cM[8] = new int [] {0,949443,1395789,1823152,2213596,2578964,3080697,3527346,3920054,4239459,4535153,4891325, ....};
cM[9] = new int [] {0,679119,1011757,1329113,1626498,1914662,2198752,2450986,2751271,3122294,3629243,4018146, ....};
cM[10] = new int [] {0,520164,930647,1444301,2172509,2647191,2900641,3134569,3329584,3511415,3699620,3866515, ....};
cM[11] = new int [] {0,545597,847884,1155534,1464850,1784330,2115695,2457886,2785499,3111559,3478500,4222588, ....};
cM[12] = new int [] {0,738274,1143228,1531028,1908745,2268481,2633770,3016190,3342282,3608520,3858791,4096508, ....};
cM[13] = new int [] {0,20841237,21454177,21907959,22290850,22603804,22948326,23302418,23657136,24014353, ....};
cM[14] = new int [] {0,20771918,20992484,21180482,21347224,21496317,21638941,21778926,21932301,22136122, ....};
cM[15] = new int [] {0,23076235,23333893,23636813,24021294,25147485,25666165,26023552,26303580,26505157, ....};
cM[16] = new int [] {0,373301,622325,887933,1170781,1476794,1806427,2276738,2923846,3538003,4196538,5152154, ....};
cM[17] = new int [] {0,284012,478595,686392,916316,1176029,1486143,1850168,2274704,2685265,3062033,3460023, ....};
cM[18] = new int [] {0,412629,672204,997498,1505590,1973801,2276661,2484145,2708109,2997137,3265553,3520400, ....};
cM[19] = new int [] {0,1398143,1648775,1890208,2122195,2344415,2559414,2770398,2971737,3140417,3328178,3512450, ....};
cM[20] = new int [] {0,292926,473572,657014,857705,1058561,1485403,1868544,2228613,2612327,3093221,3868950, ....};
cM[21] = new int [] {0,15017120,15289230,15573653,15880487,16207072,16555253,16928766,17438020,18005187, ....};
cM[22] = new int [] {0,17230886,17378089,17529780,17688402,17854613,18029673,18212895,18384517,18597145, ....};

```

Figure 3.5: centiMorgan data-set

3.3 Genotyping Errors

In the real world, genotyping errors with a frequency of about 0.1% exist in the genotyped data [49]. Due to a genotyping error, an actual genotype AA can be reported as AC , CC , II (or unknown), etc. There are some possibilities that a genotyping error could cause. The genotyping error can lead us to false negatives when matching frames, as it can report false opposite homozygotes. Yet other genotyping errors can be harmless or even lead to false positives. Each frame contains SNPs over a 5 cM range, but the total number of SNPs contained in each frame will be different. The total number of SNPs contained in a particular frame is denoted by T_1 . To tackle the genotyping error, we assign a variable T_2 with the number of SNPs contained within the frame (T_1) divided by 1000, i.e. $\lceil T_1/1000 \rceil$. So, if we have a frame that contains $T_1 = 700$ or $T_1 = 1700$ SNPs, T_2 will be 1 and 2 respectively. The usage of the variable T_2 are discussed in the upcoming Chapters 6 and 7.

Algorithm 1: Computing first centiMorgan positions for each chromosome file

Data: 1 of the 22 chromosome data files

Result: positions: list of first SNP position for each centiMorgan along that chromosome

```
1 positions[]
2 positions[0] ← 0
3 counter ← 0
4 while line ← readLine() ≠ null do
5     columns ← split(line)
6     if extract(line, type_field) = "SNP" then
7         cM ← rounded(extract(line, map_position))
8         next_cM ← cM + 1
9         while cM < next_cM & line ← readLine() ≠ null do
10            | cM ← rounded(extract(line, map_position))
11        end
12        if line ≠ null then
13            | positions[counter++] ← extract(line, base_pair_position)
14        end
15    end
16 end
17 return(positions)
```

Harmless Genotyping Errors Suppose we have two individuals, Alice and Bob, who have genotyped their DNA data. For a particular location *1234567* on chromosome 1, Alice and Bob were reported as *AA* and *AC* respectively. Alice was reported correctly, whereas Bob on that particular location was *AA*, but due to a genotyping error his result was reported as *AC*. Although the genotyping error changed the reported genotype for Bob, the DNA-matching algorithm will process this location as a match¹. Since

¹assuming there are no opposite homozygotes elsewhere in the frame.

the algorithm reports on frames, thus the respective frame will be reported as a match. So, this type of genotyping error is harmless to the DNA-matching algorithm, as it does not affect the matching results.

False Positives and Negatives Sometimes a genotyping error can lead to false positives as well. For a particular location 1234567 on chromosome 1, Alice has been reported correctly as AA , whereas Bob on that particular location was CC but due a genotyping error it was reported as AC . If Bob was reported correctly, the matching algorithm would have reported a mismatch for the frame, as AA and CC are opposite homozygotes. Since the reported genotype of Bob was AC , the DNA matching algorithm could possibly report this frame as a match. This case of genotyping errors will give us *false positives*.

For the same location 1234567 on chromosome 1, suppose Bob was actually AC and due to a genotyping error it was reported as CC . This could possibly yield the corresponding frame as a mismatch as the reported genotypes are opposite homozygotes. This case of genotyping errors will give us *false negatives*.

Data Omissions A genotyping error could lead a SNP to be reported as “ – – ” at a location. This genotype is reported by testing companies when the alleles at that location can not determined. In other words, they are unknown. During the data-cleaning process, occurrences of this type of genotype are removed from the data sets. So, if a genotyping error leads to

this genotype then this location is omitted from data set and subsequently not sent to the DNA-matching algorithm for further processing.

3.4 Data Preprocessing and Cleaning

As we know, two individuals can take the DNA test at two different test companies. So, the genomic data of both the individuals are bound to be represented in different formats. By looking at Figure 3.6 and 3.7, it is evident that the alleles in both data-sets are represented differently. The allelic value given by 23AndMe contains a string like CC. But the alleles given by AncestryDNA are two separate characters for allele1 (A) and allele2 (A).

rsid	chromosome	position	genotype
rs75333668	1	762320	CC
rs114525117	1	798959	GG
rs4475691	1	846808	CC
rs200686669	1	861349	CC
rs13302982	1	861808	GG
rs201186828	1	865545	GG
rs148711625	1	865584	GG
rs146327803	1	865625	GG
rs41285790	1	865628	GG
rs140751899	1	865662	GG
rs145442390	1	865665	GG
rs9988179	1	865694	CC

Figure 3.6: Data in the format of 23AndMe.com

rsid	chromosome	position	allele1	allele2
rs4477212	1	82154	A	A
rs114525117	1	752566	C	C
rs3131972	1	752721	C	T
rs12562034	1	768448	A	C
rs12124819	1	776546	G	G
rs11240777	1	798959	C	C
rs6681049	1	800007	C	C
rs4970383	1	838555	G	G
rs4475691	1	846808	G	G
rs7537756	1	854250	G	G
rs13302982	1	861808	G	G
rs1110052	1	873558	C	C

Figure 3.7: Data in the format of AncestryDNA

The SNPs selected by different companies also tend to differ and the companies do not sample all the SNPs. In Figures 3.6 and 3.7, only the SNPs with rsid ‘rs114525117’ and ‘rs4475691’ are common between the samples. The difference between the number of SNPs reported by different companies depends on the SNP micro-array chip and the human reference genome used. If both the companies use different SNP micro-array chips then the difference will be big. In our experiments, we have found the number of common SNPs in all test cases that we executed, and these results are reported in Chapter 8. In our approach, we will only be comparing the SNPs which are present in both Alice and Bob’s genotyped data, irrespective of the positions in which they are reported.

Sometimes, different companies report SNPs at different positions in the

genome. An SNP id ‘*rs114525117*’ can be reported at location ‘798959’ on chromosome ‘1’ by one company but at location ‘752566’ on chromosome ‘1’ by the other, as shown in Figures 3.6 and 3.7. This kind of change in reported SNP positions occurs every few years as the reference genome is fixed for bugs. The mismatches in the location of reported SNPs come from the use of different versions of the reference genome used by different companies.

Although these changes are not too large in number, some of these corrections have affected the order in which certain SNPs are believed to occur. We make sure that this does not impact the matching algorithm much. As we are going to compare only the SNPs which are common to both the individual’s genotyped data, these changes would not affect our DNA-matching scheme. The DNA-matching algorithm matches alleles only if the SNP identifiers (*rs_id*) match between individuals. In this case SNP id ‘*rs114525117*’ will be matched although they are reported at different locations in the genome. For computational purposes the SNP locations of one individual will be updated with the locations of the other individual, so that both individuals have same genomic locations in their data sets. (If same location is represented by different *rs_ids* for individuals, then this location will be discarded.)

Data cleaning steps verify the following rules:

1. All the values in a row should be in proper data format; any improper value is discarded.

2. Rows with chromosome value between 1 and 22 only are accepted, otherwise discarded.
3. Only permissible alleles are accepted in our data set,
 - If the alleles are a combination of A, C, G, T we keep them.
 - We discard alleles “- -” and combinations of I and D .
4. Only unique locations are accepted; any location reported more than once is discarded.
5. Only SNP identifiers starting with rs are permissible, and other values are discarded.

Chapter 4

Data-Privacy Techniques

The focal point of our thesis is DNA matching between two individuals using autosomal DNA testing. Data-privacy risks are involved while matching DNA sequences between individuals. So, efficient techniques for computation and security must be used. Secure two-party computation and hashing techniques can provide the required level of computation and security to achieve the objectives. This chapter discusses the data-privacy aspect of our thesis.

4.1 Data-Privacy Issues

Data-privacy risks do not only arise from the kind of technique used but also from merely allowing certain questions to be answered. It is imperative in secure two-party computation to limit the number of questions asked and also to assess the quality of questions asked. If we want to compute “Is Alice’s

claimed salary equal to Bob's?" without leaking any additional information, with about a million queries Alice can find the exact salary of Bob, provided his salary is an integer less than a million. In our thesis we have tried to limit the number and quality of questions being asked by limiting the communications between parties. Parties only communicate with each other to share the data that is essential for the computation. However certain risks do arise with the computations and are explained in Section 6.7.

4.2 Secure Two-Party Computation

Secure multi-party computation (MPC) is one of the thoroughly researched fields in cryptography and deals with the designs of protocols for computation of an arbitrary function over multiple parties' inputs without revealing the input data to the other parties [9]. MPC protocols are constructed based on a principle that one or more or all the parties involved in the communication are adversaries. Our tasks uses the concepts of *Secure two-party computation (2PC)* which is a sub-field of MPC. So, we have two parties, Alice and Bob, who wish to determine their relationship using their DNA sequences in a malicious two-party setting. Considering the fact that one of them could be an adversary, we have a problem of malicious two-party computation.

4.2.1 Garbled Circuits

Garbled circuit (GC) is a protocol in cryptography that entails two-party communication in which two suspicious parties mutually evaluate a single function over the private inputs of both parties without an involvement of a third party. The function evaluated in the *GC* protocol must be described as a Boolean circuit [24]. Initially, garbled circuit was thought to be impractical but its protocol implementation in Yao's Millionaire Problem [55] was a breakthrough. Since then it has seen improvements in terms of efficiency performance [25, 56, 30] and has found applications in various other problems as well.

4.2.1.1 Oblivious Transfer

Oblivious Transfer (OT) is a cryptographic protocol in which two users can share information with each in a secure way [46]. GC uses OT to share data between users. The oblivious-transfer protocol for sending a string between a sender (*S*) and a receiver (*R*) can be described as:

- *S* has two strings s_0 and s_1 .
- *R* chooses the value of i , such that i is in $\{0, 1\}$.
- *S* sends the string s_i to *R*.
- The OT protocol should satisfy the following criterion:
 - *R* should not have any knowledge of $s_{(1-i)}$.

- S should not have any knowledge of i .

4.2.1.2 Garbled-Circuit Protocol

In the garbled-circuit protocol, the function to be evaluated is a Boolean circuit and it is known to both the parties. If the function to be evaluated is to find the greater of two integer values, then the Boolean circuit is a digital comparator circuit, as it is the case in Yao’s millionaire problem [55]. In our thesis the Boolean circuit described is a DNA-matching circuit which is described in Section 7.4. The garbled-circuit protocol for a secure 2PC between two parties, Alice and Bob, is described in Protocol 2:

Protocol 2: Garbled-circuit protocol

1. Both parties Alice and Bob know the Boolean circuit .
 2. Alice garbles (encrypts) the circuit.
Alice is the garbler in this communication.
 3. Alice sends the garbled circuit and her garbled inputs to Bob.
 4. Bob needs to garble his own inputs with the help of Alice. Both Alice and Bob get into an OT protocol to garble Bob’s inputs.
In the OT protocol Bob is the receiver and Alice is the sender.
 5. Bob evaluates (decrypts) the Boolean circuit using his and Alice’s garbled inputs. Bob is the evaluator here.
 6. Bob communicates with Alice to share the result of the evaluation, as the encrypted result can be decrypted only by Alice.
-

4.2.1.3 Garbling

Garbling is the process in which the Boolean circuit (function to be evaluated as part of garbled circuit protocol) truth table is encrypted [31]. To understand how garbling works, consider an example where Alice and Bob want to verify if both of them want to go to a movie together. This is a sensitive conversation between them as they do not want the other party to know how they feel, unless the other party approves of watching the movie together. The garbled-circuit-protocol representation for this computation is outlined below:

- For Alice, $a = 1$ if she wants to watch the movie together, else $a = 0$.
- For Bob, $b = 1$ if he wants to watch the movie together, else $b = 0$.
- The Boolean circuit to be evaluated is a single AND gate, i.e. $(a \ \& \ b)$.
- Alice is the garbler.
- Bob is the evaluator.

4.2.1.4 Garbled Outputs

Alice picks four random labels: G_A^0 , G_A^1 , G_B^0 and G_B^1 , where each of them correspond to $a = 0$, $a = 1$, $b = 0$ and $b = 1$, respectively. She also picks the 2 random labels for the output wire, and calls these labels O^0 and O^1 . Alice then generates a truth table for their computation as shown in Table 4.1.

Table 4.1: Garbled circuit: AND gate truth table

Alice (a)	Bob (b)	Output ($a \ \& \ b$)
0	0	0
0	1	0
1	0	0
1	1	1

Alice then encrypts or garbles the outputs of the computation using an encryption function Enc to generate a symmetric encryption key. This encryption key is used to encrypt the output function that is performed, in this case ($a \ \& \ b$). The order of the rows of the garbled truth tables is randomly permuted so that the output value cannot be inferred from the order of rows in the final garbled table, as shown in Figure 4.1. Each output represents a garbled row that is independent of the other rows. The only way to decrypt O^0 is if a user knows G_A^a and G_B^b . Any attempt to decrypt the other three rows in the table using G_A^a and G_B^b will lead to detectable failure.

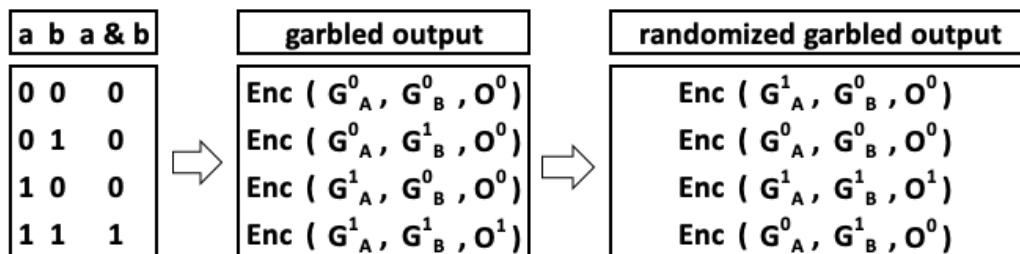


Figure 4.1: Garbled outputs

4.2.1.5 Evaluation

As soon as Bob receives the garbled gate, he needs to decrypt only one garbled row with the actual values of a and b encrypted with $\text{Enc}(G_A^a, G_B^b, O^{\text{out}})$. Alice can easily send G_A^a as she has the labels that are random and independent of each other. For G_B^b , Bob needs to communicate with Alice as she is the garbler. Alice can not send both of Bob's inputs G_B^0 and G_B^1 as Bob can decrypt two rows with these inputs. Moreover, Bob cannot explicitly ask for a particular input from Alice because Alice will learn Bob's inputs. So in this scenario, Alice and Bob will use oblivious transfer such that Bob learns G_B^b without revealing b to Alice. With G_A^a and G_B^b at Bob's disposal, he can evaluate the output wire with the known Boolean circuit (a single AND gate in this case) and share the O^{out} result with Alice.

In a multi-gate circuit, Bob would take the output of the first gate and use it to help process the next gate(s), rather than directly sharing the result of the first gate with Alice. Also, the entire garbling process for each gate (in a multi-gate circuit) is repeated by both Alice and Bob. In the GC protocol, Bob would evaluate the entire Boolean circuit and then share the output to Alice.

The software implementation used in our thesis for the garbled-circuit approach uses the garbled-circuit library created by Weng et al. [53], to execute the garbled-circuit protocol. The GCA software implementation is explained in Chapter 7 and the GC library is explained in the next section.

4.2.2 Authenticated Garbled Circuit

Authenticated Garbling protocol (AGP) for malicious two-party computation, as proposed by Wang et al. [53], is a framework which implements the garbled-circuit protocol. This protocol is an improvement of the garbled-circuit protocol as the garbled circuit is *authenticated*, which implies that the adversary cannot arbitrarily alter the garbled circuit. AGP has also improved the oblivious-transfer protocol as proposed by Nielsen et al [41] and used it in this framework. In the malicious case, this garbled circuit can be transmitted and evaluated in just one round (in cryptography, round complexity is defined as the number of communication that occurs between the parties). The reduction of round complexity to just one round increases the efficiency of proposed framework. In a state-of-the-art semi-honest (presence of an adversary) setting, 20 million AND gates within a Boolean circuit can be garbled in one second [53].

Originally, XOR, AND and OR gates had similar computation and communication costs for creating, transferring, and evaluating a garbled circuit [32]. However, the XOR-gate implementation of Kolesnikov [29] is free of these costs. In a one-round protocol, XOR gates are evaluated for free [32] and building circuits largely of XOR gates can improve the computation and communication complexity.

4.3 Secure Hashing Algorithm

For matching DNA sequences in a secure environment without leaking out input data, we can use cryptographic hash function from the *SHA* family [15]. One major advantage of the secure hashes is the limitation of reverse engineering. Once data is securely hashed, it is practically impossible to recover the data from its hash value. Secure hashing can also be used for DNA matching.

A cryptographic hash function should possess the following properties [15]:

- It should be deterministic; it means that same string will always yield same hash values.
- As hash functions use an algorithm and no public/private keys to generate the hash values, it is almost impossible to reverse the encryption. This is known as “one-way” hash function.
- If you know the hash value h , it is difficult to find a string s , where $h = hash(s)$. This property of hash functions is known as “Pre-image resistance”.
- A small change in a string will generate a new hash value which is entirely different from the original one; it should be totally unrelated to the original one.
- A hash function should be “Collision resistant”; it means that it is hard to discover two different inputs S_1 and S_2 that have the same

hash value, i.e. $hash(S_1) = hash(S_2)$ and $S_1 \neq S_2$.

- A hash function is “Weak collision resistant” if for a string s_1 , it is difficult to find another string s_2 , such that $hash(s_1) = hash(s_2)$. This property of hash functions is known as “Second pre-image resistance”.

Figure 4.2 demonstrates some of the hash function properties.

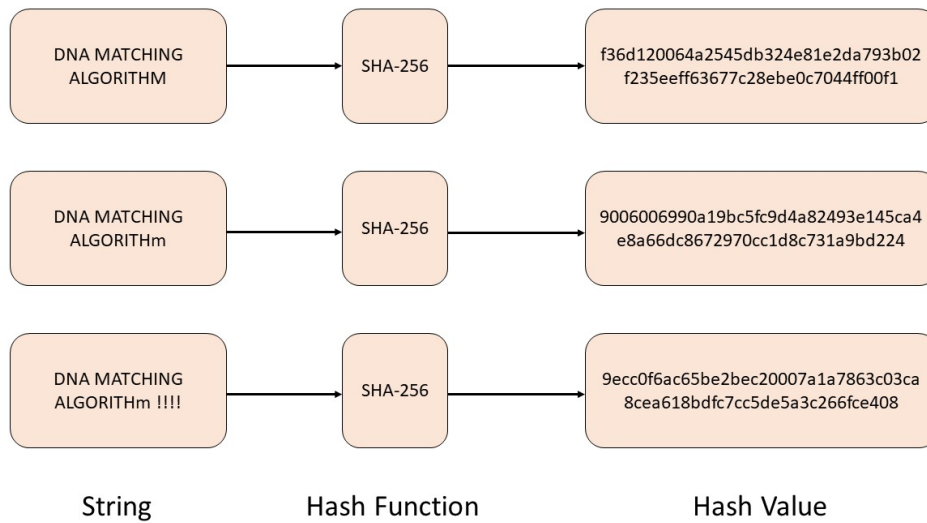


Figure 4.2: SHA-256 hash value generator

4.3.1 SHA-256

SHA-256 is a cryptographic hash function which generates a fixed size output of 256 bits or 32 bytes. SHA-256 was published in 2001 by the National Institute of Standards and Technology (NIST) [15].

Since SHA-256 is deterministic, we can make out that the hash value for a given string s will yield same hash value at all times and it is difficult to create a string that has the same hash value. For a brute-force attack, an attacker will try all strings, if they know the string is small. Therefore, when using hash functions, we must guarantee that our strings are long enough. So, in our case of DNA matching of two individuals we can use SHA-256 of hash functions to compare if both the parties match at the genome locations, as explained in Chapter 6.

SHA-256 is known to withstand all known types of cryptanalytic attack. So, SHA-256 satisfies the following properties:

- Pre-image resistance.
- Second pre-image resistance.
- Collision resistance.

A cryptographic hash function deprived of these three properties is vulnerable and not recommended in practical applications [3]. In our thesis the pre-image resistance and collision resistance properties are of utmost importance. We do not want the adversary to find the string from a hash value or generate the same hash value as the innocent user.

Chapter 5

Data Handling

This chapter explains how data is communicated between the two users of our software. The two users are individuals who are trying to verify if they are related to each other or not. Both our approaches require some amount data sharing between users. So, before diving into the actual data privacy technique implementation we need to establish how this data is parsed and communicated between the users.

5.1 Data Communication

We have implemented two different approaches in our thesis to verify genealogical relatedness between two users. Both our approaches use the same techniques to share and communicate data between them. We have used socket programming from the `java.net` library to implement the communi-

cation protocol.

5.1.1 Server-Client Architecture

We have used the server-client architecture principles from socket programming. In our technique, one user acts as a server and the other user acts as a client. As a prerequisite the server must share the IP address and TCP port to the client before they start their communication. The IP address and TCP port are required by the client to connect to the server. Figure 5.1 depicts how a server establishes connection with a client and communicates.

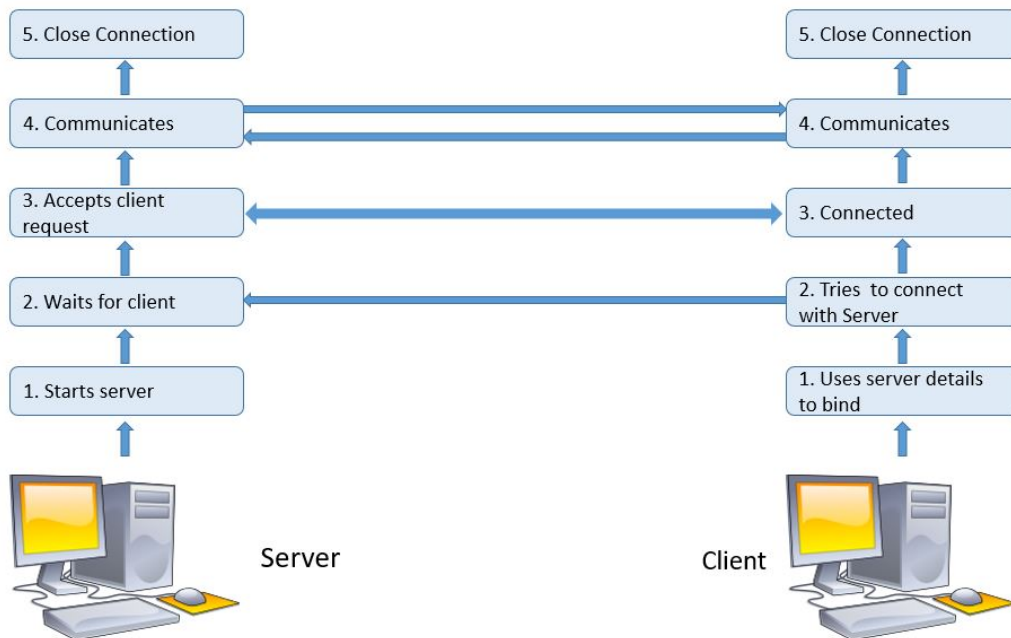


Figure 5.1: Server-client architecture

5.2 Data Parsing

Both our approaches use the same technique to parse the input data from genotyped data files into appropriate fields. We have created a custom class to store the related fields. Details of each field have been described in Table 5.1.

Table 5.1: Input fields

field_name	data_type	description
location	int	stores the location
rsid	String	stores the rsid
gene1	char	stores the first allele
gene2	char	stores the second allele

5.3 Finding and Sharing Common SNPs

Once both the users have parsed their data, they should find common SNPs between them before starting the DNA matching algorithm. Figure 5.2 demonstrates how server and client interact with each to find common SNPs for each chromosome. We have kept only 5000 SNPs concatenated in a string as there is a limit on the number of bytes that can be sent over a network by Java. For each chromosome, the server and client must follow each step outlined in Figure 5.2.

The algorithm to find the common locations executed by server and client is described in Algorithm 3. The locations reported in a genotyped data file is

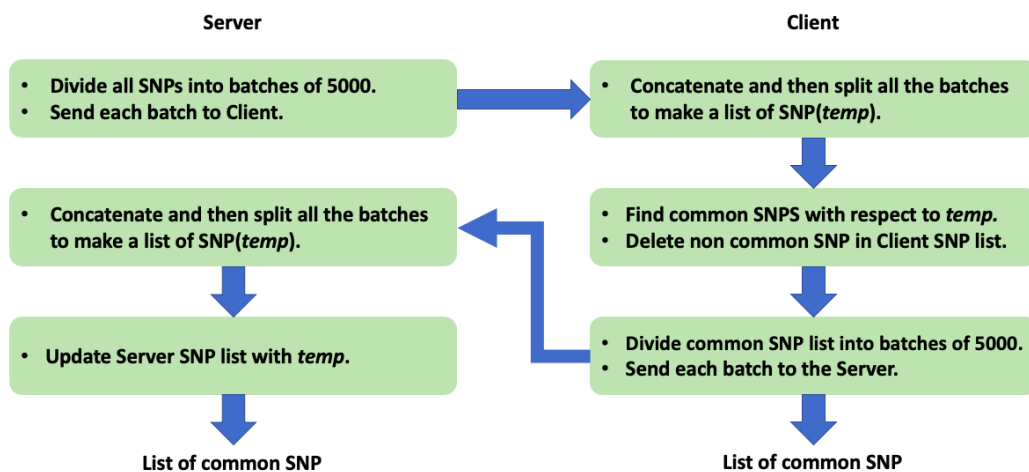


Figure 5.2: Finding common SNPs for each chromosome

sorted in ascending order for each chromosome. So the algorithm assumes the lists are in sorted order and computes common locations accordingly.

5.4 Summary

In this chapter we have established the following aspects of our software implementation:

- Connecting users by a network.
- Parsing data by both users.
- Sharing SNP between users.
- Finding common SNPs between users.

The next two chapters deals with theoretical and software implementation of our two proposed approaches.

Algorithm 3: Finding common SNPs

Data: *loc*: list containing locations of client
rsid: list containing rsid of client
genes: map containing locations and rsid of server
Result: *genes*: map containing only the common SNPs

```
1 i ← 0
2 j ← 0
3 while i < loc.size() & i < rsid.size() & j < genes.size() do
4   if rsid.get(i) = genes.getRSID(j) then
5     if loc.get(i) = genes.getLocation(i) then
6       | i ← i + 1
7       | j ← j + 1
8     end
9     else
10    | genes.setLocation(j) ← loc.get(i)
11    | i ← i + 1
12    | j ← j + 1
13    end
14  end
15  if rsid.get(i) ≠ genes.getRSID(j) & i < loc.size() & i <
    rsid.size() & j < genes.size() then
16    if loc.get(i) < genes.getLocation(j) then
17    | loc.remove(i)
18    | rsid.remove(i)
19    end
20    else if loc.get(i) > genes.getLocation(j) then
21    | genes.remove(j)
22    end
23    else
24    | genes.remove(j)
25    | loc.remove(i)
26    | rsid.remove(i)
27    end
28  end
29 end
30 if loc.size() < genes.size() then
31 | genes.resize(loc.size())
32 end
```

Chapter 6

Hashing-Based Approach

This chapter explores the software implementation of the hashing-based approach. It also delves into the privacy and security risks associated with this approach.

6.1 Main Steps of HBA

From the software design point of view, the Hashing-Based Approach (HBA) executes the following steps sequentially to match DNA of two individuals:

- **Creating network connection between parties**, explained in Section 5.1.
- **Generating nonce**, explained in Section 6.2
- **Reading and parsing input file**, explained in Section 5.2

- **Data preprocessing**, explained in Section 3.4
- **Finding common SNPs**, explained in Section 5.3
- **Dividing frames**, explained in Chapter 3 and Section 6.3.1
- **Generating hashes**, explained in Section 6.4
- **DNA matching**, explained in Section 6.5
- **Reporting results**, explained in Section 6.6

6.2 Generating the Nonce

The first step in HBA is to establish the network connection between the users. The process of creating the network connection is explained in Section 5.1. Once the connection is established, the next step is to generate the nonce.

A nonce is an arbitrary (or a pseudo-random number) number that is used just once in a cryptographic setting. In HBA, we generate a random number that is seeded into a hash function to prevent replay attacks (explained in Section 6.7.1). The hash function is then used to generate the hash value of each frame, will be explained in Section 6.4.

Both parties generate a secure random number using the `SecureRandom` class in Java. We have used `SHA1PRNG` algorithm with `SUN` provider as a pseudo random number generator algorithm to generate the nonce.

The protocol to generate the nonce is described in Protocol 4, as we want to prevent either party from forcing the other to use a nonce that is not random. In this protocol, both parties generate a secure random number and then generate a hash of the secure random number. Both parties share the hash value with each other before sharing the actual secure random number. It is done to ascertain the honesty of each party. Once the honesty of the parties are established, the nonce is generated by using a bit-wise XOR function on these secure random numbers. The collision-resistance property of SHA ensures that a malicious P_2 has no realistic possibility of sending anything other than R_2 to P_1 after they have learned R_1 .

6.3 Frame Division

After generating the nonce, the next steps involve reading the data file, data preprocessing and finding common SNPs. These steps have been explained in Sections 5.2, 3.4 and 5.3 respectively. Once we have the sets of common SNPs, the next step is to divide the chromosomes into frames. The process of dividing frames is explained in Sections 3.1 and 3.1.1.

If we look at Figure 2.4, we can see that we only need to match the homozygous SNP of both parties. The heterozygous SNP match in every case since the SNPs are biallelic. So matching of SNPs and finding IBD segments will be reduced to just matching the homozygous SNPs. While dividing the frames we will generate the SNP string with homozygous SNPs only. This is

Protocol 4: Nonce generation

Data: Random number R_1 from Party P_1 .
Random number R_2 from Party P_2 .

Result: Nonce n as $R_1 \oplus R_2$

1. Hashing Secure Random Number

- (i) P_1 generates a secure random number R_1 .
- (ii) P_1 generates hash $h_1 = H(R_1)$
- (iii) P_2 generates a secure random number R_2 .
- (iv) P_2 generates hash $h_2 = H(R_2)$

2. Share h_1 and h_2

- (i) P_1 sends h_1 to P_2 .
- (ii) P_2 sends h_2 to P_1 .

3. Share R_1 and R_2

- (i) P_1 sends R_1 to P_2 .
- (ii) P_2 sends R_2 to P_1 .

4. Verification of the Random Number

- (i) P_1 verifies if $H(R_2) = h_2$.
- (ii) P_2 verifies if $H(R_1) = h_1$.

5. Nonce Creation

- If verification by P_1 and P_2 is successful
 $n = R_1 \oplus R_2$
- If verification by P_1 or P_2 is unsuccessful
abort

in accordance with the opposite homozygote technique as explained in Section 2.4.1, where the objective is to find homozygotes which are opposite to

each other.

While implementing the HBA, this step is added to the data-parsing step. During the data-parsing step, we apply a filter while reading the data and accept only homozygous SNPs. So, in HBA the SNP string gathered for each frame will consist of only homozygotes. This may result in frames that have very few SNPs. If the size of the frame is too low, the generated hash-codes (explained in Section 6.4) of the frames can be evaluated using a brute-force technique. So, to avoid this we have security parameter `threshold = 100`, that excludes any frames that have fewer than 100 SNPs or homozygotes.

6.3.1 Handling Genotyping Errors

As explained in Section 3.3, genotyping errors can lead to false negatives. So the objective of handling genotyping errors is to reduce the occurrences of false negatives. We know genotyping errors with a frequency of about 0.1% exist in the genotyped data. Genotyping error value is denoted as T_2 . Only one genotyping error is currently handled in HBA. So the value of T_2 is always 1 irrespective of the number of SNPs in a frame. The major drawback of this T_2 setting is that we can get false negatives if a frame contains many shared SNPs.

During frame division we generate two strings of genotypes instead of one. Figure 6.1 shows the generation of two strings in a frame. Alice has a frame with genotypes ‘AA-CC-AA-CC-AA-CC-AA-CC-AA’, while Bob has a frame with genotypes ‘AA-GG-AA-CC-AA-CC-AA-CC-AA’. Both Alice and Bob

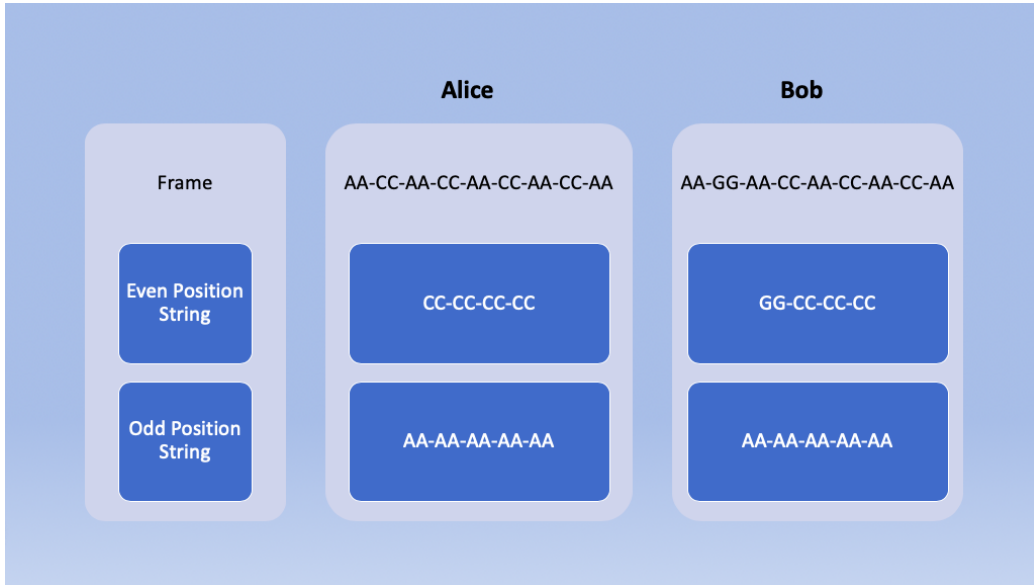


Figure 6.1: Dividing frames into two strings

then divide their frame into two string of genotypes based on the even and odd position of genotypes in the frame.

This division of the frame into two strings enable us to handle one genotyping error. There exists only one mismatch in Figure 6.1 (at the second position). Alice has genotype ‘CC’ whereas Bob has ‘GG’ at the second position of the frame. This entails that the even-position string between Alice and Bob is a mismatch and the odd-position string is a match. With this logic we can conclude that, if there is one mismatch in a frame, then it will either yield a mismatch at the odd-position string or at the even-position string, but not both. We attribute this one mismatch in a frame to a genotyping error that caused this mismatch.

We assume if a segment (frame) is not an IBD segment, then there should

be many mismatches — the presence of many opposite homozygotes. So the probability of having mismatches only at even positions or only at odd positions is very low. If we have multiple mismatches in a frame then they are very likely to occur randomly at both even and odd positions. Hence, this logic to create two strings in a frame looks feasible.

6.4 Hashing

Sharing the string of genotypes in its raw form is a breach of data privacy for both parties. To ensure data privacy of the parties in HBA, the string of genotypes generated for each frame will be hashed before sending it to the other party. In HBA we use the `MessageDigest` class of Java which provides secure one-way hash functions that take arbitrary-sized data and output a fixed-length hash value. In our implementation we have used the ‘SHA-256’ algorithm to generate hash values. The code snippet to generate the hashed values of each string of genotypes is given below.

```
public String getSHAWithNonce(String input, long nonce ){
    MessageDigest md = MessageDigest.getInstance("SHA-256");
    md.update(String.valueOf(nonce).getBytes());
    byte[] messageDigest = md.digest(input.getBytes());
    BigInteger no = new BigInteger(1, messageDigest);
    String hashtext = no.toString(16);
```

```
while (hashtext.length() < 32) {  
    hashtext = "0" + hashtext;  
}  
return hashtext;  
}
```

- The `MessageDigest` class creates an instance with the ‘SHA-256’ algorithm.
- The instance is seeded with the nonce.
- It is then converted to a 32-character hexadecimal string.

To tackle replay attacks, as explained in Section 6.7.1, both parties should calculate the checksum of the generated hashes. A cryptographic checksum is an algorithm that ensures data integrity when a block of data is transmitted over a medium or shared between users. A checksum, when applied on a block of data, generates a mathematical value. So, before sending the data, both parties must generate a checksum (C_1) on that data and share the checksum of the data to each other. After sharing the checksum, the parties should share the actual block of data. Upon receiving the block of data the receiver generates the checksum (C_2) on the data and matches with the received checksum. If the received (C_1) and generated (C_2) checksum match, then the data has not been altered. Otherwise the data may have been maliciously altered.

Protocol 5: Checksum exchange

1. Alice and Bob both calculate the checksum of their respective generated hashes.
 2. Alice divides her checksum into two halves.
 3. Alice sends the first half of the checksum to Bob.
 4. Bob sends his entire calculated checksum to Alice.
 5. Alice sends the second half of her checksum to Bob.
-

In HBA, we generate checksum to verify the data integrity of the hashes that are exchanged between both parties. A checksum also helps both parties to ensure that neither party replays the received hashes to the other party. After the strings have been hashed by respective users, users should calculate the checksum. The protocol for exchanging checksum is given in Protocol 5.

Alice divides the checksum in two halves, as we do not want a scenario where Alice has sent the checksum to Bob and Bob tries to replay the same checksum back to Alice, pretending to be her identical twin.¹ So with this setting, Bob will have only the first half of Alice's checksum and then he will have to send the entire checksum. This condition makes sure that the parties can not replay the checksum of the other party.

After exchanging the checksum, both parties should share the generated hashes with each other. The protocol to share hashes between parties is given in Protocol 6.

¹Identical twins share almost identical DNA, hence with some probability of getting identical genotyping errors, the checksum of hashes could be same.

Protocol 6: Hash exchange

1. Even and odd position strings for each frame are concatenated using a delimiter.
2. A list of concatenated strings for all frames is created.
3. The list is sorted, so that the parties do not get to know which string belongs to which frame.
4. Batches of 100 strings from the list are shared by both the parties to each other, one at a time.
5. Once all the batches are sent, both the parties will have the hashed values of each frame.
6. Both parties will split the received hashed values in batches and add them to a list.
7. Both parties will calculate the checksum of the received hashes and match with the checksum received at the start of the current communication.
8. If the checksum match is true for both parties, then the parties appear to have followed the protocol, else one or both of the parties is malicious.

6.5 DNA Matching

After all the strings generated by frames are hashed and shared between the parties, the DNA-matching algorithm is applied to find IBD segments. The DNA matching is done using the opposite-homozygotes technique from Section 2.4.1.

The frame structure has been designed to represent IBD segments. If a frame contains opposite homozygotes then it not an IBD segment. In Section 6.3.1

it was established that each frame will have two strings of genotypes (an even-position string and an odd-position string, each represented by a SHA-256 hash code). For a frame to be a match and handle one genotyping error for each frame, at least one of the odd- or even-position strings should match. If both the strings of the frame match (share the same hash code) then no opposite homozygotes exist in a frame. If only one string matches then we assume there exists only one opposite homozygote in the unmatched string. This opposite homozygote in the unmatched string is attributed to the one genotyping error and it is ignored. Algorithm 7 describes how the opposite-homozygotes technique is implemented in HBA.

6.6 Reporting Results

In HBA, both parties receive the hashes of frames. Both parties will run their DNA-matching algorithms to find matching segments in each chromosome. Hence, both parties will report the matching results independent of each other.

6.7 Data-Privacy Evaluation

- In Section 6.3 we have established that only homozygous locations are shared between parties for finding IBD segments. Exposing only homozygous locations has certain data-privacy risks. Both parties will be

Algorithm 7: DNA matching algorithm in HBA

Data: hashList: list containing hash values of each frame received from the other party
frameList: Array List of objects containing frame data of the current user

Result: frameList: updated Array List of objects containing frame data of the current user

```
1 matchingCount ← 0
2 foreach hashes  $h_i \in hashList$  do
3   temp ← split( $h_i$ , delimiter)
4   for  $i = 1; i \leq 22; i = i + 1$  do
5     for  $j = 1; j < frameList[i].size(); j = j + 1$  do
6       frame ← fetchObject(frameList[ $i$ ].get( $j$ ))
7       if temp[0] = frame.evenHashValue or
8         temp[1] = frame.oddHashValue then
9         frame.match ← true
10        matchingCount = matchingCount + 1
11      end
12    end
13 end
```

able to learn with 100% certainty that shared SNPs are homozygous.

- If both parties share an IBD segment, that entails both parties are identical in either one of the even or odd positions or both. So, if both even and odd position strings of the frame match for an IBD segment then even an honest relative can figure out the constituents of that IBD segment of the other party.
- Let us suppose that there is a SNP in an IBD segment where ‘AA’ means an individual has disease ‘X’, and ‘TT’ means they do not.

Because two parties have a match with each other on this segment and if both even and odd strings match, then both the parties will know the allele value of that SNP of the other party.

- If the same SNP lies in an unmatched frame, then it will be very difficult to guess the allelic value of the SNP. Since we have a `threshold= 100` security parameter to generate each frame, this setting makes it difficult to guess the constituents of an unmatched frame.
- Let us suppose both parties have a match on a frame from 2 to 6 cM. A malicious relative can try to brute-force match another frame from 3 to 7 cM, as they already match between 3 to 6 cM in the previous frame. So, the malicious relative will only have to predict the homozygotes of 1 cM. Perhaps there are only a few SNPs in the region between 6 cM and 7 cM. If successful, he can similarly extend this attack to subsequent adjacent frames. This is even more practical with the help of information on different allele frequencies and statistical dependencies on SNPs along chromosomes. Although we assume relatives to be not malicious, this attack can be performed if a relative is malicious.
- Let us suppose that there is a particular SNP in an IBD segment where being homozygous means an individual has disease 'X', and being heterozygous means they do not (if that kind of disease exists in the real world), then we have the following scenarios:

- If both the parties share that SNP in HBA, this means that both parties are homozygous at that SNP, and this entails both users have disease ‘X’.
- If one party is homozygous on that SNP and this SNP is not a part of any frame, this means that the other party is heterozygous, which entails the other party does not have disease ‘X’.
- Let us suppose that there is a particular SNP where being homozygous means you could either be healthy and not a carrier or you could be unhealthy. If you know the person is heterozygous, you know they are not sick, but are a carrier of the disease. So, we have the following scenarios:
 - If that particular SNP is shared, that implies both parties are homozygous and neither is a healthy carrier of the disease.
 - If that particular SNP is not shared, then either one party or both parties are heterozygous. If one party is homozygous at that SNP, this implies that the other party is heterozygous and he is a healthy carrier.

6.7.1 Replay Attacks

A replay attack is a form of network attack when an adversary detects a data transmission and tries to either repeat it or delay it. HBA would be prone to these attacks except that we take special steps.

A malicious party will send hashes previously received during an earlier session, in order to make it appear that they match on these frames. To combat replay attacks, we use a nonce to seed the hash function. This way, the hashes generated in one session are unlikely to match any generated in a later session. A malicious party can also perform a replay attack by sending back the received hashes from an innocent party during the same session.

Chapter 7

Garbled-Circuit Approach

This chapter explores the software implementation of the garbled-circuit approach. There are various steps involved in this approach that both parties need to perform in order to determine their relatedness.

7.1 Main Steps of GCA

The GCA implementation is divided into three phases. The first phase involves both parties having to connect with each other and find common SNPs. This phase also requires both parties to generate the data needed for garbled-circuit evaluation. We call this phase the pre-processing phase. The second phase is the garbled-circuit evaluation phase. The third phase is the result-reporting phase. From a software-design point of view, the implementation of GCA can be summarized as :

- **Pre-processing Phase**
 - Creating network connection between parties
 - Reading and parsing input file
 - Data preprocessing
 - Finding common SNPs
 - Dividing frames

- **Garbled-Circuit Evaluation**
 - Reading frame data
 - Generating garbled-circuit binaries
 - Evaluating garbled circuits
 - DNA matching

- **Reporting Results**

7.2 Pre-processing Phase

The first phase of GCA is very similar to the first half of the HBA implementation. We have re-used most of the code of HBA for the pre-processing phase in GCA. The goal of the pre-processing phase is to prepare the data for the next phase, i.e. garbled-circuit evaluation. The garbled-circuit protocol uses a Boolean circuit as a function to evaluate over inputs provided by two

parties, as indicated in Section 4.2.1. So, in the pre-processing phase we get the inputs from the parties and convert into their Boolean equivalents.

The first step in this phase is to create a network connection between users, so that users can find common SNPs between them. This step has been explained in Section 5.1. Both parties read and parse their respective data sets, as shown in Section 5.2. After parsing their data, the users pre-process their data sets to remove all unwanted SNPs, as shown in Section 3.4.

After pre-processing the input data sets, the parties find all the common SNPs between them, as shown in Section 5.3. In GCA we process all the heterozygotes and homozygotes, unlike in HBA, where we only process homozygotes. This results in more common SNPs in GCA than HBA. As we know the inputs will be evaluated over a Boolean circuit, the inputs should be encoded into Boolean values.

7.3 Encoding

The rules and logic for allelic matching between individuals are described in Section 2.4.2 and Figure 2.4. Based on that logic, we will create a Boolean circuit that will serve as the garbled-circuit function to match the alleles for a particular SNP. To achieve this, we will encode the allelic values AA, AC, AG, AT, CC, CG, CT, GG, GT and TT into binary equivalents. The encoded allelic value will be in three-bit binary form, which is shown in Table 7.1. For heterozygotes, we know that the vast majority of SNPs are biallelic, i.e.,

SNPs have a combination of only two alleles. So, a heterozygote matches with everything that is possible, e.g., a heterozygote ‘AC’ with alleles ‘A’ and ‘C’ will match with ‘AA’, ‘CC’ and ‘AC’. This is the reason for assigning the heterozygotes with the same encoding value.

Table 7.1: Encoding of alleles into their three-bit binary equivalents

Allelic Value	Binary Equivalent
AA	100
CC	101
GG	110
TT	111
AC	000
AG	000
AT	000
CG	000
CT	000
GT	000

7.4 Boolean Circuits

Before jumping into how frames are divided, the nature and design of the Boolean circuits should be discussed. Every frame generated from the data set will be evaluated over the DNA-matching Boolean circuit. The over-all frame-matching Boolean circuit is divided into 3 parts;

- SNP-matching Boolean circuit.
- Tree-of-adders circuit.

- Less-than- T_2 circuit.

7.4.1 SNP-Matching Circuit

The SNP-matching Boolean circuit takes as input the three-bit encoded Boolean equivalent allelic values from both parties and outputs 0 if there is match and 1 if there is a mismatch. The goal is to find the number of mismatches in the frame.

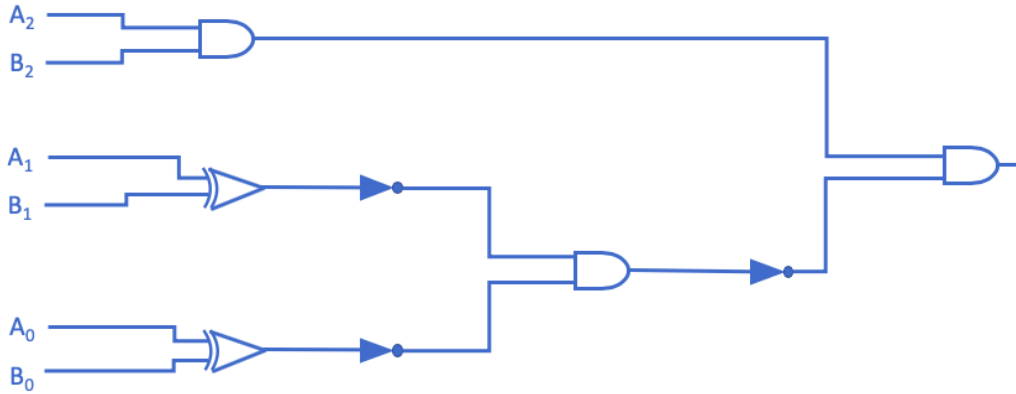


Figure 7.1: SNP-matching circuit

Figure 7.1 shows the SNP-matching circuit for two parties, Alice and Bob, whose inputs are $A_2A_1A_0$ and $B_2B_1B_0$. The most-significant bits are A_2 , B_2 and least-significant bits are A_0 and B_0 . The Boolean expression for the SNP-matching circuit for two inputs is given by

$$(A_2B_2) \& \overline{(A_1 \oplus B_1)} \& \overline{(A_0 \oplus B_0)}.$$

The truth table for the SNP-matching circuit is given in Table 7.2 for all possible SNP combinations. The output of each SNP-matching circuit serves as

Table 7.2: Truth table of SNP-matching circuit

Input of Alice	Input of Bob	Output
100	101	1
100	110	1
100	111	1
100	100	0
100	000	0
101	110	1
101	111	1
101	101	0
101	000	0
110	111	1
110	110	0
110	000	0
111	111	0
111	000	0
000	000	0

the input to the tree-of-adders circuit that counts the number of mismatches. If a frame contains 500 SNPs, then there will be 500 SNP-matching circuits in the frame-matching circuit. Subsequently there will 500 output bits from SNP-matching circuits that the tree-of-adders circuit will add and produce a 9 bit output. With this technique, if we have n frames where each has a different number of SNPs, it will generate n frame-matching circuits. In our setting, the value of n ranges between $3300 < n < 3600$, so generating a large number of circuits during run time will affect the overall running time of GCA.

7.4.2 Frame Division

As explained in the previous section, keeping the actual number of SNPs in a frame will result in generating a large number of circuits at run time, impacting the running time of GCA. To overcome this hurdle, we will fix the number of frame-matching circuits and we will generate the circuits beforehand. The sizes of the frame-matching circuits will be powers of 2, i.e. a circuit will process 2^n SNPs in a frame. This setting will generate few circuits, each of which will be used by multiple frames. Hence we will have a fixed number of circuits that can process frames where the number of SNPs is 128 , 256, 512, . . . , up to 16384.

In GCA, the frames of each party are padded with heterozygotes. Padding is required to make the size of each frame a power of 2. For example, if a frame has 500 SNPs, then it will be padded to the next power of 2, which is 512. While generating the frame for 500 SNPs, we will add the encoded values of all 500 SNPs and then from the 501st to 512th SNP we will pad the frame with heterozygote values, i.e. 000. We pad with heterozygote values because we know the heterozygotes match with each other and hence this padding will not produce any mismatches.

There are two security parameters used in GCA while creating frames, and they are used to avoid brute-force attacks. The first security parameter, `geneCount`, is set to 100; it means that a frame should contain at least 100 SNPs to qualify as a frame. The second security parameter, `threshold`, is set to 40%; it means that the frame should contain at least 40% of total

SNPs in the frame as homozygotes. If there are less than 100 SNPs or if homozygotes are less than 40% of total SNPs in a frame, then the frame will be excluded.

7.4.3 Tree-of-Adders Circuit

The input of the tree-of-adders circuit will be the output of each SNP-matching circuit. So the number of inputs to adder circuits will be a power of 2. The goal of this circuit is to count the 1s among the output bits of the SNP-matching circuit and produce the number of mismatches in a frame.

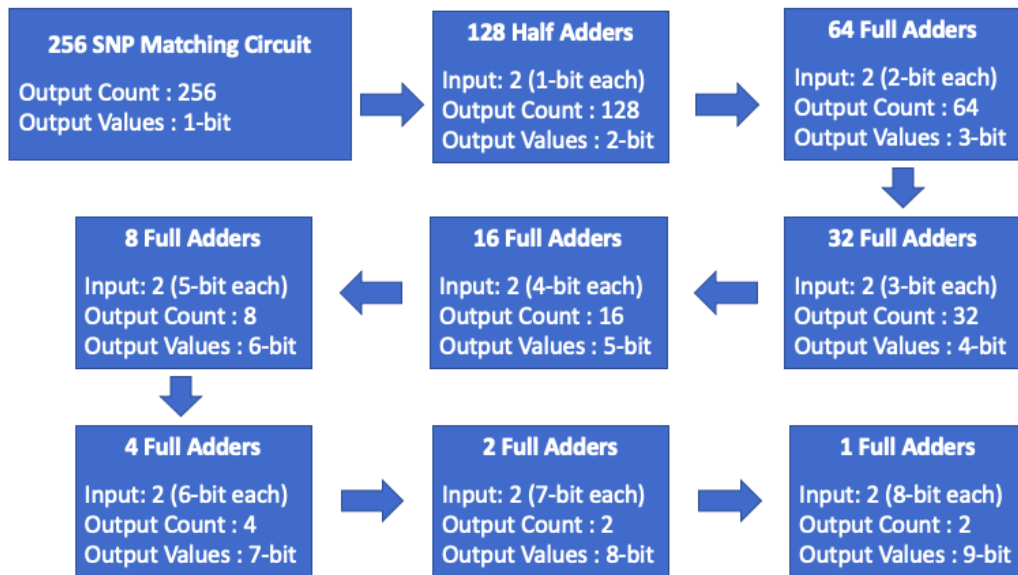


Figure 7.2: Adder tree for a frame with 256 SNPs

We call the circuit that counts the 1s a ‘Tree-of-Adders Circuit’ because it generates a tree-like structure of adder circuits while adding e.g. 512 output bits to produce a 10-bit mismatch value. For a frame with 256 SNPs, we will

have 256 1-bit values to add. We add these 256 1-bit values using 128 half-adder circuits. Each half-adder circuit will input 2 1-bit values and produce a 2-bit output. The 128 half-adder circuits will produce 128 2-bit values. Then we will have 64 2-bit full-adder circuits, each of which will input 2 2-bit values and output 3-bit values. Subsequently with this approach we will finally have a 9-bit output value. Figure 7.2 shows the tree-of-adders circuit structure for 256 SNPs.

Figure 7.3 shows a sample half-adder circuit that inputs two 1-bit values a and b and produces a carry ($a \& b$) and sum ($a \oplus b$). We produce n -bit full-adder circuits by combining n 1-bit full-adder circuits using carry-in wires

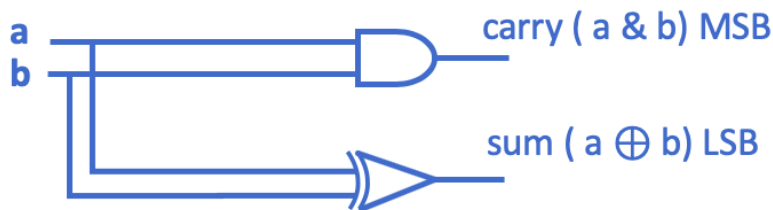


Figure 7.3: Half-adder circuit

Figure 7.4 shows a 3-bit full-adder circuit that adds two 3-bit input numbers and produces a 4-bit output. The circuit adds two 3-bit numbers $A_2A_1A_0$ and $B_2B_1B_0$ and produces a four bit output as $C_3S_2S_1S_0$, where C_3 is the most significant bit and S_0 is the least significant bit. The input wire C_{in} is a wire with signal 0. The Boolean expressions for the circuit is given as

$$C_{in} = 0$$

$$S_0 = A_0 \oplus B_0 \oplus C_{in}$$

$$C_1 = \overline{A_0 B_0} \ \& \ \overline{C_{in}(A_0 \oplus B_0)}$$

$$S_1 = A_1 \oplus B_1 \oplus C_1$$

$$C_2 = \overline{A_1 B_1} \ \& \ \overline{C_1(A_1 \oplus B_1)}$$

$$S_2 = A_2 \oplus B_2 \oplus C_2$$

$$C_3 = \overline{A_2 B_2} \ \& \ \overline{C_2(A_2 \oplus B_2)}$$

7.4.4 Handling Genotyping Errors

As explained in Section 3.3, genotyping errors (denoted as T_2) with a frequency of about 0.1% exist in the genotyped data. The value of T_2 is determined by dividing the total number of SNPs in a frame (T_1) by 1000. To tackle false negatives, in GCA we handle genotyping errors by allowing 1 mismatch in every 1000 SNPs. Genotyping errors are handled by a ‘less-than- T_2 circuit’. This circuit allows mismatches in a frame if they are fewer than T_2 .

7.4.5 Less-Than- T_2 Circuit

The output of the tree-of-adders circuit is an n -bit value. The goal of the ‘less-than- T_2 circuit’ is to compare this n -bit value with T_2 . If the n -bit value is less than T_2 then the output of this circuit will be 1, otherwise 0.

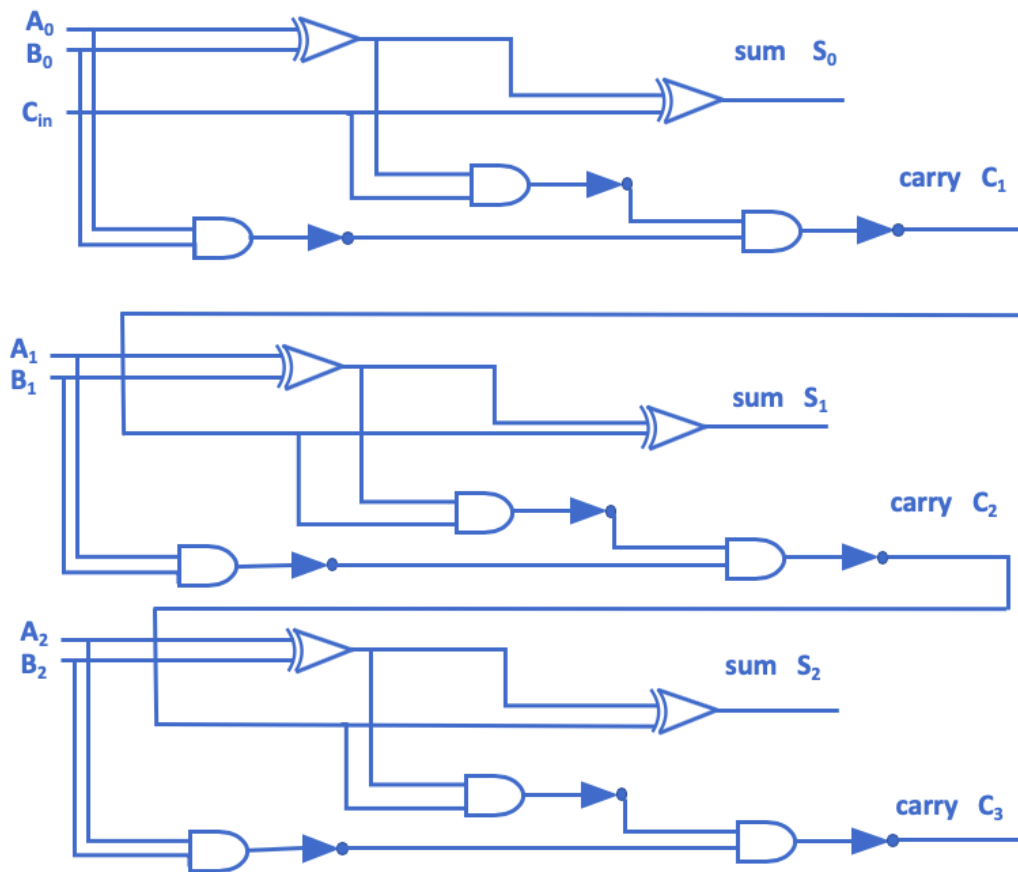


Figure 7.4: 3-bit full-adder circuit

To compare an n -bit value with T_2 , we generate a Boolean expression. Then the less than T_2 circuit is designed according to that expression. The algorithm to generate the Boolean expression is given in Algorithm 8. After the expressions are generated using the algorithm, the whole expression is negated using a NOT gate over the expression. The corresponding expressions for various circuit sizes are given in Table 7.3. The output of the less-than- T_2 circuit is eventually the output of the overall frame-matching

circuit. Output 1 is indicative that the frame is a match, whereas 0 means that the frame is a mismatch.

In the table, the numbers indicate the position of the bit in the n -bit value, where 0 is the least-significant bit. Since our circuit can process only XOR, NOT and AND gates, the OR gates from the expression are transformed using NOT and AND gates.

Algorithm 8: Generate Boolean expression for less-than- T_2 circuit

Data: t_2 : value of T_2

nbits: number of bits (n) in the n -bit value

Result: expr: the Boolean expression for less-than- T_2 circuit

```

1 Function Circuit(thresh, nbits):
2   expr  $\leftarrow$  ""
3   if  $t_2 \leq 0$  or nbits < 1 then
4     | return("TRUE")
5   end
6   for  $i = \text{nbits} - 1; (i \geq 0) \ \& \ ((\text{thresh} \ \& \ (0x1 \ll i)) ==$ 
7     |  $0); i = i - 1$  do
8     | expr  $\leftarrow$  expr +  $i$  + "OR"
9   end
10  if  $i \geq 0$  then
11    |  $s \leftarrow \text{Circuit}(\text{thresh} \oplus (0x1 \ll i), i)$ 
12    | if  $s = \text{"TRUE"}$  then
13      | expr  $\leftarrow$  expr +  $i$ 
14    | else
15      | expr  $\leftarrow$  expr + ("(" +  $i$  + "AND(" +  $s$  + ")")
16    | end
17  return(expr)

```

Table 7.3: Boolean expression for less-than- T_2 circuit

Frame Size	T_2	Expression
128	1	$\overline{6 \vee 5 \vee 4 \vee 3 \vee 2 \vee 1 \vee 0}$
512	1	$\overline{8 \vee 7 \vee 6 \vee 5 \vee 4 \vee 3 \vee 2 \vee 1 \vee 0}$
1024	2	$\overline{9 \vee 8 \vee 7 \vee 6 \vee 5 \vee 4 \vee 3 \vee 2 \vee 1}$
2048	3	$\overline{10 \vee 9 \vee 8 \vee 7 \vee 6 \vee 5 \vee 4 \vee 3 \vee 2 \vee (1 \wedge (0))}$
4096	5	$\overline{11 \vee 10 \vee 9 \vee 8 \vee 7 \vee 6 \vee 5 \vee 4 \vee 3 \vee (2 \wedge (1 \vee 0))}$
8192	9	$\overline{12 \vee 11 \vee 10 \vee 9 \vee 8 \vee 7 \vee 6 \vee 5 \vee 4 \vee (3 \wedge (2 \vee 1 \vee 0))}$
16384	17	$\overline{13 \vee 12 \vee 11 \vee 10 \vee 9 \vee 8 \vee 7 \vee 6 \vee 5 \vee (4 \wedge (3 \vee 2 \vee 1 \vee 0))}$

7.4.6 Circuit Size

The high-level design of the frame-matching circuit is shown in Figure 7.5. The frame-matching circuit is a combination of AND, XOR and NOT gates. As the size of the frame increases, the size of the frame-matching circuit and the number of individual gates also increase. As described in Section 4.2.1, XOR and NOT gates are evaluated for free [32]. So it is imperative to find the size of the circuits and the corresponding number of XOR, AND and NOT gates.

A frame-matching circuit is a combination of few discrete circuits. These circuits include a SNP-matching circuit for each SNP, a combination of half-adder circuits and full-adder circuits, a less-than- T_2 circuit and two gates for

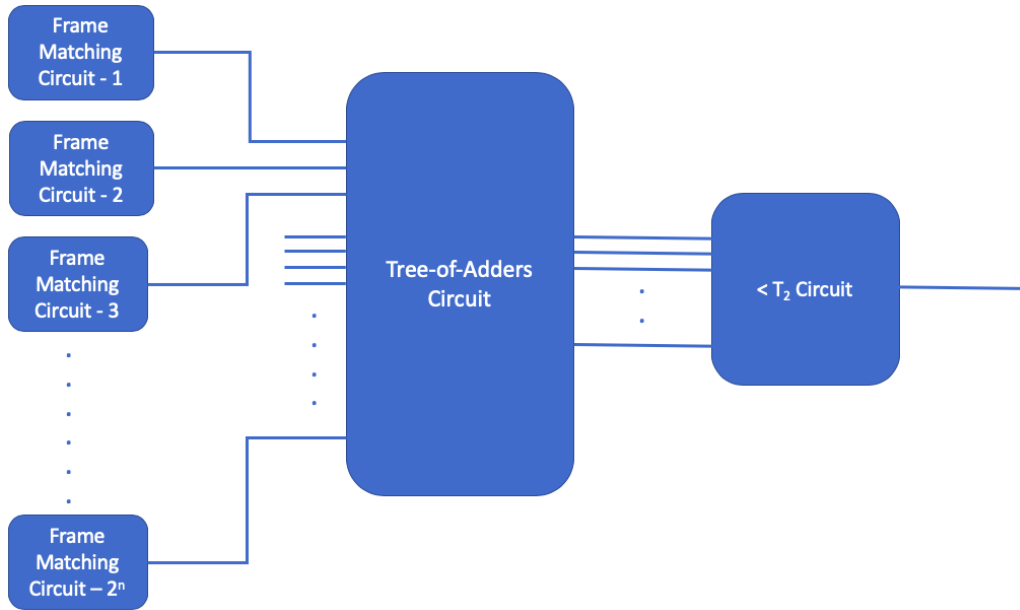


Figure 7.5: High-level circuit depiction of a frame-matching circuit for 2^n SNPs.

producing the Carry_{in} wire. For a frame containing 2^n SNPs, we can find the size of the circuit and number of gates as follows:

SNP-Matching Circuits: A single SNP-matching circuit contains 3 AND gates, 3 NOT gates and 2 XOR gates. A frame containing 2^n SNPs will have 2^n SNP-matching circuits. The counts of each individual gate type used by 2^n SNP-matching circuits are given in Table 7.4.

Half-Adder Circuits: For a frame circuit containing 2^n SNPs, there are exactly $\frac{2^n}{2} = 2^{n-1}$ half-adder circuits. Each half-adder circuit contains one

Table 7.4: Total number of individual gates used by SNP-matching circuits in a frame-matching circuit

Type of gate	Count
AND gates(SNP_{AND})	3×2^n
NOT gates(SNP_{NOT})	3×2^n
XOR gates(SNP_{XOR})	2×2^n
Total count of gates in 2^n SNP-matching circuits (SNP)	$(3 \times 2^n) + (3 \times 2^n) + (2 \times 2^n) = 2^{n+3}$

AND gate and one XOR gate only. The count of each individual gates used by 2^{n-1} half-adder circuits are given in Table 7.5.

Table 7.5: Total number of individual gates used by half-adder circuits in a frame-matching circuit

Type of gate	Count
AND gates(HA_{AND})	$1 \times 2^{n-1}$
XOR gates(HA_{XOR})	$1 \times 2^{n-1}$
Total count of gates in 2^{n-1} half-adder circuit (HA)	$2^{n-1} + 2^{n-1} = 2^n$

Full-Adder Circuits: The tree of full-adders starts with a layer of 2^{n-2} 2-bit adders (each with 2 full-adders). Subsequently, we have 2^{n-3} 3-bit adders (each made of 3 full-adders), until we have a top layer with $2^{n-n} = 1$ n -bit adders.

So, the total number of full-adder circuits in the frame circuit is given by

$$\begin{aligned}
& 2 * 2^{n-2} + 3 * 2^{n-3} + \dots + n * 2^{n-n} \\
&= \sum_{i=0}^{n-2} (n-i)2^i \\
&= n \sum_{i=0}^{n-2} 2^i - \sum_{i=0}^{n-2} i2^i \\
&= n(2^{n-1} - 1) - (2^{n-1}(n-3) + 2) \\
&= 3 \times 2^{n-1} - n - 2 \\
&\sim \frac{3}{2}2^n
\end{aligned}$$

Each individual full-adder circuit contains 3 AND gates, 3 NOT gates and 2 XOR gates. The total count of each individual gates used by $3 \times 2^{n-1} - n - 2$ full-adder circuits is given in Table 7.6

Frame-Matching Circuit: A frame-matching circuit consists of SNP-matching circuits, half-adder circuits and full-adder circuits. So, the total number of gates used by the frame-matching circuit (FMC) is given by the following equation:

$$FMC = SNP + HA + FA + x + y$$

Table 7.6: Total number of individual gates used by full-adder circuits in a frame-matching circuit

Type of gate	Count
AND gates(FA_{AND})	$3 \times (3 \times 2^{n-1} - n - 2)$ $= \frac{9}{2}2^n - 3n - 6$ $\sim \frac{9}{2}2^n$
NOT gates(FA_{NOT})	$3 \times (3 \times 2^{n-1} - n - 2)$ $= \frac{9}{2}2^n - 3n - 6$ $\sim \frac{9}{2}2^n$
XOR gates(FA_{XOR})	$2 \times (3 \times 2^{n-1} - n - 2)$ $= 3 \times 2^n - 2n - 4$ $\sim 3 \times 2^n$
Total number of gates in $3 \times 2^{n-1} - n - 2$ full-adder circuits (FA)	$8 \times (3 \times 2^{n-1} - n - 2)$ $= 12 \times 2^n - 8n - 16$ $\sim 12 \times 2^n$

where $x = 2$ (1 NOT gate and 1 AND gate: the number of gates required to produce the Carry_{in} wire), and $11 \leq y \leq 17$ is the number of gates required to generate the less-than- T_2 circuit for the range of T_2 values needed. Substituting the values of SNP, HA and FA from Tables 7.4, 7.5, 7.6 respectively, we have

FMC

$$\begin{aligned}
&= 2^{n+3} + 2^n + (24 \times 2^{n-1} - 8 \times n - 16) + 2 + y \\
&= 2^n + 12 \times 2^n + 8 \times 2^n - 8n - 14 + y \\
&= 21 \times 2^n - 8n - 14 + y \\
&\sim 21 \times 2^n
\end{aligned}$$

The total number of AND gates used by the frame-matching circuit is the sum of SNP_{AND} , HA_{AND} and FA_{AND} . Substituting the values of SNP_{AND} , HA_{AND} and FA_{AND} from Tables 7.4, 7.5, 7.6 respectively, we have

$$\begin{aligned}
&SNP_{\text{AND}} + HA_{\text{AND}} + FA_{\text{AND}} + 1 + y_1 \\
&= 3 \times 2^n + 2^{n-1} + 3 \times (3 \times 2^{n-1} - n - 2) + 1 + y_1 \\
&= 2^{n+3} - 3 \times n - 5 + y_1 \\
&\sim 8 \times 2^n
\end{aligned}$$

where y_1 is the number of AND gates in the less-than- T_2 circuit.

The total number of free (XOR and NOT) gates used by the frame-matching circuit is the sum of SNP_{NOT} , SNP_{XOR} , HA_{XOR} , FA_{NOT} and FA_{XOR} . Substituting respective values from Tables 7.4, 7.5, 7.6, we have

$$\begin{aligned}
& SNP_{\text{NOT}} + SNP_{\text{XOR}} + HA_{\text{XOR}} + FA_{\text{NOT}} + FA_{\text{XOR}} + 1 + y_2 \\
&= 5 \times 2^n + 2^{n-1} + 5 \times (3 \times 2^{n-1} - n - 2) + 1 + y_2 \\
&= 8 \times 2^n + 5 \times (2^n - n) - 9 + y_2 \\
&\sim 13 \times 2^n
\end{aligned}$$

where y_2 is the number of NOT gates in less-than- T_2 circuit.

A summary of the number of gates used by the frame-matching circuit is shown in Table 7.7. The total number of gates used by a frame-matching circuit that contains 2^n SNPs is approximately 21×2^n . So, for processing each SNP, we process 21 gates and out of those 21, 8 gates are AND gates and 13 are free XOR and NOT gates.

Table 7.7: Total number of individual gates used in a frame-matching circuit containing 2^n SNPs

Gates	Exact Count	Approximate
AND gates	$2^{n+3} - 3n - 5 + y_1$	8×2^n
Free gates	$2^{n+3} + 5(2^n - n) - 9 + y_2$	13×2^n
Total	$2^n + 3 \times 2^{n+2} + 2^{n+3} - 8n - 14 + y$	21×2^n

7.5 Garbled-Circuit Evaluation

We have used the authenticated garbled circuit (AGC) library developed by Wang et al. [53] for garbled-circuit evaluation. For two parties to evaluate the garbled-circuit protocol, this library takes the Boolean circuit and data of both parties as its inputs. It then uses the oblivious-transfer protocol to garble the data between them and evaluate the Boolean circuit to produce an output.

7.5.1 Grouping of Frames

In GCA setting, we use the amortized 2PC execution of AGC library; it means multiple frames of two parties can be evaluated over a single DNA-matching circuit in one execution. In theory, all the frames with the same (padded) size can be evaluated over the same DNA-matching circuit in one execution, i.e. all the frames with 1024 SNPs can be evaluated over the 1024-SNP DNA-matching circuit. However, due to input/output constraints of the system used, all the frames of same size can not be executed in a single execution. To tackle this constraint, we group the frames of similar sizes together. If there are 1036 frames of 2048 SNPs each, then we group the 2048 SNPs frame into a batches of 128. So with this setting we will have eight batches of 128 frames and one batch of 12 frames. So, the 2048-SNP frame will be executed in nine runs. Similarly we will group all same-sized frames for the amortized execution.

7.5.2 Reporting Results

In GCA, the garbled circuit is evaluated by only party, i.e. only one party is the evaluator. This is one of the drawbacks of the garbled-circuit protocol. User fairness can be defined in the context of secure 2PC as a property through which if one party receives a result then it is ensured that the other party also receives it [21]. In the ideal world, it is possible to achieve user fairness when we assume all the parties involved are honest. But, in case of a malicious setting where one or more parties are assumed to be adversaries, it is considered that achieving user fairness is often difficult and increases the evaluation complexity. Gordon et al. [21] considered a family of functions where user fairness is possible, but it also increases the round complexity. The generic software available for secure 2PC does not even try to solve this problem and to provide user fairness.

The AGC library does not provide user fairness. So, to report the results we execute the following steps:

- The evaluator evaluates all the frames over the corresponding DNA matching circuit.
- The evaluator writes the output of each frame in a text file (results file).
- The evaluator shares the results file with the other party by establishing a network connection as shown in Chapter 5.

- Both the parties will now find the results of their DNA test using the results file.

7.6 Data-Privacy Evaluation

- We have the `threshold` security parameter in place, which excludes the frames with less than 40% homozygotes. But a malicious party can tweak our application to remove that security parameter. With this tweak the malicious party can supply a fabricated data consisting of only heterozygotes. This will imply that the matching result will show the parties as identical twins or relatives, matching on every frame. But with this setting, the malicious party would not be able to learn anything about the innocent user's data.
- If the garbler of the garbled-circuit is a malicious party, then they can stop the communication after receiving the output label from evaluator of the DNA-matching circuit and not share the results with the other party.
- Let us suppose a malicious party Bob wants to know if the other party is "TT" at a particular SNP, where the possible alleles for that SNP are 'A' and 'T'. Bob will assign "AA" to that particular SNP in his malicious data. Then he will find the frame in which that SNP lies. If the size of that frame is 11000, then that frame will allow 11 mismatches

in that frame as part of genotyping-error handling. So, in that frame Bob will add 11 random (unlikely) allelic values to 11 SNPs to cause 11 mismatches. For the rest of the SNPs in the frame he will assign heterozygous values. With this setup, if that frame is a mismatch, then Bob will certainly know that the other party has “TT” value in that SNP (as “AA” and “TT” are opposite homozygotes), because for that frame all other SNPs match. If that frame is a match, then the party could be either “AA” or “AT”.

Chapter 8

Experimental Setup and Results

This chapter explores the experiments that were performed and reports the corresponding observations and results.

8.1 Build and Test Setup

All the experiments were performed inside an Oracle VirtualBox [43] with the following configurations:

- CPU – 1.8 GHz Dual-Core Intel Core i5
- Virtual Box Image – Ubuntu 18.04.4 LTS
- Kernel Version – 5.3.0-46-generic

- Memory – 4 GB 1600 MHz DDR3
- Dedicated Processors – 2

The virtual machine is installed on a host machine with the following details:

- CPU – 1.8 GHz Dual-Core Intel Core i5
- OS – macOS Catalina (version 10.15.6)
- Memory – 8 GB 1600 MHz DDR3

8.2 Data Sets

Before jumping into the experiments, it is important to establish the source of the data sets used. In 2012, Glusman et al. [19] studied and examined the genomic data¹ (genotyped at *23andMe.com*, which is self reported and publicly available²) of a Spanish family of five blood relatives to determine the quality of the data and the comparison of the similarity of shared SNPs between the family. The five members include son, daughter, father, mother and aunt (mother’s sister) across two generations, where the father is not related to the mother and aunt.

In our experiments we have used the publicly available genomic data of this family as its quality has been well established in Glusman et al.’s paper [19].

¹23andMe version 2 data genotyped data for Mother, Father, Daughter and Aunt.

²<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3941016/bin/f1000research-1-103-s0001.tgz>

Table 8.1: Data-set descriptions

Data Sets	Testing Agency	Description	Avg. SNPs
4	23andMe.com	Spanish Family	960,000
4	Ancestry.com	Family of 4	650,000
16	23andMe.com	Random 16 unrelated parties	620,000
16	FTDNA	Random 16 unrelated parties	650,000
6	Ancestry.com	Random 6 unrelated parties	550,000

Similarly, we also have the genomic data-set of four family members (father, mother, son and daughter) genotyped at Ancestry.com. Since these two families have been tested for being relatives, they have served as the cases for related scenarios in our experiments. For testing the unrelated individuals or the non-matching scenario, we have taken another publicly available data-source [45] where we found genomic data for various DNA tested individuals. Overall 46 DNA data sets have been used in our experiments. All the data sets have been utilised from publicly available portals [19] [45]. A brief description of the data sets used are described in in Table 8.1. The data source of these 46 data sets are given in Appendix A.

8.3 Test Scenarios

Based on the types of data sets, there are multiple cases to test for two scenarios. The following list shows the different types of test scenarios and the corresponding test cases that are performed for both the approaches.

- a) **Number of common SNPs, frames and centiMorgans shared between two parties**
 - (i) **TASA** : Both parties Tested At Same Agency.
 - (ii) **TADA** : Both parties Tested At Different Agency.

- b) **Relationship between two parties**
 - (i) **RTASA** : Related parties Tested At Same Agency.
 - (ii) **UTASA** : Unrelated parties Tested At Same Agency.
 - (iii) **UTADA** : Unrelated parties Tested At Different Agency.

Due to data-set unavailability, there are no test cases for “Related parties Tested At Different Agency” (RTADA).

8.4 Data Characteristics

A genotyped data set is a collection of SNPs, as discussed in Chapter 2. There are various values of alleles that determine the constituents of an SNP. Figure 8.1 shows the distribution of the various characteristics of the SNPs present in the 46 data sets that have been used in the experiments.

Most of the experimental results are expressed as box-plots in this chapter. A box-plot is a convenient graphical representation that outlines the distribution of a variable. A box-plot shows six characteristics or summaries of a variable, i.e. minimum, first quartile, median, third quartile, maximum and

outliers. The box encompasses 50% of the observations, those between the 25th and 75th percentiles. The line inside the box represents the median. The vertical lines from the top to the bottom of the distribution typically show the maximum and minimum. If some data points are too far away from the median, i.e 1.5 times the inter-quartile range, they are marked as outliers.

Figure 8.1 displays five major components from the data sets. The first box shows the distribution of the total number of SNPs present in the data sets. The rest of the four boxes represent the characteristics distribution of each data set in terms of *homozygotes*, *heterozygotes*, *dashes* and *RSID with 'i'*. *Dashes* are the SNPs where permissible alleles (A,C,G and T) are missing and instead are reported as ‘-’, as discussed in Chapter 3. SNPs that contain dashes are excluded as part of the frame division. Some SNPs in the data set start with ‘i’ (e.g. i287639) instead of ‘rs’ (e.g. rs287639). These types of SNPs are also excluded as part of frame division.

In Figure 8.1, it is visible that there are outliers in the distribution of *Total SNPs* and *Homozygotes*. This is due to the presence of 4 data sets from the Spanish family, where average number of SNPs is much greater than the average number of SNPs in the rest of the data sets, shown in Table 8.1.

8.5 Common SNPs

This section outlines the percentage of common SNPs shared between the test data sets. The test cases are divided into TASA and TADA scenarios

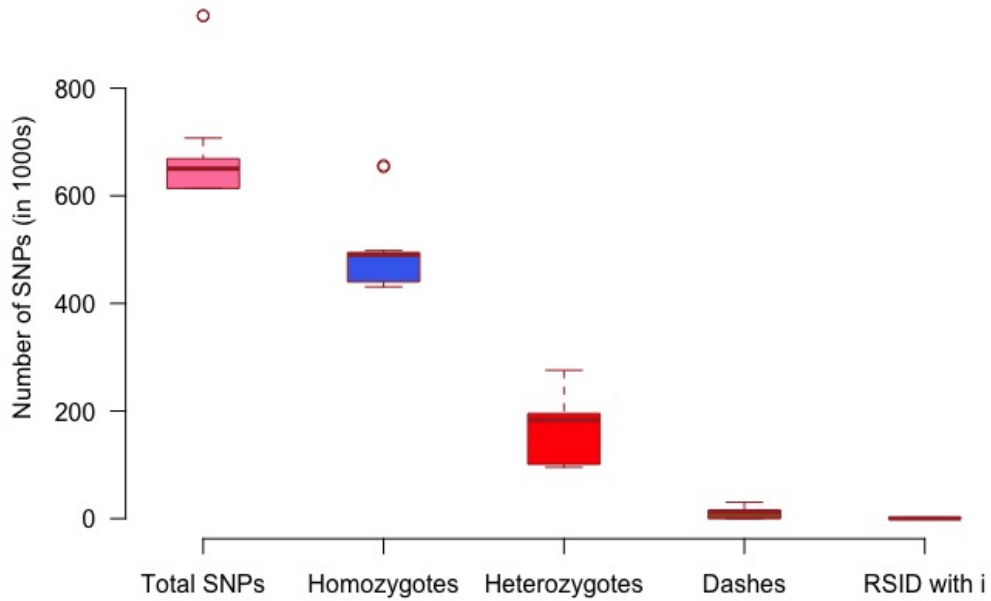


Figure 8.1: Data characteristics of test user’s DNA data

for both approaches. The total number of data points in TASA and TADA are 129 and 272, respectively.

Table 8.2 shows the percentage of common SNPs shared in both approaches for TASA and TADA test cases. It can be concluded from the results that parties who have been reported from the same testing agency tend to share more common SNPs than parties who have been reported from different testing agencies.

Based on the results shown in Table 8.2, it is evident that GCA uses far

Table 8.2: Percentage of shared common SNPs

	GCA	HBA
TASA	70 – 95	37 – 69
TADA	20 – 61	9 – 28

more common SNPS while matching the SNPs than HBA. The reason behind this disparity is that GCA uses both homozygotes and heterozygotes in its DNA matching algorithm, whereas HBA uses only homozygotes in its DNA matching algorithm.

8.6 Frames and centiMorgan Distribution

The experimental setup of both GCA and HBA generates frames within the range of $0 \leq f \leq 3600$, where f is the number of frames. Shared centiMorgan among individuals range between $0 \leq cM \leq 3750$ as established in Section 2.5.

Figure 8.2 shows the distribution of frames and centiMorgans shared between parties being tested in GCA. The shared centiMorgan for both TASA and TADA are greater than 3400 cM and near the vicinity of 3600 cM . Hence, GCA computation generates enough common SNP and shared centiMorgans for the DNA-matching algorithm to produce an acceptable result, as established in Section 2.6.

Figure 8.3 shows the distribution of shared frames and cM for HBA. Based

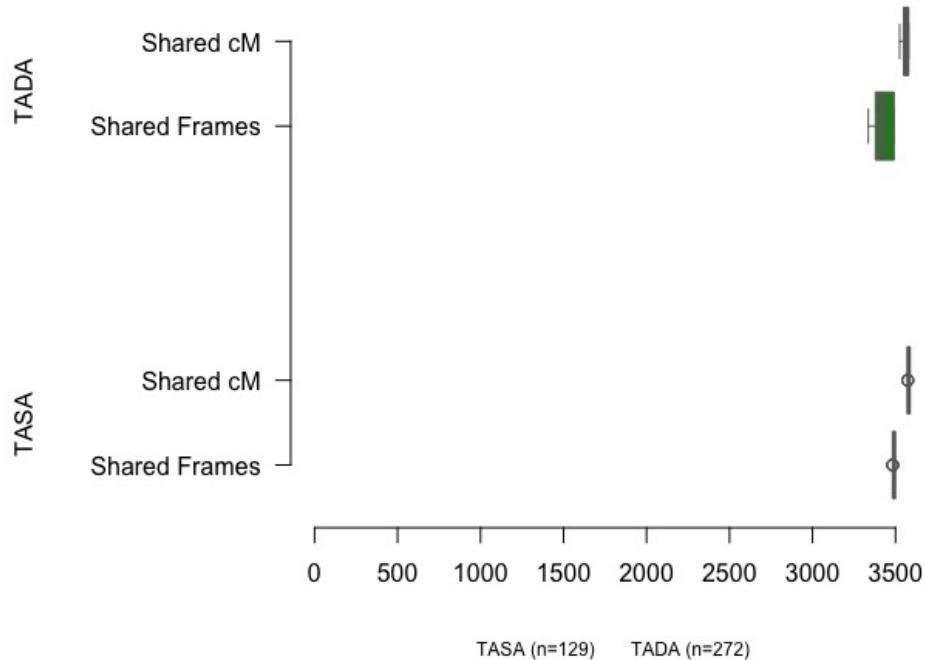


Figure 8.2: Frames and centiMorgan distribution for TASA and TADA test cases for GCA

on the results of our experiments we found that the majority of the cases in *TADA* were reported to share very few frames and cM, and this is because of the security parameter ‘threshold’ that is set to a minimum of 100 homozygotes per frame, as explained in Chapter 6. Since many frames with this setting do not qualify, they are excluded. We experimented with a combination of ‘threshold’ values and different frame sizes to reduce the count of excluded frames. The combination of `threshold = 100` and `cMperFrame = 25 cM` proved to be much better. Figure 8.4 demonstrates how the change in the

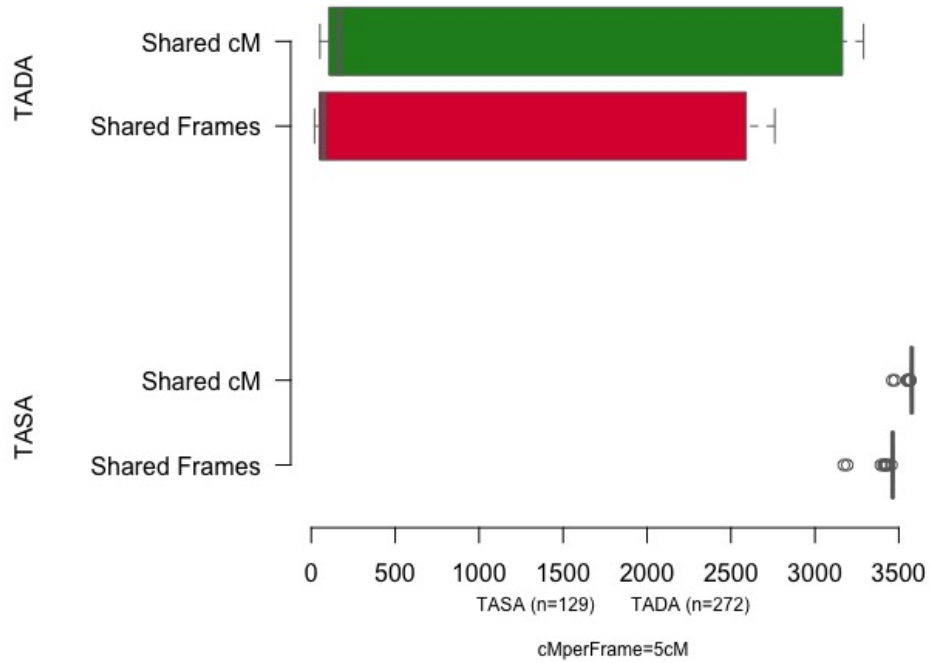


Figure 8.3: Frames and centiMorgan distribution for TASA and TADA test cases for HBA with frame-size = 5 cM

frame size to 25 cM drastically reduced the number of excluded frames in HBA for TADA cases. The setting of `cMperFrame = 25 cM` has its disadvantages, which have been explained in Section 8.8.

Figure 8.5 shows the distribution of frames and cM for HBA (`cMperFrame = 25 cM`). This distribution is ideal as it gives the DNA-matching algorithm in HBA enough frames to process the data and produce better results.

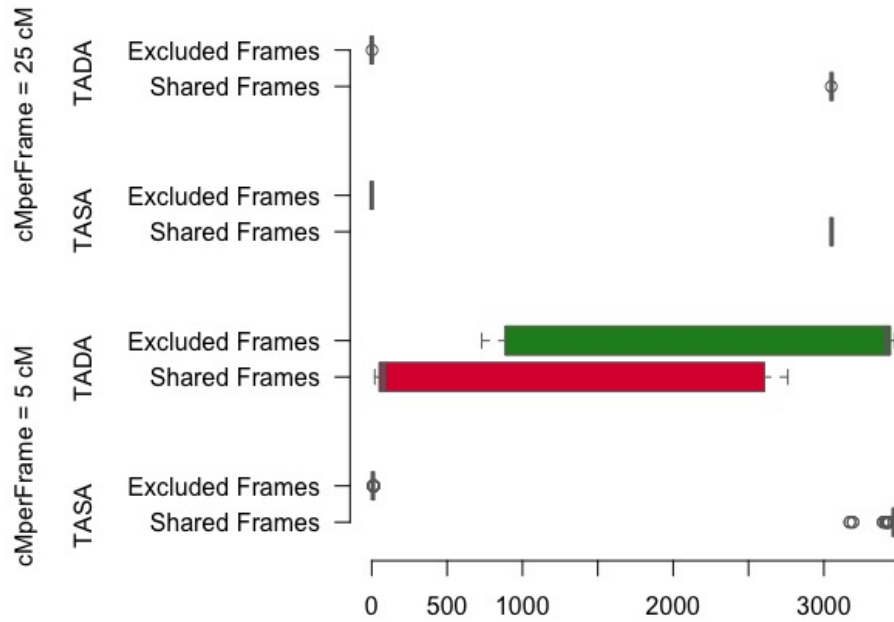


Figure 8.4: Excluded frames distribution of HBA for TSRG and TDRG test cases with cMperframe size of 5 cM and 25 cM

8.7 Relatedness

There are four test cases to validate relationship between two parties. The nature and count of relationships determined in RTASA case are shown in Table 8.3. The total number of cases of UTASA and UTADA are 120 and 272, respectively.

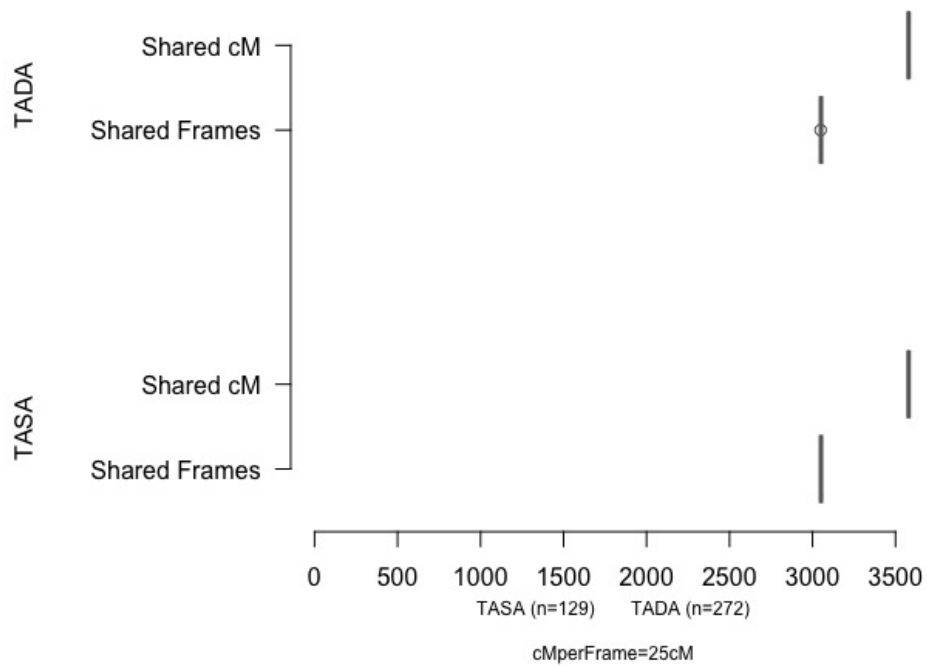


Figure 8.5: Frames and centiMorgan distribution for TASA and TADA test cases for HBA with $cMperFrame = 25 cM$

Table 8.3: Number of related cases in RTASA

Relationship	No. of cases
Parent/Child	6
Sibling	2
Aunt/Niece	1
Total	9

8.7.1 Garbled-Circuit Approach

The results for all the RTASA cases tested in GCA are shown in Table 8.4. The results of GCA do conform with the data in Table 2.1. For all the unrelated cases of UTASA and UTADA ($n = 120 + 272 = 392$), the count of matching centiMorgan lie between 0 and 33. The low value of matching centiMorgan indicates that the individuals involved in DNA matching are not second cousins or closer.

Table 8.4: GCA relatedness results for RTASA cases

Relationship	Family	Shared cM	Matching cM	Segments
Father/Son	Ancestry	3579	3578	23
Father/Daughter	Ancestry	3579	3575	23
Father/Daughter	Spanish	3574	3569	25
Mother/Daughter	Spanish	3574	3569	26
Mother/Daughter	Ancestry	3579	3578	23
Mother/Son	Ancestry	3579	3576	25
Aunt/Niece	Spanish	3574	1516	47
Sibling	Spanish	3574	2483	50
Sibling	Ancestry	3579	2675	44

8.7.2 Hashing-Based Approach

The results for all the RTASA cases tested in HBA (`cMperFrame = 5 cM`) are shown in Table 8.5. These results do not conform with the data in Table 2.1, as the number of IBD segments are more than expected. As explained in Section 8.6, many excluded frames exist in HBA with the `cMperFrame =`

5 cM setting. Due to this, some of the centiMorgan in the chromosome are excluded, leading to discontinuous frames.

For the 5 cM setting, the shared cM distribution for Father/Son in chromosome 1 are $[0 - 35]$ cM, $[37 - 162]$ cM and $[165 - 281]$ cM. All the three segments match. The frames that contains data for centiMorgans 36, 163 and 164 are excluded, as these frames do not contain more than 100 SNPs. Hence there are 3 matching segments reported on chromosome 1. Similar instances are there for other chromosomes as well. For the 25 cM setting, the shared cM distribution for Father/Son in chromosome 1 is $[0-281]$ cM and the entire segment matches. So we get only 1 matching segment there.

Table 8.5: HBA relatedness results for RTASA cases (`cMperFrame` = 5 cM)

Relationship	Family	Shared cM	Matching cM	Segments
Father/Son	Ancestry	3567	3558	29
Father/Daughter	Ancestry	3569	3556	28
Father/Daughter	Spanish	3553	3540	30
Mother/Daughter	Spanish	3549	3263	34
Mother/Daughter	Ancestry	3476	3446	42
Mother/Son	Ancestry	3461	3432	46
Aunt/Niece	Spanish	3548	1426	48
Sibling	Spanish	3562	2392	51
Sibling	Ancestry	3564	2662	48

For all the 392 unrelated cases of UTASA and UTADA , the count of matching centiMorgan lie between 0 and 86. The low value of matching centiMorgan indicates that the individuals involved in DNA matching are not second

cousins or closer, with one ambiguous case of 86 cM.

The results for all the RTASA cases tested in HBA (`cMperFrame` = 25 cM) are shown in Table 8.6. These results somewhat conform with the data in Table 2.1, as the ‘Father/Daughter’ test from the Spanish family matches less than expected centiMorgans between them. It is because HBA with `cMperFrame` = 25 cM is prone to false negatives, as explained in Section 8.8. All the 392 unrelated cases of UTASA and UTADA do not share a single matching centiMorgan between them.

Table 8.6: HBA relatedness results for RTASA cases (`cMperFrame` = 25 cM)

Relationship	Family	Shared cM	Matching cM	Segments
Father/Son	Ancestry	3579	3578	23
Father/Daughter	Ancestry	3579	3574	24
Father/Daughter	Spanish	3579	3182	30
Mother/Daughter	Spanish	3579	3511	29
Mother/Daughter	Ancestry	3579	3561	23
Mother/Son	Ancestry	3579	3535	23
Aunt/Niece	Spanish	3579	1195	26
Sibling	Spanish	3579	2138	36
Sibling	Ancestry	3579	2511	32

8.8 False Negatives in HBA

HBA with `cMperFrame` = 5 cM setting is susceptible to exclude more frames than GCA. HBA only uses homozygotes for frame division, such that the security parameter `threshold`= 100 only allows frames with a minimum of 100

homozygotes. So to prevent the exclusion of frames, the size of `cMperFrame` value is increased to 25, which entails that the number of SNPs in a frame is increased significantly.

As the number of SNPs in a frame increases when we set `cMperFrame = 25 cM`, the number of frames containing more than 1000 SNPs is also considerably increased. This increase in the number of SNPs in a frame results in poor genotyping handling by HBA, as HBA can handle only 1 genotyping error in a frame. As a result, some of the frames which are actually a match will result in a mismatch. So, these mismatches make HBA with the `cMperFrame = 25 cM` setting prone to false negatives.

8.9 Running Times

We tested 11 runs of 250 executions each for both HBA and GCA. Each run of 250 executions contains cases from RTASA, UTASA and UTADA.

8.9.1 Hashing-Based Approach

The running times result for HBA from the 11 runs is plotted in Figure 8.6. The median of each run for HBA lies around 7.5 seconds. The best running time for HBA was just over 6 seconds and the worst was just below 9 seconds.

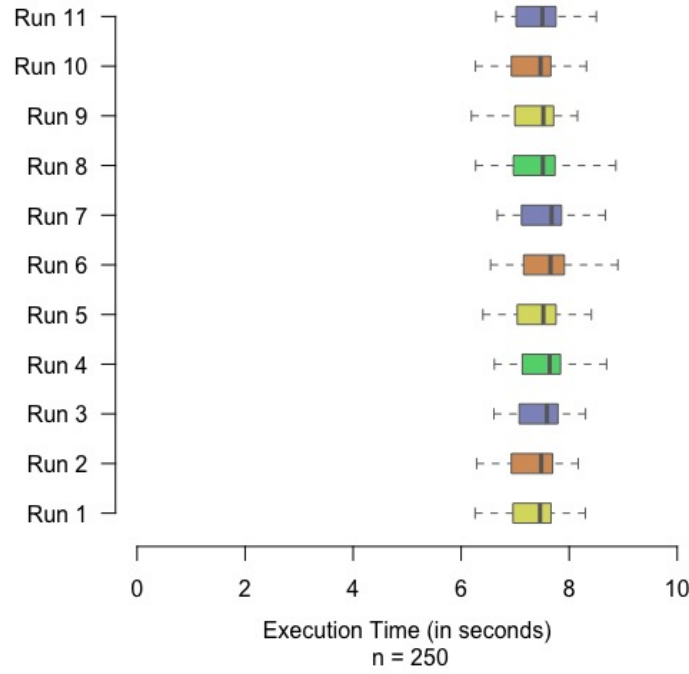


Figure 8.6: Running times for HBA

8.9.2 Garbled-Circuit Approach

GCA takes a much longer time to execute than HBA. In GCA, the running time of our software depends on the number of SNPs processed. Running times of some of the cases with different number of common SNPs is shown in Table 8.7.

Hence, in GCA the running times is represented as the ‘number of SNP processed per second’. The running times result for GCA from the 11 runs is plotted in Figure 8.7. The median of these runs lie somewhat around 3.2 thousand SNP processed per second. The best execution is around 3.35

Table 8.7: GCA running times for different cases

Case	Number of SNPs	Running Time (in seconds)	SNPs Processed per second
RTASA			
Spanish Family	1,316,864	400.424	3288.67
Ancestry.com Family	873,728	266.479	3278.79
UTASA			
23andme.com	856,320	263.986	3243.81
UTADA			
FTDNA and Ancestry	592,640	181.251	3269.72
FTDNA and 23andme	212,608	65.372	3252.28

thousand SNP processed per second, whereas the worst execution is just above 3 thousand SNP processed per second.

8.10 Running Times Over Network

We executed 100 tests each for HBA and GCA on two machines connected in a LAN setting, with the link between them having 304.2 Mbps bandwidth. The configuration of both the machines used are identical and are given in Section 8.1. The only difference between the two machines is the manufacturing year. The comparison of running times between execution on the same machine and two machines connected in a LAN showed negligible difference ($< 1\%$).

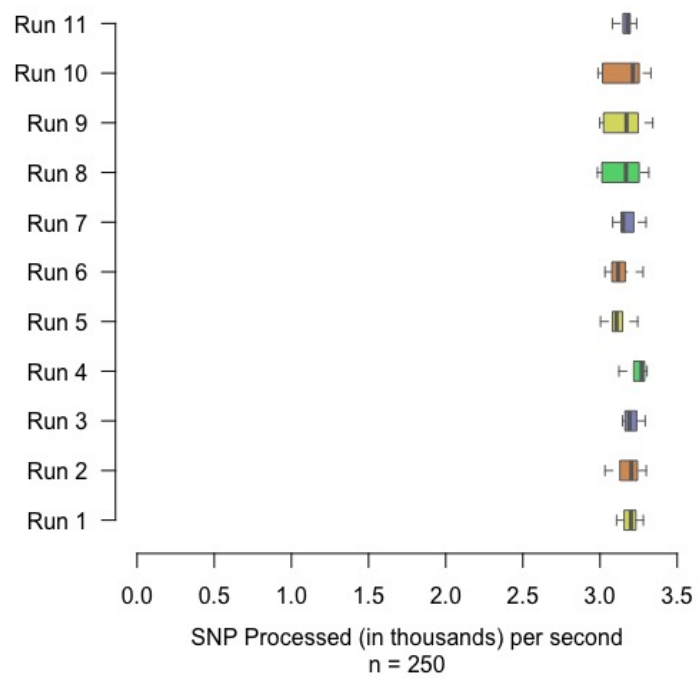


Figure 8.7: Running times for GCA

Chapter 9

Related Work

The advancement in the field of genetic genealogy has paved the way for better understanding and extraction of relevant information from human DNA. As the genetic information is very sensitive in nature, the privacy concerns related to direct-to-consumer DNA testing and its data handling are of prime importance.

Genetic genealogists and computer scientists have published many papers regarding the privacy concerns of human genomic data. In 2018, Garner and Kim published a case study on 23andMe.com and Ancestry.com, analyzing the security concerns related to direct-to-consumer testing [17]. They suggested both the direct-to-consumer genetic-testing industry and federal agencies should reconsider their entire approach towards data privacy. Similarly, the works of Naveed et al. [39], Thomas [36], De Cristofaro et al. [4], De Vries [11] have also raised the issues of ethical, moral and legal obligations

of the genomic data usage. In this thesis we have tried to solve the privacy risks highlighted in the papers by Naveed et al. [39], Thomas [36] and Garner and Kim [17].

To acknowledge and resolve the data-privacy concerns of human genomic data, a good deal of work has been done by the computer-science community using different computation and cryptographic techniques on a variety of genomic data. In this chapter, we have divided the different kinds of privacy-preserving works done by others on the basis of genomic data used.

9.1 Computations on Haplotypes

In 2016, De Cristofaro et al. [10] proposed a protocol for privacy-preserving genetic relatedness test (PPGRT). In this protocol they have used a cloud server to run the relatedness test on an encrypted genetic database. They have used haplotypes (a haplotype is a group of SNPs within an individual that was inherited together from a single parent) to match genetic relatedness between individuals. The working principle of the framework consists of four stages:

- In the first stage, the genotype of an individual is phased into haplotypes by a trusted and certified institution.
- In the second stage, the haplotypes are added into a genetic database.
- In the third stage, the genetic database is encrypted and sourced to a

cloud server or a storage and processing unit(SPU) for genetic matching.

- In the fourth stage, the tester will encrypt its haplotypes using asymmetric key encryption and send it to SPU, the SPU will run tests and return the results to the tester.

The process of phasing the haplotype data from genotyped data introduces errors in the haplotyped data, and even small amounts of phasing error can make an IBD segment undetectable [5]. Ancestry.com is the only direct-to-consumer testing company that uses it, and that too only minimally. Instead, variations of the opposite-homozygote technique are used by most of them [27]. This work is different from our work as we use genotyped data to find genetic relatedness. For the computation of genetic relatedness, De Cristofaro et al. has used a cloud server and a storage unit, which our work has eliminated.

To find relatives while keeping the genomic data private, He et al. [22] proposed a privacy-preserving protocol using haplotyped genomic data. In the proposed protocol, the individuals will follow the outlined steps:

- phase their genotyped data into haplotypes;
- partition the haplotyped data into segments of 300 SNPs;
- encrypt their haplotyped data using a fuzzy-encryption technique and store the data in a public central repository.

To find the relatives, the authors have used the IBD technique to find matching segments. The matching algorithm matches the segments using the secure-sketch concept as proposed by Dodis et al. [13]. Although fuzzy encryption and its implementations allow the computation to handle communication and transmission errors, they do not handle genotyping errors that our work does.

9.2 Computations on STRs

Homomorphic encryption can also be used in privacy-preserving protocols for computing genetic relatedness. The “protocol for privacy-preserving matching of DNA profiles” proposed by Bruekers et al. [8] uses the Paillier encryption scheme to perform homomorphic encryption in a semi-honest attack model. In this work, the authors have used Short Tandem Repeats¹ (STR) as the genomic data to find genetic relatedness between individuals. The input genomic data are expressed as polynomials as proposed by Freedman et al. [16]. The polynomials are encrypted using the asymmetric homomorphic encryption. Then the matching algorithm evaluates the polynomials under encryption using the addition properties of homomorphic encryption. The major difference between the volume of genotyped data and STRs is that works based on STRs involves only a few thousand STRs over the entire genome, as compared to more than half a million SNPs in genotyped data.

¹Certain parts of DNA contain sequences of nucleotides that repeat a number of times in each chromosome, e.g. “ACAAGGTT”. Such a repeated sequence is known as an STR.

9.3 Computations on Genotypes

To protect the genomic data hosted on untrusted clouds, Lauter et al. [33] proposed a solution involving single-key homomorphic encryption. The proposed system enables mutually trusting institutions like hospitals, research facilities, etc. to encrypt and store the data in an untrusted cloud server. The encryption of the data is done by a modified homomorphic-encryption scheme (public-key encryption scheme). The authors have tested their encryption scheme by running the statistical algorithms used in genetic-association studies such as Hardy-Weinberg Equilibrium, Pearson Goodness-Of-Fit Test, EM Algorithm for Haplotyping and Linkage Disequilibrium on genotyped, haplotyped and phenotyped genomic data. The main difference between Lauter’s work and our work is that the involved parties trust each other but not the platform that stores the data and runs the software, whereas we have a peer to peer setup where neither party trusts the other.

Shen et al. [50] proposed privacy-preserving computation on STR profiles using a private-set-intersection (PSI) protocol and random-oblivious-transfer protocol in a semi-honest setting. The authors used the ‘SNPforID 52 SNP-plex assay in paternity testing’ technique, as proposed by Børsting et al. [6] to determine parent-child relationships.

9.4 Other Works

The work of Riazi et al. [48] deals with the privacy-preserving matching of Human Leukocyte Antigen (HLA) for organ transplant using GC protocol. The HLA data are extracted from the fully sequenced genome. The HLA data is processed using “HLA-genotyper” (a python-based library). The HLA data contains a pair of haplotypes (received from both parents). There are 6 pairs of haplotype data, HLA-A, HLA-B, HLA-C, HLA-DQA, HLA-DQB, and HLA-DRB, where each haplotype is a 2-bit string.

Other than organ-transplant compatibility, HLA data can reveal information about genetic ancestry also. Using GC protocol, HLA data of the recipient individuals are matched with a database of possible donors’ HLA data. The software framework performs matching of recipient HLA data on a million-donor database in a few hours on a Intel Core i7-2600 CPU @ 3.4GHz with 12GB RAM.

Chapter 10

Conclusions and Future Work

Conclusion

The goal of this thesis is to create a software platform that can be used to verify individual relatedness without leaking any information. The software implementations aim to keep the data private during the computation, although there is no control over the type of the data (fabricated or original) supplied to our software.

This thesis explores the opposite-homozygotes technique to find the genetic relatedness between individuals using privacy-preserving protocols. IBD segments are established between individuals by dividing the chromosomes into overlapping frames. All the frames between individuals are matched and corresponding results are displayed to both the users.

Computations on the genotyped data of an individual in its raw form lead to

various personal privacy issues. So, we defined, designed and implemented two different software implementations to preserve the privacy of the genotyped data. Both the approaches have their merits and demerits and are concluded as follows:

- HBA is many-fold faster than GCA. The median of 2750 executions show that HBA is executed in 7.5 seconds, whereas the execution time of GCA is dependent on the number of common SNPs shared between individuals. GCA on average can process 3200 SNPs per second, so for a general case of 600,000 common SNPs, GCA will approximately take 188 seconds to complete. In general, an individual will run this software only a few times in his lifetime, so the running time of GCA is acceptable.
- HBA is prone to false negatives while computing the individual relatedness, whereas there are no concerns regarding false negatives in GCA.
- As there is no control over the type of data (fabricated or original) provided to the software, the data security of the approaches is determined considering that the data provided is original and authentic. However, we have checks in place that prevents a malicious user getting knowledge about the data of an innocent user. Keeping the above statement in mind, the data security of GCA is better than that of HBA.

If either of the users runs a malicious modification of our software framework,

they can not learn the data of the other party, or trick the other party into believing they are related when they are not. So, based on the results of our experiments and the arguments presented, GCA performs better than HBA in finding individual relatedness while preserving the data privacy of both users.

Future Work

Based on the literature survey, experimentation results and the experience gathered during the development of this thesis, the future works can be summarised as follows:

- There is a limitation of starting a frame from the first position of a centiMorgan, which does yield a very few missed IBD segments. Instead, a frame structure can be designed where the frames can start from any location in a centiMorgan.
- HBA currently assumes that the communication channel is secure, such that we do not handle the possibility of impersonation, eavesdropping or tampering with the messages both parties exchange. The future work in this scenario would be to assess the impact of this assumption.
- HBA can handle only one genotyping error. If HBA can handle genotyping errors with the 0.1% of the size of each IBD segment then HBA will not be vulnerable to false negatives.

- HBA only processes homozygotes in its computation, leading to certain data risks. Alternatively, a hashing principle might be designed to compute heterozygotes as well.
- The number of gates used in the circuit in GCA can be reduced to reduce the overall processing time of GCA.
- User fairness is a drawback in the malicious 2PC setting. So future work in this regard can help achieve the required goals in GCA.
- A graphical user interface can be implemented to show the length and details of matching segments of the genetic relatedness between users.
- For hyper-sensitive users, a circuit can be designed which can evaluate all the frames at one execution and just output the nature of the relationship, i.e. parent-child, sibling, 3rd cousin, etc.
- For GCA, a frame-matching circuit can be designed that can implicitly count the number of homozygotes in both user's frame. The counting of homozygotes will enforce the security parameter threshold (40% homozygotes) and exclude frames accordingly. This will help in tackling the data-privacy issues of GCA.

Bibliography

- [1] Bruce Alberts, Alexander Johnson, Julian Lewis, Martin Raff, Keith Roberts, and Peter Walter, *The structure and function of DNA*, Molecular Biology of the Cell. 4th edition, Garland Science, 2002.
- [2] Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman, *Basic local alignment search tool*, Journal of Molecular Biology **215** (1990), no. 3, 403–410.
- [3] Elena Andreeva and Bart Preneel, *A three-property-secure hash function*, International Workshop on Selected Areas in Cryptography, Springer, 2008, pp. 228–244.
- [4] Erman Ayday, Emiliano De Cristofaro, Jean-Pierre Hubaux, and Gene Tsudik, *The chills and thrills of whole genome sequencing*, arXiv preprint arXiv:1306.1264 (2013).
- [5] Catherine A Ball, Mathew J Barber, Jake Byrnes, Peter Carbonetto, Kenneth G Chahine, Ross E Curtis, Julie M Granka, Eunjung Han,

- Eurie L Hong, Amir R Kermany, et al., *Ancestry dna matching white paper*, 2020.
- [6] Claus Børsting, Juan J Sanchez, Hanna E Hansen, Anders J Hansen, Hanne Q Bruun, and Niels Morling, *Performance of the snpforid 52 snp-plex assay in paternity testing*, *Forensic Science International: Genetics* **2** (2008), no. 4, 292–300.
- [7] Robert S Boyer and J Moore, *SA fast string searching algorithm*, *Communications of the ACM* **20** (1977), no. 10, 762–772.
- [8] Fons Bruekers, Stefan Katzenbeisser, Klaus Kursawe, and Pim Tuyls, *Privacy-Preserving Matching of DNA Profiles*, Tech. Report 2008/203, IACR Cryptol. ePrint Arch., 2008.
- [9] Ran Canetti, *Security and composition of multiparty cryptographic protocols*, *Journal of Cryptology* **13** (2000), no. 1, 143–202.
- [10] Emiliano De Cristofaro, Kaitai Liang, and Yuruo Zhang, *Privacy-preserving genetic relatedness test*, arXiv preprint arXiv:1611.03006 (2016).
- [11] Jantina De Vries, Susan J Bull, Ogobara Doumbo, Muntaser Ibrahim, Odile Mercereau-Puijalon, Dominic Kwiatkowski, and Michael Parker, *Ethical issues in human genomics research in developing countries*, *BMC Medical Ethics* **12** (2011), no. 1, 1–10.

- [12] Debbie Kennett, *Autosomal DNA statistics*, International Society of Genetic Genealogy Wiki, https://isogg.org/wiki/Autosomal_DNA_statistics, 2018, Accessed: 2020-02-03.
- [13] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith, *Fuzzy extractors: How to generate strong keys from biometrics and other noisy data*, SIAM Journal on Computing **38** (2008), no. 1, 97–139.
- [14] Michael D. Edge and Graham Coop, *Attacks on genetic privacy via uploads to genealogical databases*, *Elife* **9** (2020), e51810.
- [15] Federal Information Processing Standards Publication, *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*, Tech. Report FIPS 202, Information Technology Laboratory, National Institute of Standards and Technology, 2014.
- [16] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas, *Efficient private matching and set intersection*, International conference on the theory and applications of cryptographic techniques, Springer, 2004, pp. 1–19.
- [17] Samuel A Garner and Jiyeon Kim, *The privacy risks of direct-to-consumer genetic testing: A case study of 23andme and ancestry*, *Wash. UL Rev.* **96** (2018), 1219.
- [18] *Genome Reference Consortium Human Build 38 patch release 13*, https://www.ncbi.nlm.nih.gov/assembly/GCF_000001405.39, Accessed: 2020-05-12.

- [19] Gustavo Glusman, Mike Cariaso, Rafael Jimenez, Daniel Swan, Bastian Greshake, Jong Bhak, Darren W Logan, and Manuel Corpas, *Low budget analysis of Direct-To-Consumer genomic testing familial data*, F1000Research **1** (2012), 1:3.
- [20] Vicki Gonzalez, *How DNA ingenuity led to wave of cold case arrests*, KCRA-TV, <https://www.kcra.com/article/dna-cold-case-arrests-golden-state-killer-norcal-rapist/27245135>, 2019, Accessed: 2019-05-10.
- [21] S. Dov Gordon, Carmit Hazay, Jonathan Katz, and Yehuda Lindell, *Complete fairness in secure two-party computation*, J. ACM **58** (2011), no. 6, 24:1–24:37.
- [22] Dan He, Nicholas A Furlotte, Farhad Hormozdiari, Jong Wha J. Joo, Akshay Wadia, Rafail Ostrovsky, Amit Sahai, and Eleazar Eskin, *Identifying genetic relatives without compromising privacy*, Genome research **24** (2014), no. 4, 664–672.
- [23] Brenna M. Henn, Lawrence Hon, J. M. Macpherson, Nick Eriksson, Serge Saxonov, Itsik Pe’er, and Joanna L. Mountain, *Cryptic distant relatives are common in both isolated and cosmopolitan genetic samples*, PLoS One **7** (2012), no. 4.
- [24] Yan Huang, David Evans, Jonathan Katz, and Lior Malka, *Faster secure two-party computation using garbled circuits*, Proceedings of the

- 20th USENIX Conference on Security (Berkeley, CA, USA), SEC'11, USENIX Association, 2011, pp. 35–35.
- [25] Yan Huang, Jonathan Katz, and David Evans, *Efficient secure two-party computation using symmetric cut-and-choose*, Advances in Cryptology – CRYPTO 2013 (Berlin, Heidelberg) (Ran Canetti and Juan A. Garay, eds.), Springer Berlin Heidelberg, 2013, pp. 18–35.
- [26] 23andMe Inc., *Find out what your DNA says about your health, traits and ancestry*, online, <https://www.23andme.com/en-ca/dna-health-ancestry/>, 2020, Accessed: 2020-01-09.
- [27] ISOGG Wiki, *Autosomal DNA match thresholds*, International Society of Genetic Genealogy Wiki, https://isogg.org/wiki/Autosomal_DNA_match_thresholds, 2018, Accessed: 2020-02-08.
- [28] Justin Jouvenal, *To find alleged golden state killer, investigators first found his great-great-great-grandparents*, Washington Post, <https://wapo.st/2Xaj40R>, April 2018, Accessed: 2019-05-10.
- [29] Vladimir Kolesnikov, *Gate evaluation secret sharing and secure one-round two-party computation*, Advances in Cryptology - ASIACRYPT 2005 (Berlin, Heidelberg) (Bimal Roy, ed.), Springer Berlin Heidelberg, 2005, pp. 136–155.
- [30] Vladimir Kolesnikov, Payman Mohassel, and Mike Rosulek, *FlexOR: Flexible Garbling for XOR Gates That Beats Free-XOR*, Advances in

- Cryptology – CRYPTO 2014 (Berlin, Heidelberg) (Juan A. Garay and Rosario Gennaro, eds.), Springer Berlin Heidelberg, 2014, pp. 440–457.
- [31] Vladimir Kolesnikov, Ahmad-Reza Sadeghi, and Thomas Schneider, *Improved garbled circuit building blocks and applications to auctions and computing minima*, Cryptology and Network Security (Berlin, Heidelberg) (Juan A. Garay, Atsuko Miyaji, and Akira Otsuka, eds.), Springer Berlin Heidelberg, 2009, pp. 1–20.
- [32] Vladimir Kolesnikov and Thomas Schneider, *Improved Garbled Circuit: Free XOR Gates and Applications*, Automata, Languages and Programming (Berlin, Heidelberg) (Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfssdóttir, and Igor Walukiewicz, eds.), Springer Berlin Heidelberg, 2008, pp. 486–498.
- [33] Kristin Lauter, Adriana López-Alt, and Michael Naehrig, *Private computation on encrypted genomic data*, International Conference on Cryptology and Information Security in Latin America, Springer, 2014, pp. 3–27.
- [34] J Louhelainen, *SNP Arrays*, Microarrays **5(4):27** (2016).
- [35] Tara C. Matise, *Matise Laboratory of Computational Genetics*, http://compgen.rutgers.edu/download_maps.shtml, 2014, Accessed: 2019-05-10.

- [36] Thomas May, *Sociogenetic risks—ancestry DNA testing, third-party identity, and protection of privacy*, The New England Journal of Medicine **379** (2018), 410–412.
- [37] Sarah McQuate, *Popular third-party genetic genealogy site is vulnerable to compromised data, impersonations*, University of Washington News (2019), no. Oct 29, 2019, Accessed: 2020-01-08.
- [38] Nature.com, *SNP Definition*, Scitable by Nature Education, <https://www.nature.com/scitable/definition/single-nucleotide-polymorphism-snp-295/>, 2019, Accessed: 2019-06-04.
- [39] Muhammad Naveed, Erman Ayday, Ellen W Clayton, Jacques Fellay, Carl A Gunter, Jean-Pierre Hubaux, Bradley A Malin, and XiaoFeng Wang, *Privacy in the genomic era*, ACM Computing Surveys (CSUR) **48** (2015), no. 1, 1–44.
- [40] Peter Ney, Luis Ceze, and Tadayoshi Kohno, *Genotype extraction and false relative attacks: security risks to third-party genetic genealogy services beyond identity inference*, Network and Distributed System Security Symposium (NDSS), 2020.
- [41] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra, *A new approach to practical active-secure two-party computation*, Advances in Cryptology – CRYPTO 2012 (Berlin, Heidel-

- berg) (Reihaneh Safavi-Naini and Ran Canetti, eds.), Springer Berlin Heidelberg, 2012, pp. 681–700.
- [42] Office of Rare Diseases Research. National Institutes of Health, *Terms and definitions*, Terms and Definitions, <https://web.archive.org/web/20120717121400/http://rarediseases.info.nih.gov/Glossary.aspx?acronym=False#C>, 2012, Accessed: 2020-02-03.
- [43] Oracle, *Oracle VirtualBox*, <https://www.virtualbox.org>, Accessed: 2020-04-03.
- [44] Thomas A. Pearson and Teri A. Manolio, *How to interpret a genome-wide association study*, JAMA **299** (2008), no. 11, 1335–1344.
- [45] Personal Genome Project, *Public genetic data*, https://my.pgp-hms.org/public_genetic_data, Accessed: 2019-05-12.
- [46] Michael O Rabin, *How To Exchange Secrets with Oblivious Transfer*, Tech. Report 2005/187, IACR Cryptol. ePrint Arch., 2005.
- [47] Antonio Regalado, *More than 26 million people have taken an at-home ancestry test*, MIT Technology Review (2019), Accessed: 2020-01-03.
- [48] M Sadegh Riazi, Neeraj KR Dantu, LN Vinay Gattu, and Farinaz Koushanfar, *GenMatch: Secure DNA compatibility testing*, 2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), IEEE, 2016, pp. 248–253.

- [49] Ian W Saunders, Jesper Brohede, and Garry N Hannan, *Estimating genotyping error rates from Mendelian errors in SNP array genotypes and their impact on inference*, *Genomics* **90** (2007), no. 3, 291–296.
- [50] Liyan Shen, Xiaojun Chen, Dakui Wang, Binxing Fang, and Ye Dong, *Efficient and private set intersection of human genomes*, 2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), IEEE, 2018, pp. 761–764.
- [51] Ann Turner, *Satiable curiosity: Identity crisis: Identical by state or identical by descent?*, *Journal of Genetic Genealogy* **7** (2011), no. Fall 2011.
- [52] Adam Vaughan, *DNA site GEDmatch sold to firm helping US police solve crime*, *New Scientist*, <https://bit.ly/2PvIIeq>, 2019, Accessed: 2020-01-09.
- [53] Xiao Wang, Samuel Ranellucci, and Jonathan Katz, *Authenticated garbling and efficient maliciously secure two-party computation*, Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (New York, NY, USA), CCS '17, ACM, 2017, pp. 21–37.
- [54] Peter Weiner, *Linear pattern matching algorithms*, 14th Annual Symposium on Switching and Automata Theory (SWAT 1973), IEEE, 1973, pp. 1–11.

- [55] A. C. Yao, *How to generate and exchange secrets*, 27th Annual Symposium on Foundations of Computer Science (FOCS 1986), Oct 1986, pp. 162–167.
- [56] Samee Zahur, Mike Rosulek, and David Evans, *Two halves make a whole*, Advances in Cryptology - EUROCRYPT 2015 (Berlin, Heidelberg) (Elisabeth Oswald and Marc Fischlin, eds.), Springer Berlin Heidelberg, 2015, pp. 220–250.

Appendix A

Data-sets Used in Experiments

Tables A.1, A.2, A.3 and A.4 contain the details of the data sets used in the experiments.

Table A.1: Ancestry family data sets

Index	Individual	Link
1	Son	https://my.pgp-hms.org/profile/huB1557E
2	Father	https://my.pgp-hms.org/profile/hu58B346
3	Mother	https://my.pgp-hms.org/profile/huB08B19
4	Sibling	https://my.pgp-hms.org/profile/hu6A2DBA

Table A.2: 23AndMe data sets

Index	Link
1	https://my.pgp-hms.org/profile/huCB7D85
2	https://my.pgp-hms.org/profile/hu0B4CF6
3	https://my.pgp-hms.org/profile/huD2AD97
4	https://my.pgp-hms.org/profile/hu35E970
5	https://my.pgp-hms.org/profile/hu2C1D99
6	https://my.pgp-hms.org/profile/huC7BB3B
7	https://my.pgp-hms.org/profile/hu39A63C
8	https://my.pgp-hms.org/profile/hu6DB840
9	https://my.pgp-hms.org/profile/hu5EC009
10	https://my.pgp-hms.org/profile/hu1A57E4
11	https://my.pgp-hms.org/profile/hu97688D
12	https://my.pgp-hms.org/profile/hu85F3ED
13	https://my.pgp-hms.org/profile/hu706427
14	https://my.pgp-hms.org/profile/huAFE8F3
15	https://my.pgp-hms.org/profile/huEC682F
16	https://my.pgp-hms.org/profile/huF7A4DE

Table A.3: Ancestry data sets

Index	Link
1	https://my.pgp-hms.org/profile/hu8D9A84
2	https://my.pgp-hms.org/profile/hu1C1368
3	https://my.pgp-hms.org/profile/hu6036FB
4	https://my.pgp-hms.org/profile/hu1247AF
5	https://my.pgp-hms.org/profile/huB19789
6	https://my.pgp-hms.org/profile/huA29F28

Table A.4: FTDNA data sets

Index	Link
1	https://my.pgp-hms.org/profile/hu85F3ED
2	https://my.pgp-hms.org/profile/hu26B551
3	https://my.pgp-hms.org/profile/hu436A1D
4	https://my.pgp-hms.org/profile/hu502084
5	https://my.pgp-hms.org/profile/huBC964C
6	https://my.pgp-hms.org/profile/hu95E46D
7	https://my.pgp-hms.org/profile/hu3C5109
8	https://my.pgp-hms.org/profile/huFAA0CF
9	https://my.pgp-hms.org/profile/hu971175
10	https://my.pgp-hms.org/profile/hu7B4393
11	https://my.pgp-hms.org/profile/hu5CABA7
12	https://my.pgp-hms.org/profile/hu19E9D2
13	https://my.pgp-hms.org/profile/hu436A1D
14	https://my.pgp-hms.org/profile/huAEC1B0
15	https://my.pgp-hms.org/profile/hu4A6650
16	https://my.pgp-hms.org/profile/huB96844

Vita

Candidate's full name: Syed Zeeshan Ahmed

University attended (with dates and degrees obtained):

- Amity University, 2008-2012, B.Tech. Computer Science and Engineering with First Class
- University of New Brunswick, 2018-2020, MCS

Contributed talks:

- S.Z. Ahmed, O. Kaser, D. Lemire, 'Preserving Consumer DNA Privacy For Finding Relatives In A Malicious Two Party Computation', Science Atlantic Conference for Mathematics, Statistics and Computer Science, Halifax, Research Talk, October 2019.