SECURE BIOMETRIC COMPUTATION AND OUTSOURCING

A Dissertation

Submitted to the Graduate School

of the University of Notre Dame

in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

by

Fattaneh Bayatbabolghani

Marina Blanton, Co-Director

Aaron Striegel, Co-Director

Graduate Program in Computer Science and Engineering

Notre Dame, Indiana

June 2017

SECURE BIOMETRIC COMPUTATION AND OUTSOURCING

Abstract

by

Fattaneh Bayatbabolghani

Biometric computations are becoming increasingly popular, and their impact in real world applications is undeniable. There are different types of biometric data used in a variety of applications such as biometric recognition including verification and identification. Because of the highly sensitive nature of biometric data, its protection is essential during biometric computations.

Based on the computations that need to be carried out and the type of biometric data, there is a need for application-specific privacy preserving solutions. These solutions can be provided by developing secure biometric computations in a way that no information gets revealed during the protocol execution. In some biometric applications (e.g. verification and identification), data is distributed amongst different parties engaged in computations. In some other applications, the execution of computations is bounded by the computational power of the parties, motivating the use of cloud or external servers. In both these cases, 1) there is a higher risk for sensitive data to be disclosed, making secure biometric protocols a prominent need for such practical applications, 2) it is more challenging to develop novel and efficient solutions for these computational settings, making the design of secure biometric protocols a research-worthy effort.

In our research, we worked on three different biometric modalities which require various computational settings. In more detail:

- We focused on voice recognition in semi-honest and malicious adversarial models using floating point arithmetic (the most common and accurate type of data for voice recognition). Based on the application, we considered two secure computational settings which are two-party and multi-party settings. For this purpose, we designed new secure floating-point operations necessary for voice recognition computations.
- We designed novel and general approaches to securely compute three genomic tests (i.e., paternity, ancestry, and genomic compatibility). We considered server aided two-party computation for those applications. Also, based on the genomic computational settings we proposed novel certified inputs technique to provide stronger security guarantees with respect to malicious users.
- We built secure fingerprint recognition protocols which include both alignment (for the first time) and matching processes. The solutions were proposed in both two-party and multi-party computational settings. We also designed a number of new secure and efficient protocols for essential operations in fingerprint recognition process.

To the best of our knowledge, our unique contributions in different biometric modalities largely benefit the field of secure biometric computation. Our solutions consider the nature of computational setting (two-party and multi-party settings) of biometric applications in practice, and efficiently protect the data during the computations. In addition, our secure protocols and building blocks are general and can be used for other applications where the same data types and settings are used.

CONTENTS

FIGURES

TABLES

ACKNOWLEDGMENTS

I would like to thank so many people for helping me during this journey. They made my last four years a lot easier that I thought it was going to be. I am truly appreciative for their support.

I would like to genuinely thank my advisor, Prof. Marina Blanton. I learned many lessons from her, from foundations of scientific research, to being a hard-working, motivated, passionate, and professional researcher. I also learned from her to be much more open to discussions and to take into account different points of view before proposing a new idea. I will always be grateful for the opportunity that I was given to work under her advice.

In addition, I was privileged to work with Prof. Mehrdad Aliasgari, a knowledgeable and inspiring researcher, in different research projects. I am really thankful for all his support throughout my Ph.D. years, and his early guidance before starting my work at Notre Dame.

I would like to deeply thank Prof. Aaron Striegel, Prof. Walter Scheirer, and Prof. Scott Emrich for providing invaluable guidance and being always available to help and support me during the last year of my Ph.D.

I also would like to thank the CSE family especially, Joyce Yeats and Ivor Zhang among all. Joyce was truly helpful to me particularly in academic procedures, and Ivor was a friendly collaborator and his ideas were always constructive in my research. Moreover, I am thankful for having wonderful and supportive friends; they kept me energetic and delighted during the last four years.

Last but not least, I would like to acknowledge with support and love of my family-My parents, Asgar and Ehteram; my husband, Kayhan; my brother and sister, Saeed and Maryam. Without all their helps I doubt that I would reach up to this place today.

CHAPTER 1

INTRODUCTION

Biometric data is a type of data which comes from human characteristics such as iris, voice, fingerprint, and DNA. Also, different types of biometric data have become increasingly ubiquitous during recent decades. Because of the highly sensitive nature of biometric data, protection is crucial in a variety of applications such as biometric recognition for verification and identification.

Secure computation on biometric data is an emerging topic and it has been drawing growing attention during recent years. There are different types of biometric data with a variety of usages. Based on the type of biometric modality and its application, we need to follow specific computations. These computations are usually carried out on a large number of inputs, and it demands high computational power. Therefore, optimizing the computations is necessary and makes this problem more challenging.

Also, oftentimes the data owner is computationally limited. Thus, outsourcing the computations makes the solution more efficient and more practical in real world settings. However, assigning computations to the cloud or servers increases the risk of revealing sensitive data. Thus, designing secure solutions for outsourced biometric computations is even more challenging than the standard case.

Consequently, to address secure regular and outsourced biometric computations, different computational settings are considered as follows:

- *Two-party computation:* Each party has his/her own inputs, and both are going to jointly compute a function. Eventually, one party or both learn the output depending on the predefined settings and the utilized techniques.

- *Server aided two-party computation:* In some applications such as non-medical use of genomic data, computations often take place in a server-mediated setting where the server offers the ability for joint computations between the users. In this setting, the two-party scenario is enhanced with a server which helps the parties to compute the function, while no information is being revealed to the server.

- *Multi-party computation:* In this setting, usually one or two parties are going to outsource data to a collection of outside parties to carry out computations collectively. The assumption here is the number of non-colluding parties is bigger than a predetermined threshold; otherwise, the data can be compromised.

In addition to the computational settings introduced above, there are adversarial models to be considered for the secure computations. Here, we consider two standard adversarial models:

- *Semi-honest or passive:* This type of adversary is a party who correctly follows the protocol specification, but it attempts to learn additional information by analyzing the transcript of messages received during the execution.

- *Malicious or active:* This type of adversary is a party who can arbitrarily deviate from the protocol specification.

Furthermore, each solution (based on the biometric modality and its usage) carries out specific computations which contain some operations as building blocks. In addition, the structure of the building blocks may differ depending on the data type (i.e., integer, fixed point, and floating point), the secure computational setting, adversarial model, and applied tools (e.g., garbled circuit and secret sharing). Therefore, this demands some effort to address all the security concerns and consider all varieties of settings in biometric applications.

As a result of these research pursuits, one of the innovative contributions of our work is the building of general and time-efficient privacy-preserving solutions and protocols which may be used not only with other biometric modalities, but also in other types of computations where the privacy of the data is one of the main concerns [12, 24, 31, 32, 109]. Furthermore, we worked on three common modalities that have different real world applications: (1) Voice is utilized for speaker-based authentication. (2) Genomic tests are used for a verity of purposes; for instance, they help find common ancestors between two people (i.e., an ancestry test). (3) Fingerprint is one of the most accurate types of biometric data for the purpose of verification and identification. The following three sections will address these common modalities in more detail.

1.1 Voice Recognition

Voice recognition includes speech recognition and speaker recognition. For both tasks Hidden Markov Models (HMMs) are the most common and accurate approaches. Thus, we concentrated on secure computation of HMMs. Furthermore, to ensure that Gaussian mixture models (GMMs), which are commonly used in HMM computation, can be a part of the secure computation, GMM computation is integrated into the privacy-preserving solution.

In [12], we provided a privacy-preserving solution for the Viterbi decoding algorithm (the most commonly used in voice recognition), but the techniques can be used to securely execute other HMM decoding algorithms (e.g. the forward and the expectation maximization algorithms) as well. In addition, we developed secure techniques for computation with floating-point numbers, which provide adequate precision and are the most appropriate for HMM computation. As a matter of fact, we developed and implemented secure HMM computations in semi-honest model using floating point numbers. The solution is also proven to have a reasonable performance.

From that point onward we continued to designed secure floating-point protocols for necessary HMM operations (multiplication, comparison, truncation, inversion, prefix multiplication, and bit decomposition) in stronger security circumstances for the first time where the computational parties can arbitrarily deviate from the protocol specification (named as the malicious party). These protocols also have applicability well beyond the HMM domain. Their rigorous simulation-based proofs of security are constructed from scratch. They are the most challenging part of this work, but constitute substantial contributions.

1.2 DNA Computation

During recent years the use of genomic data in a variety of applications has rapidly expanded. It is especially important to protect genomic data because it can reveal information not only about the data owner but also his/her relatives. Thus, this portion of our research focused on general privacy-preserving protocols and solutions which can be used for common DNA applications such as ancestry, paternity, and genomic compatibility tests. These applications are normally facilitated by some service provider or third party. Such service providers are as a point of contact for aiding the individuals with private computations of their sensitive genomic data. Therefore, it is more appropriate to assume that such computations are carried out by two individuals through some third-party service provider.

Another important consideration from a security point of view is enforcing correct (i.e., truthful) inputs to be entered into the computation. This requirement is outside of the traditional security model and normally is not addressed, but it becomes important in the context of genomic computation. This is because for certain types of genomic tests it is easy for one participant to modify his inputs and learn sensitive information about genetic conditions of the other party. For these reasons it is vital that participants should be prevented from modifying their inputs used in genomic computations.

In [31, 32], I studied private genomic computations in the light of server-mediated or server-aided settings and utilized the server to lower the cost of the computations for the participants. In addition, and also for the first time, stronger security guarantees with respect to malicious parties are provided. In particular, novel certified inputs are incorporated into secure computations to guarantee that a malicious user is unable to modify her inputs in order to learn information about the data of the other user. The solutions are general in the sense that they can be used for other computations and offer reasonable performance compared to the state of the art.

1.3 Fingerprint Recognition

Fingerprint recognition is an accurate method of identification. Fingerprints need to be protected because they can reveal the identity of their owners. There are some different techniques for fingerprint recognition, but most of them have two main steps: the first step is alignment and the second one is matching. The purpose of the alignment step is to improve the accuracy of the matching step. To the best of our knowledge, all existing articles on secure fingerprint recognition focused on the matching step because the alignment step is computationally expensive, albeit an interesting challenge.

Therefore, we aimed to design an efficient solution to the entire recognition process including both the alignment and matching steps for the first time, in order to achieve more reliable results [24]. We focused on three algorithms that compare fingerprints using both alignment and matching. The algorithms are selected based on their precision, speed, and/or popularity, with the goal of building efficient and practical security solutions.

The proposed solutions carry out specific computations, and they contain complex operations that require dealing with non-integer values. As a result, a substantial part of this work is dedicated to building practical and time-efficient secure protocols (sine, cosine, arctangent, square-root, and selecting the fth smallest element) for non-integer operations.

1.4 Organization

In the rest of this dissertation: The related literature to this work is described in Chapter 2. Chapter 3 provides all necessary cryptographic background and notations. In Chapters 4–6, we describe our main contributions in terms of designing secure, efficient, and novel privacy-preserving protocols, and our proposed solutions respectively for voice recognition, DNA computation, and fingerprint recognition. And finally, at the end we discuss some conclusionary remarks and future directions in Chapter 7.

CHAPTER 2

RELATED WORK

In this chapter, we are going to describe the related literature in secure biometric computations. Firstly, we introduce previous voice recognition works in Section 2.1, then we provide the recent works in the context of secure genomic computation in Section 2.2. Finally, Section 2.3 is dedicated to explain related work in secure fingerprint recognition.

2.1 Related Work in Secure Voice Recognition

To the best of our knowledge, privacy-preserving HMM computation was first considered in [113], which provides a secure two-party solution based on homomorphic encryption for speech recognition using integer representation of values. In general, integer representation is not sufficient for HMM computation because it involves various operations on probability values, which occupy a large range of real numbers and demand high precision. In particular, probabilities need to be repeatedly multiplied during HMM computation, and the resulting product can quickly diminish with each multiplication, leading to inability to maintain precision using integer or fixed-point representation. [113] computes this product using logarithms of the values, which becomes the sum of logarithms (called logsum in [113]). This allows the solution to retain some precision even with (scaled) integer representation, but the computation was nevertheless not shown to be computationally stable and the error was not quantified. Also, as was mentioned in [60], one of the building blocks in [113] is not secure.

The techniques of [113] were later used as-is in [111] for Gaussian mixture models. The same idea was used in [104], [105] to develop privacy-preserving speaker verification for joint two-party computation, where the HMM parameters were stored in an encrypted domain. Also, [106] treats speaker authentication and identification and speech recognition in the same setting. Similar to [111], [103] aimed at providing secure two-party GMM computation using the same high-level idea, but with implementation differences. The solution of [103], however, has security weaknesses. In particular, the protocol reveals a non-trivial amount of information about the private inputs, which, in combination with other computation or outside knowledge, may allow for full recovery of the inputs (additional detail about this security weakness is provided in [10]). Some of the above techniques were also used in privacy-preserving network analysis and anomaly detection in two-party or multi-party computation [100, 101].

Another work [107] builds a privacy-preserving protocol for HMM computation in the two-party setting using a third-party commodity server to aid the computation. In [107], one participant owns the model and the other holds observations. We build a more general solution that can be applied to both two-party (without an additional server) and multi-party settings, uses high precision floating-point arithmetic, and is secure in a stronger security setting (in the presence of malicious participants).

All of the above work uses integer-based representations, where in many cases multiplications were replaced with additions of logarithms, as originated in [113]. With the exception of [106], these publications did not quantify the error, while using integer (or fixed-point) representation demands substantially larger bit length representation than could be used otherwise and the error can accumulate and introduce fatal inaccuracies. [106] evaluated the error and reports that it amounted to 0.52% for their specific set of parameters.

The need to use non-integer representation for HMM computation was recognized in [61] and the authors proposed solutions for secure HMM forward algorithm computation in the two-party setting using logarithmic representation of real numbers. The solution that uses logarithmic representation was shown to be accurate for HMM computation used in bioinformatics (see [60]), but it still has its limitations. In particular, the look-up tables used in [61] to implement certain operations in logarithmic representations grow exponentially in the bitlength of the operands. This means that the approach might not be suitable for some HMM applications or a set of parameters. The use of floating-point numbers, on the other hand, allows one to avoid the difficulties mentioned above and provides a universal solution that works for any application with a bounded (and controlled) error. Thus, we decided to address the need to develop secure computation techniques for HMMs on standard real number representations and provide the first provably secure floating-point solution for HMM algorithms, which initially appeared in [10].

2.2 Related Work in Secure DNA Computation

There are a number of publications, e.g., [16, 18, 19] and others, that treat the problem of privately computing personalized medicine tests with the goal of choosing an optimal medical treatment or drug prescription. Ayday et al. [17] also focus on privacy-preserving systems for storing genomic data by means of homomorphic encryption.

To the best of our knowledge, privacy-preserving paternity testing was first considered by Bruekers et al. in [36]. The authors propose privacy-preserving protocols for a number of genetic tests based on Short Tandem Repeats (STRs) (see Section 5.3 for detail). The tests include identity testing, paternity tests with one and two parents, and common ancestry testing on the Y chromosome. The proposed protocols for these tests are based on additively homomorphic public key encryption and are secure in the presence of semi-honest participants. Implementation results were not given in [36], but Baldi et al. [20] estimates that the paternity test in [36] is several times slower than that in [20]. We thus compare our paternity test to the performance of an equivalent test in [20].

Baldi et al. [20] concentrate on a different representation of genomic data (in the form of fully-sequenced human genome) and provide solutions for paternity, drug testing for personalized medicine, and genetic compatibility. The solutions use private set intersection as the primary cryptographic building block in the two-party server-client setting. They were implemented and shown to result in attractive runtimes and we compare the performance of our paternity and compatibility tests to the results reported in [20] in Section 5.7.

Related to that is the work of De Cristofaro et al. [56] that evaluates the possibility of using smartphones for performing private genetic tests. It treated paternity, ancestry, and personalized medicine tests. The protocol for the paternity test is the same as in [20] with certain optimizations for the smartphone platform (such as performing pre-processing on a more powerful machine). The ancestry test is performed by sampling genomic data as using inputs of large size deemed infeasible on a smartphone. The implementation also used private set intersection as the building block. Our implementation, however, can handle inputs of very large sizes at low cost.

Two recent articles [69, 72] describe mechanisms for private testing for genetic relatives and can detect up to fifth degree cousins. The solutions rely on fuzzy extractors. They encode genomic data in a special form and conduct testing on encoded data. The approach is not comparable to the solutions we put forward here as [69, 72] are based on non-interactive computation and is limited to a specific set of functions.

Although not as closely related to our work as publications that implement specific genetic tests, there are also publications that focus on applications of string matching to DNA testing. One example is the work of De Cristofaro et al. [57] that provides a secure and efficient protocol that hides the size of the pattern to be searched and its position within the genome. Another example is the work of Katz et al. [81] that applies secure text processing techniques to DNA matching.

2.3 Related Work in Secure Fingerprint Recognition

The first work to treat secure fingerprint comparisons in particular is due to Barni *et al.* [22]. Their method utilizes the FingerCode representation of fingerprints (which uses texture information from a fingerprint) and they secure their method using a homomorphic encryption scheme. FingerCode-based fingerprint comparisons can be implemented efficiently; hence, the method of Barni *et al.* is relatively fast. Unfortunately, the FingerCode approach is not as discriminative as other fingerprint recognition algorithms (in particular, minutia-based algorithms) and is not considered suitable for fingerprint-based identification suitable for criminal trials.

Huang *et al.* [74] propose a privacy-preserving protocol for biometric identification that is focused on fingerprint matching and uses homomorphic encryption and garbled circuit evaluation. In their fingerprint matching protocol, the FingerCode representation is utilized and the matching score is computed using the Euclidean distance. They provide some optimizations, such as using off-line execution and fewer circuit gates, which make their solution more efficient than prior work. Nevertheless, their method still suffers from the lack of accuracy derived from being based on the FingerCode representation.

Blanton and Gasti [33, 34] also improve the performance of secure fingerprint comparisons based on FingerCodes and additionally provide the first privacy-preserving solution for minutia-based fingerprint matching. Their methods use a combination of homomorphic encryption and garbled circuit evaluation, but assume that fingerprints are independently pre-aligned. That is, they treat the matching step only, not the more difficult alignment step. To compare two fingerprints, T and S, consisting of pre-aligned sets of m and n minutia points, respectively, their algorithm considers each point t_i of T in turn, determines the list of points in S within a certain distance and orientation from s_i that have not yet been paired up with another point in T. If this list is not empty, t_i is paired up with the closest point on its list. The total number of paired up points is the size of matching, which can consequently be compared to a threshold to determine whether the fingerprints are related or not. Although their method is lacking in the alignment step, we use similar logic for computing matching between two transformed fingerprints in our protocols.

Shahandashti *et al.* [108] also propose a privacy-preserving protocol for minutia-based fingerprint matching, which uses homomorphic encryption and is based on evaluation of polynomials in encrypted form. The complexity of their protocol is substantially higher than that of Blanton and Gasti [34], however, and their method can also introduce an error when a minutia point from one fingerprint has more than one minutia point from the other fingerprint within a close distance and orientation from it.

More recently, Blanton and Saraph [35] introduce a privacy-preserving solution for minutia-based fingerprint matching that formulates the problem as determining the size of the maximum flow in a bipartite graph, which raises questions of practicality for this method. The algorithm is guaranteed to pair the minutiae from S with the minutiae from T in such a way that the size of the pairing is maximal (which previous solutions could not achieve). The algorithm can be used in both two-party and multi-party settings, but only the two-party protocol based on garbled circuit evaluation was implemented.

Lastly, Kerschbaum *et al.* [82] propose a private fingerprint verification protocol between two parties that includes both alignment and matching steps. Unfortunately, the solution leaks information about fingerprint images used and the authors also used a simplified alignment and matching computation that is not as robust to fingerprint variations as other algorithms.

CHAPTER 3

PRELIMINARIES

In this chapter, we describe some essential cryptographic background that we used in our work. Firstly, we talk about homomorphic encryption, secret sharing, and garbled circuit evaluation techniques in Sections 3.1.1–3.1.3. Then, we talk about essential and known building blocks which are used in this thesis in Section 3.2. We discuss signature scheme, commitment scheme, and zero-knowledge proofs of knowledge in Sections 3.3 and 3.4. Lastly, we discuss the security model in Section 3.5.

3.1 Secure Two-Party and Multi-Party Computational Techniques

In this section, we explain the two-party (homomorphic encryption and garbled circuit evaluation) and multi-party (secret sharing) computational techniques that we use in our research.

3.1.1 Homomorphic Encryption

Homomorphic encryption is a type of encryption that allows computations to be performed on encrypted data without revealing any information from the data. In this thesis, we use an specific type of homomorphic encryption where its key is defined in a public-key cryptosystem. This scheme is defined by three algorithms (Gen, Enc, Dec), where Gen is a key generation algorithm that on input of a security parameter 1^κ produces a public-private key pair (pk, sk); Enc is an encryption algorithm that on input of a public key pk and message m produces ciphertext c; and Dec is a decryption algorithm that on input of a private key sk and ciphertext c produces decrypted message m or special character $\perp$ that indicates failure. For conciseness, we use notation $\mathsf{Enc}_{pk}(m)$ or $\mathsf{Enc}(m)$ and $\mathsf{Dec}_{sk}(c)$ or $\mathsf{Dec}(c)$ in place of $\mathsf{Enc}(pk, m)$ and $\mathsf{Dec}(sk, c)$, respectively. An encryption scheme is said to be additively homomorphic if applying an operation to two ciphertexts results in the addition of the messages that they encrypt, i.e., $\mathsf{Enc}_{pk}(m_1) \cdot \mathsf{Enc}_{pk}(m_2) = \mathsf{Enc}(m_1 + m_2)$. This property also implies that $\mathsf{Enc}_{pk}(m)^k = \mathsf{Enc}_{pk}(k \cdot m)$ for a known k. In a public-key (np, t)-threshold encryption scheme, the decryption key sk is partitioned among np parties, and $t \leq np$ of them are required to participate in order to decrypt a ciphertext while $t - 1$ or fewer parties cannot learn anything about the underlying plaintext. Lastly, a semantically secure encryption scheme guarantees that no information about the encrypted message can be learned from its ciphertext with more than a negligible (in κ) probability. One example of a semantically secure additively homomorphic threshold public-key encryption scheme is Paillier encryption [102].

In this setting, computation of a linear combination of protected values (addition, subtraction, multiplication by a known integer) can be performed locally by each participant on encrypted values, while multiplication is interactive. In the two-party setting based on homomorphic encryption, interactive operation (e.g. multiplication and jointly decrypting a ciphertext) in particular the round complexity determines efficiency of a computation. In this setting, public-key operations (and modulo exponentiations in particular) also impose a significant computational overhead, and are used as an additional performance metric.

3.1.2 Secret Sharing

Secret sharing techniques allow for private values to be split into random shares, which are distributed among a number of parties, and perform computation directly on secret shared values without computationally expensive cryptographic operations.

Of a particular interest to us are linear threshold secret sharing schemes. With a (np, t)-secret sharing scheme, any private value is secret-shared among np parties such that any $t + 1$ shares can be used to reconstruct the secret, while t or fewer parties cannot learn any information about the shared value, i.e., it is perfectly protected in the information-theoretic sense. In a linear secret sharing scheme, a linear combination of secret-shared values can be performed by each party locally, without any interaction, but multiplication of secret-shared values requires communication between all of them. In a linear secret sharing scheme, any linear combination of secret shared values is performed by each participant locally (which in particular includes addition and multiplication by a known), while multiplication requires interaction of the parties. It is usually required that $t < np/2$ which implies $np \geq 3$. In here, we assume that Shamir secret sharing scheme [110] is used with $t < np/2$ in the semi-honest setting for any $np \geq 3$ malicious players.

In this setting, we can distinguish between the input owner who provide input data into the computation (by producing secret shares), computational parties who conduct the computation on secret-shared values, and output recipients who learn the output upon computation termination (by reconstructing it from shares). These groups can be arbitrarily overlapping and be composed of any number of parties as long as there are at least 3 computational parties.

Also, in secret sharing, as we mentioned, computation of a linear combination of protected values can be performed locally by each participant, while multiplication is interactive. Because often the overhead of interactive operations dominates the runtime of a secure multi-party computation algorithm base on secret sharing, its performance is measured in the number of interactive operations (such as multiplications, as well as other instances which, for example, include opening a secret-shared value). Furthermore, the round complexity, i.e., the number of sequential interactions, can have a substantial impact on the overall execution time, and serves as the second major performance metric.

3.1.3 Garbled Circuit Evaluation

The use of garbled circuit allows two parties P_1 and P_2 to securely evaluate a Boolean circuit of their choice. That is, given an arbitrary function $f(x_1, x_2)$ that depends on private inputs x_1 and x_2 of P_1 and P_2, respectively, the parties first represent is as a Boolean circuit. One party, say P_1, acts as a circuit generator and creates a garbled representation of the circuit by associating both values of each binary wire with random labels. The other party, say P_2, acts as a circuit evaluator and evaluates the circuit in its garbled representation without knowing the meaning of the labels that it handles during the evaluation. The output labels can be mapped to their meaning and revealed to either or both parties.

An important component of garbled circuit evaluation is 1-out-of-2 Oblivious Transfer (OT). It allows the circuit evaluator to obtain wire labels corresponding to its inputs. In particular, in OT the sender (i.e., circuit generator in our case) possesses two strings s_0 and s_1 and the receiver (circuit evaluator) has a bit σ. OT allows the receiver to obtain string s_σ and the sender learns nothing. An oblivious transfer extension allows any number of OTs to be realized with small additional overhead per OT after a constant number of regular more costly OT protocols (the number of which depends on the security parameter). The literature contains many realizations of OT and its extensions, including very recent proposals such as [14, 76, 99] and others.

The fastest currently available approach for circuit generation and evaluation we are aware of is by Bellare et al. [26]. It is compatible with earlier optimizations, most notably the "free XOR" gate technique [84] that allows XOR gates to be processed without cryptographic operations or communication, resulting in virtually no overhead for such gates. A recent half-gates optimization [116] can also be applied to this construction to reduce communication associated with garbled gates.

Note that, in the two-party setting solution based on garbled circuit, the complexity of an operation is measured in the number of non-free (i.e., non-XOR) Boolean gates because of optimization in XOR gate. Also, some computations like shift operation does not consist of any kind of gate and it is totally free. Therefore, to have an optimize solution, we need to minimize the number of non-XOR gates by using more free operations during the computation instead.

3.2 Secure Building Blocks

In this section, we give a brief description of the building blocks for integer, fixed-point, and floating-point operations from the literature used in our solutions. First note that having secure implementations of addition and multiplication operations alone can be used to securely evaluate any functionality on protected values represented as an arithmetic circuit. Prior literature, however, concentrated on developing secure protocols for commonly used operations which are more efficient than general techniques. In particular, the literature contains a large number of publications for secure computation on integers such as comparisons, bit decomposition, and other operations. From all of the available techniques, we have chosen the building blocks that yield the best performance for our construction because efficient performance of the developed techniques is one of our primary goals.

Throughout this thesis, we use notation [x] to denote that the value of x is protected and not available to any participant in the clear. In the complexity of two-party setting based in homomorphic encryption, notation C denotes the ciphertext length in bits, and D denotes the length of the auxiliary decryption information, which when sent by one of the parties allows the other party to decrypt a ciphertext. Communication is also measured in bits. In this thesis, we list computational overhead incurred by each party (in homomorphic encryption) separately, with the smaller amount of work first (which can be carried out by a client) followed by the larger amount of work (which can be carried out by a server).

3.2.1 Fixed-Point and Integer Building Blocks

Some of the operations for multi-party computation based on secret sharing and two-party computation based on garbled circuit used in this thesis are elementary and are well-studied in the security literature (e.g., [29, 30, 45, 46]), while others are more complex, but still have presence in prior work (e.g., [15, 28, 29, 46]). When it is relevant to the discussion, we assume that integer values are represented using ℓ bits and fixed-point values are represented using the total of ℓ bits, k of which are stored after the radix point (and thus $\ell - k$ are used for the integer part). Here is the list of fixed-point and/or integer building blocks for multi-party setting based on secret sharing and/or two-party setting based on garbled circuit:

- *Addition* $[c] \leftarrow [a] + [b]$ and *subtraction* $[c] \leftarrow [a] - [b]$ are considered free (non-interactive) using secret sharing using both fixed-point and integer representations [46]. Their cost is ℓ non-free gates for ℓ-bit a and b [85] using garbled circuit for both integer and fixed-point representations.

- *Multiplication* $[c] \leftarrow [a] \cdot [b]$ of integers involves 1 interactive operation (in 1 round) using secret sharing. For fixed-point numbers, truncation of k bits is additionally needed, resulting in $2k + 2$ interactive operations in 4 rounds (which reduces to 2 rounds after pre-computation) [46]. Using garbled circuit, multiplication of ℓ-bit values (both integer and fixed-point) involves $2\ell^2 - \ell$ non-free gates using the traditional algorithm [85]. This can be reduced using the Karatsuba's method [80], which results in fewer gates when $\ell > 19$ [70]. Note that truncation has no cost in Boolean circuits.

- *Comparison* $[c] \leftarrow \mathsf{LT}([a], [b])$ that tests for $a < b$ (and other variants) and outputs a bit involves $4\ell - 2$ interactive operations in 4 rounds (which reduces to 3 rounds after pre-computation) using secret sharing [45] (alternatively, $3\ell - 2$ interactive operations in 6 rounds). This operation costs ℓ non-free gates using garbled circuit [85]. Both implementations work with integer and fixed-point values of length ℓ.

- *Equality testing* $[c] \leftarrow \mathsf{EQ}([a], [b])$ similarly produces a bit and costs $\ell + 4 \log \ell$ interactive operations in 4 rounds using secret sharing [45]. Garbled circuit based implementation requires ℓ non-free gates [84]. The implementations work with both integer and fixed-point representations.

- *Division* $[c] \leftarrow \mathsf{Div}([a], [b])$ is available in the literature based on different underlying algorithms and we are interested in the fixed-point version. A fixed-point division based on secret sharing is available from [46] which uses Goldschmidt's method. The algorithm proceeds in ξ iterations, where $\xi = \lceil \log_2(\frac{\ell}{3.5}) \rceil$. The same underlying algorithm could be used to implement division using garbled circuit, but we choose to use the readily available solution from [29] that uses the standard (shift and subtraction) division algorithm. The complexities of these implementations are given in Tables 3.1 and 3.2.

- *Integer to fixed-point conversion* $[b] \leftarrow \mathsf{Int2FP}([a])$ converts an integer to the fixed-point representation by appending a number of zeros after the radix point. It involves no interactive operations using secret sharing and no gates using garbled circuit.

- *Fixed-point to integer conversion* $[b] \leftarrow \mathsf{FP2Int}([a])$ truncates all bits after the radix point of its input. It involves no gates using garbled circuit and costs $2k+1$ interactive operations in 3 rounds to truncate k bits using secret sharing [46].

- *Conditional statements with private conditions* of the form "if $[priv]$ then $[a] = [b]$; else $[a] = [c]$;" are transformed into statements $[a] = ([priv] \wedge [b]) \vee (\neg [priv] \wedge [c])$, where b or c may also be the original value of a (when only a single branch is present). Our optimized implementation of this statement using garbled circuit computes $[a] = ([priv] \wedge ([b] \oplus [c])) \oplus [c]$ with the number of non-XOR gates equal to the bitlength of variables b and c. Using secret sharing , we implement the statement as $[a] = [priv] \cdot ([b] - [c]) + [c]$ using a single interactive operation.

- *Maximum or minimum* of a set $\langle [a_{max}], [i_{max}] \rangle \leftarrow \mathsf{Max}([a_1], \ldots, [a_m])$ or $\langle [a_{min}], [i_{min}] \rangle \leftarrow \mathsf{Min}([a_1], \ldots, [a_m])$, respectively, is defined to return the maximum/minimum element together with its index in the set. The operation costs $2\ell(m-1) + m + 1$ non-free gates using garbled circuit, where ℓ is the bitlength of the elements a_i. Using secret sharing , the cost is dominated by the comparison operations, giving us $4\ell(m-1)$ interactive operations. Instead of performing comparisons sequentially, they can be organized into a binary tree with $\lceil \log m \rceil$ levels of comparisons. Then in the first iteration, $m/2$ comparisons are performed, $m/4$ comparisons in the second iteration, etc., with a single comparison in the last iteration. This allows the number of rounds to grow logarithmically with m and give us $4\lceil \log m \rceil + 1$ rounds.

  When each record of the set contains multiple fields (i.e., values other than those being compared), the cost of the operation increases by $m-1$ non-free gates for each additional *bit* of the record using garbled circuit and by $m-1$ interactive operations for each additional *field element* of the record without increasing the number of rounds.

- *Prefix multiplication* $\langle [b_1], \ldots, [b_m] \rangle \leftarrow \mathsf{PreMul}([a_1], \ldots, [a_m])$ simultaneously computes $[b_i] = \prod_{j=1}^{i} [a_j]$ for $i = 1, \ldots, m$. We also use an abbreviated notation $\langle [b_1], \ldots, [b_m] \rangle \leftarrow \mathsf{PreMul}([a], m)$ when all a_i's are equal. In the secret sharing setting, this operation saves the number of rounds (with garbled circuit, the number of rounds is not the main concern and multiple multiplications can be used instead of this operation). The most efficient constant-round implementation of PreMul for integers is available from [45] that takes only 2 rounds and $3m-1$ interactive operations. The solution, however, is limited to non-zero integers. We are interested in prefix multiplication over fixed-point values and suggest a tree-based solution consisting of fixed-point multiplications, similar to the way minimum/maximum protocols are constructed. This requires $(m-1)(2k+2)$ interactive operations in $2\lceil \log m \rceil + 2$ rounds.

- *Compaction* $\langle [b_1], \ldots, [b_m] \rangle \leftarrow \mathsf{Comp}([a_1], \ldots, [a_m])$ pushes all non-zero elements of its input to appear before any zero element of the set. We are interested in order-preserving compaction that also preserves the order of the non-zero elements in the input set. A solution from [35] (based on data-oblivious order-preserving compaction in [67]) can work in both garbled circuit and secret sharing settings using any type of input data. In this work, we are interested in the variant of compaction that takes a set of tuples $\langle a'_i, a''_i \rangle$ as its input, where each a'_i is a bit that indicates whether the data item a''_i is zero or not (i.e., comparison of each data item to 0 is not needed). The complexities of this variant are given in Tables 3.1 and 3.2.

- *Array access at a private index* allows to read or write an array element at a private location. In this work we utilize only read accesses and denote the operation as a table lookup $[b] \leftarrow \mathsf{TLookup}\ (\langle [a_1], \ldots, [a_m] \rangle, [ind])$. The array elements a_i might be protected or publically known, but the index is always private. Typical straightforward implementations of this operations include a multiplexer (as in, e.g., [117]) or comparing the index $[ind]$ to all positions of the array and obliviously choosing one of them. Both implementations have complexity $O(m \log m)$ and work with garbled circuit and secret sharing techniques and data of different types. Based on our analysis and performance of components of this functionality, a multiplexer-based implementation outperforms the comparison-based implementation for garbled circuit, while the opposite is true for secret sharing based techniques. We thus report performance of the best option for secret sharing and garbled circuit settings in Tables 3.1 and 3.2.

  Each record a_i can be large, in which case the complexity of the operation additionally linearly grows with the size of array elements (or the number of field elements that each array stores in the secret sharing setting).

- *Oblivious sorting* $\langle [b_1], \ldots, [b_m] \rangle \leftarrow \mathsf{Sort}([a_1], \ldots, [a_m])$ obliviously sorts an m-element set. While several algorithms of complexity $O(m \log m)$ are known, in practice the most efficient oblivious sorting is often the Batcher's merge sort [23]. According to [30], the algorithm involves $\frac{1}{4}m(\log^2 m - \log m + 4)$ compare-and-exchange operations that compare two elements and conditionally swap them.

The complexities of all building blocks are listed in Tables 3.1 and 3.2, and notation is explained with each respective protocol. All functions with the exception of Int2FP and FP2Int the associated integer and fixed-point variants, performance of which might differ in the secret sharing sharing. Because most protocols exhibit the same performance for integer and fixed-point variants, for the functions with different performance, we list both variants (integer followed by fixed-point) separated by ":".

TABLE 3.1

PERFORMANCE OF KNOWN SECURE BUILDING BLOCKS ON INTEGER AND FIXED-POINT VALUES IN SECRET SHARING

Protocol	Rounds	Interactive operations
Add/Sub	0	0
LT	4	$4\ell - 2$
EQ	4	$\ell + 4\log \ell$
Mul	$1 : 4$	$1 : 2k + 2$
Div	$- : 3\log \ell + 2\xi + 12$	$- : 1.5\ell \log \ell + 2\ell\xi + 10.5\ell + 4\xi + 6$
PreMul	$2 : 2\log m + 2$	$3m - 1 : (m-1)(2k+2)$
Max/Min	$4\log m + 1$	$4\ell(m-1)$
Int2FP	0	0
FP2Int	3	$2k + 1$
Comp	$\log m + \log\log m + 3$	$m\log m \log\log m + 4m\log m - m + \log m + 2$
Sort	$2\log m(\log m + 1) + 1$	$\ell(m - 0.25)(\log^2 m + \log m + 4)$
TLookup	5	$m\log m + 4m\log\log m + m$

TABLE 3.2

PERFORMANCE OF KNOWN SECURE BUILDING BLOCKS ON INTEGER AND FIXED-POINT VALUES IN GARBLED CIRCUIT

Protocol	XOR gates	Non-XOR gates
Add/Sub	4ℓ	ℓ
LT	3ℓ	ℓ
EQ	ℓ	ℓ
Mul	$4\ell^2 - 4\ell$	$2\ell^2 - \ell$
Div	$7\ell^2 + 7\ell$	$3\ell^2 + 3\ell$
Max/Min	$5\ell(m-1)$	$2\ell(m-1)$
Int2FP	0	0
FP2Int	0	0
Comp	$(\ell+4)m\log m$ $-m\ell - 4\log m + \ell$	$(2\ell+1)m\log m - 2\ell m$ $+(\ell-1)\log m + 2\ell$
Sort	$1.5m\ell(\log^2 m + \log m + 4)$	$0.5m\ell(\log^2 m + \log m + 4)$
TLookup	$m\ell + \log m - \ell$	$m\log m + m(\ell-1)$

3.2.2 Floating-Point Building Blocks

For floating-point operations, we adopt the same floating-point representation as in [11]. Namely, a real number x is represented as 4-tuple $\langle v, p, s, z\rangle$, where v is an ℓ-bit normalized significand (i.e., the most significant bit of v is 1), p is a k-bit (signed) exponent, z is a bit that indicates whether the value is zero, and s is a bit set only when the value is negative. We obtain that $x = (1 - 2s)(1 - z)v \cdot 2^p$. As in [11], when $x = 0$, we maintain that $z = 1$, $v = 0$, and $p = 0$.

The work [11] provides a number of secure floating-point protocols, some of which we use in our solution as floating-point building blocks. While the techniques of [11] also provide the capability to detect and report errors (e.g., in case of division by 0, overflow or underflow, etc.), for simplicity of presentation, we omit error handling in this work. The building blocks from [11] that we use here are:

- *Multiplication* $\langle [v], [p], [z], [s] \rangle \leftarrow \mathsf{FLMul}(\langle [v_1], [p_1], [z_1], [s_1] \rangle, \langle [v_2], [p_2], [z_2], [s_2] \rangle)$ performs floating-point multiplication of its two real valued arguments.
- *Division* $\langle [v], [p], [z], [s] \rangle \leftarrow \mathsf{FLDiv}(\langle [v_1], [p_1], [z_1], [s_1] \rangle, \langle [v_2], [p_2], [z_2], [s_2] \rangle)$ allows the parties to perform floating-point division using $\langle [v_1], [p_1], [z_1], [s_1] \rangle$ as the dividend and $\langle [v_2], [p_2], [z_2], [s_2] \rangle$ as the divisor.
- *Addition* $\langle [v], [p], [z], [s] \rangle \leftarrow \mathsf{FLAdd}(\langle [v_1], [p_1], [z_1], [s_1] \rangle, \langle [v_2], [p_2], [z_2], [s_2] \rangle)$ performs the computation of addition (or subtraction) of two floating-point arguments.
- *Comparison* $[b] \leftarrow \mathsf{FLLT}(\langle [v_1], [p_1], [z_1], [s_1] \rangle, \langle [v_2], [p_2], [z_2], [s_2] \rangle)$ produces a bit, which is set to 1 iff the first floating-point argument is less than the second argument.
- *Exponentiation* $\langle [v], [p], [z], [s] \rangle \leftarrow \mathsf{FLExp2}(\langle [v_1], [p_1], [z_1], [s_1] \rangle)$ computes the floating-point representation of exponentiation $[2^x]$, where $[x] = (1 - 2[s_1])(1 - [z_1])[v_1]2^{[p_1]}$.

These protocols were given in [11] only for secret sharing, but we also evaluate their performance in homomorphic encryption using the most efficient currently available integer building blocks (as specified in [11]). The complexities of the resulting floating-point protocols in secret sharing and homomorphic encryption can be found in Tables 3.3 and 3.4 respectively.

3.3 Signature and Commitment Schemes

Here, we introduce additional building blocks, which are signature schemes with protocols and commitment schemes.

TABLE 3.3

COMPLEXITY OF FLOATING-POINT PROTOCOLS IN SECRET SHARING

Prot.	Rounds	Interactive operations
FLMul	11	$8\ell + 10$
FLDiv	$2\log\ell + 7$	$2\log\ell(\ell+2) + 3\ell + 8$
FLAdd	$\log\ell + \log\log\ell + 27$	$14\ell + 9k + (\log\ell)\log\log\ell + (\ell+9)\log\ell + 4\log k + 37$
FLLT	6	$4\ell + 5k + 4\log k + 13$
FLExp2	$6\log\ell + \log\log\ell + 31$	$4\ell^2 + 23\ell + 3\ell\log\ell + (\log\ell)\log\log\ell + 6\log\ell + 16k + 1$

From the available signature schemes, e.g., [37, 38] with the ability to prove knowledge of a signature on a message without revealing the message, the Camenisch-Lysyanskaya scheme [37] is of interest to us. It uses public keys of the form (n, a, b, c), where n is an RSA modulus and a, b, c are random quadratic residues in $\mathbb{Z}_n^*$. A signature on message m is a tuple (e, s, v), where e is prime, e and s are randomly chosen according to security parameters, and v is computed to satisfy $v^e \equiv a^m b^s c \pmod{n}$. A signature can be issued on a block of messages. To sign a block of t messages m_1, $\ldots$, m_t, the public key needs to be of the form $(n, a_1, \ldots, a_t, b, c)$ and the signature is (e, s, v), where $v^e \equiv a_1^{m_1} \cdots a_t^{m_t} b^s c \pmod{n}$.

Given a public verification key (n, a, b, c), to prove knowledge of a signature $(e, s$ $, v)$ on a secret message m, one forms a commitment $c = \mathsf{Com}(m)$ and proves that she possesses a signature on the value committed in c (see [37] for detail). The commitment c can consecutively be used to prove additional statements about m in zero knowledge (is described in Section 3.4). Similarly, if one wants to prove statements about multiple messages included in a signature, multiple commitments will be formed.

TABLE 3.4

COMPLEXITY OF FLOATING-POINT PROTOCOLS IN HOMOMORPHIC ENCRYPTION

Prot.	Rounds	Communication size	Computation complexity	
			Client	Server
FLMul	13	$39C + 10D$	17	25
FLDiv	$2\log\ell + 8$	$4\log\ell(3C + D) + 18C + 4D$	$6\log\ell + 12$	$12\log\ell + 16$
FLAdd	$\log\ell + 45 + \log\log\ell$	$(\ell\log\ell + 14\log\ell + \log\log\ell \times \log\ell + 6\log k + 54)D + (15\ell + 3\ell\log\ell + 19\log\ell + 13\log k + 3\log\ell\log\log\ell + 155)C$	$18\ell + k + 2\ell \times \log\ell + 32\log\ell + 14\log k + 2\log\ell \times \log\log\ell + 125$	$21\ell + k + 3\ell \times \log\ell + 40\log\ell + 17\log k + 3\log\ell \times \log\log\ell + 144$
FLLT	10	$(6\log\ell + 20)D + (13\log\ell + 63)C$	$k + 14\log k + 41$	$k + 17\log k + 57$
FLExp2	$15\log\ell + 38$	$(10\ell + \ell\log\ell + 12\log\ell + \log\ell\log\log\ell + 35)D + (34\ell + 3\ell\log\ell + 26\log\ell + 3\log\ell\log\log\ell + 107)C$	$40\ell + 2\ell\log\ell + 28\log\ell + 2\log\ell\log\log\ell + 44$	$53\ell + 3\ell\log\ell + 3\log\ell\log\log\ell + 34\log\ell + 59$

The commitment scheme used in [37] is that of Damgård and Fujisaki [51]. The setup consists of a public key (n, g, h), where n is an RSA modulus, h is a random quadratic residue in $\mathbb{Z}_n^*$, and g is an element in the group generated by h. The modulus n can be the same as or different from the modulus used in the signature scheme. For simplicity, we assume that the same modulus is used. To produce a commitment to x using the key (n, g, h), one randomly chooses $r \in \mathbb{Z}_n$ and sets

$\mathsf{Com}(x, r) = g^x h^r \bmod n$. When the value of r is not essential, we may omit it and use $\mathsf{Com}(x)$ instead. This commitment scheme is statistically hiding and computationally binding. The values x, r are called the opening of $\mathsf{Com}(x, r)$.

3.4 Zero-Knowledge Proofs of Knowledge

Zero-knowledge proofs of knowledge (ZKPKs) allow one to prove a particular statement about private values without revealing additional information besides the statement itself. Following [40], we use notation $PK\{(vars) : statement\}$ to denote a ZKPK of the given statement, where the values appearing in the parentheses are private to the prover and the remaining values used in the statement are known to both the prover and verifier. If the proof is successful, the verifier is convinced of the statement of the proof. For example, $PK\{(\alpha) : y = g_1^{\alpha} \vee y = g_2^{\alpha}\}$ denotes that the prover knows the discrete logarithm of y to either the base g_1 or g_2.

In Chapter 4, we utilize four particular ZKPKs: a proof that a ciphertext encrypts a value in an specific range, a proof that a ciphertext encrypts one of the two given values, a proof of plaintext knowledge, and a proof of plaintext multiplication. Below we specify these ZKPKs more formally using the popular notation of [41], $\mathsf{ZKPK}\{(S, P) \colon R\}$, which states that the prover possesses set S as her secret values, the values in set P are known to both parties, and the prover proves statement R.

- $\mathsf{RangeProof}((x, \rho), (e, L, H)) = \mathsf{ZKPK}\{(x, \rho), (e, L, H) : (e = \mathsf{Enc}(x, \rho)) \wedge (L \leq x \leq H)\}$. The prover wishes to prove to the verifier that a ciphertext e encrypts a value x where $x \in [L, H]$. In Chapter 4, we use $\mathsf{RangeProof}(x, L, H, e)$ instead of $\mathsf{RangeProof}((x, \rho), (e, L, H))$ because we do not use parameter ρ in the solutions.
- $\mathsf{PK12}((a, \rho), (a', p_1, p_2)) = \mathsf{ZKPK}\{(a, \rho), (a', p_1, p_2) : (a' = \mathsf{Enc}(a, \rho)) \wedge ((a = p_1) \vee (a = p_2))\}$. Here, the prover wishes to prove to the verifier that $a' = \mathsf{Enc}(a, \rho)$ is an encryption of one of the two known plaintexts p_1 and p_2.
- $\mathsf{PKP}((a, \rho), (a')) = \mathsf{ZKPK}\{(a, \rho), (a') : (a' = \mathsf{Enc}(a, \rho))\}$. The prover wishes to prove to the verifier that he knows the value a that the ciphertext a' encrypts (and thus that a' is a valid ciphertext).

- $\mathsf{PKPM}((b, \rho_b), (a', b', c')) = \mathsf{ZKPK}\{(b, \rho_b), (a', b', c') : (b' = \mathsf{Enc}(b, \rho_b)) \wedge (a' = \mathsf{Enc}(a)) \wedge (c' = \mathsf{Enc}(c)) \wedge (c = ab)\}$. The prover wishes to prove to the verifier that c' encrypts the product of the corresponding plaintexts of a' and b', where the prover knows the plaintext of b' (i.e., this is multiplication of an encrypted value by a known plaintext value).

For additional information (such as the appropriate choice of parameters), we refer the reader to [50, 52]. Also, complexity of the above ZKPKs based on homomorphic encryption can be found in Table 3.5.

TABLE 3.5

COMPLEXITY OF ZKPKS IN HOMOMORPHIC ENCRYPTION

Protocol	Rounds	Communication size	Computation complexity	
			Client	Server
RangeProof	1	$6 \log(H-L)C$	$5 \log(H-L)$	$6 \log(H-L)$
PK12	1	$4C$	3	2
PKPK	1	$2.5C$	2	2
PKPM	1	$4.5C$	4	4

Also, in Chapter 5 we use abbreviation $\mathsf{Sig}(x)$ and $\mathsf{Com}(x)$ to indicate the knowledge of a signature and commitment, respectively. For example, $PK\{(\alpha) : \mathsf{Sig}(\alpha) \wedge y = \mathsf{Com}(\alpha) \wedge (\alpha = 0 \vee \alpha = 1)\}$ denotes a proof of knowledge of a signature on a bit committed to in y. Because proving the knowledge of a signature on x in [37] requires a commitment to x (which is either computed as part of the proof or may already be available from prior computation), we explicitly include the commitment into all proofs of a signature.

3.5 Security Model

Security of any multi-party protocol (with two or more participants) can be formally shown according to one of the two standard security definitions (see, e.g., [65]) based on hybrid model. The first, weaker security model assumes that the participants are semi-honest (also known as honest-but-curious or passive), defined as they follow the computation as prescribed, but might attempt to learn additional information about the data from the intermediate results. The second, stronger security model allows dishonest participants to arbitrarily deviate from the prescribed computation. The definition of security in the semi-honest model is given in the following.

Definition 1 *Let parties $P_1, \ldots, P_{np}$ engage in a protocol Π that computes a (possibly probabilistic) np-ary function $f : (\{0,1\}^*)^{np} \to (\{0,1\}^*)^{np}$, where P_i contributes input in_i and receives output out_i. Let $\mathrm{VIEW}_\Pi(P_i)$ denote the view of participant P_i during the execution of protocol Π. More precisely, P_i's view is formed by its input and internal random coin tosses r_i, as well as messages $m_1, \ldots, m_k$ passed between the parties during protocol execution: $\mathrm{VIEW}_\Pi(P_i) = (\mathsf{in}_i, r_i, m_1, \ldots, m_k)$. Let $I = \{P_{i_1}, P_{i_2}, \ldots, P_{i_t}\}$ denote a subset of the participants for $t < np$ and $\mathrm{VIEW}_\Pi(I)$ denote the combined view of participants in I during the execution of protocol Π (i.e., $\mathrm{VIEW}_\Pi = (\mathrm{VIEW}_\Pi(P_{i_1}, \ldots, \mathrm{VIEW}_\Pi(P_{i_t})))$) and $f_I(\mathsf{in}_1, \ldots, \mathsf{in}_{np})$ denote the projection of $f(\mathsf{in}_1, \ldots, \mathsf{in}_{np})$ on the coordinates in I (i.e., $f_I(\mathsf{in}_1, \ldots, \mathsf{in}_{np})$ consists of the i_1th, $\ldots$, i_tth elements that $f(\mathsf{in}_1, \ldots, \mathsf{in}_{np})$ outputs). We say that protocol Π is t-private in the presence of semi-honest adversaries if for each coalition I of size at most t and all $\mathsf{in}_i \in \{0,1\}^*$ there exists a probabilistic polynomial time simulator S_I such that $\{S_I(\mathsf{in}_I, f_I(\mathsf{in}_1, \ldots, \mathsf{in}_{np})), f(\mathsf{in}_1, \ldots, \mathsf{in}_{np})\} \equiv \{\mathrm{VIEW}_\Pi(I), (\mathsf{out}_1, \ldots, \mathsf{out}_{np})\}$, where $\mathsf{in}_I = (\mathsf{in}_1, \ldots, \mathsf{in}_t)$ and "$\equiv$" denotes computational or statistical indistinguishability.*

In the two-party setting, we have that $np = 2$, $t = 1$. The participants' inputs in_1, in_2 and outputs out_1, out_2 are set as described above. In the multi-party setting, $np > 2$, $t < np/2$, and the computational parties are assumed to contribute no input and receive no output (to ensure that they can be disjoint from the input and output parties). Then the input parties secret-share their inputs among the computational parties prior the protocol execution takes place and the output parties receive shares of the output and reconstruct the result after the protocol termination. This setting then implies that, in order to comply with the above security definition, the computation used in protocol Π must be data-oblivious, which is defined as requiring the sequence of operations and memory accesses used in Π to be independent of the input.

Security of a protocol in the malicious model is shown according to the ideal/real simulation paradigm. In the ideal execution of the protocol, there is a *trusted third party* (TTP) that evaluates the function on participants' inputs. The goal is to build a simulator S who can interact with the TTP and the malicious party and construct a protocol's view for the malicious party. A protocol is secure in the malicious model if the view of the malicious participants in the ideal world is computationally indistinguishable from their view in the real world where there is no TTP. Also the honest parties in both worlds receive the desired output. This gives us the following definition of security in the malicious model.

Definition 2 *Let Π be a protocol that computes function $f : (\{0,1\}^*)^{np} \rightarrow (\{0,1\}^*)^{np}$, with party P_i contributing input in_i. Let $\mathcal{A}$ be an arbitrary algorithm with auxiliary input x and S be an adversary/simulator in the ideal model. Let $REAL_{\Pi,\mathcal{A}(x),I}(\mathsf{in}_1, \ldots, \mathsf{in}_{np})$ denote the view of adversary $\mathcal{A}$ controlling parties in I together with the honest parties' outputs after real protocol Π execution. Similarly, let $IDEAL_{f,S(x),I}(\mathsf{in}_1, \ldots, \mathsf{in}_{np})$ denote the view of S and outputs of honest parties after ideal execution of function f. We say that Π t-securely computes f if for each coalition I of size at most t,* *every probabilistic* $\mathcal{A}$ *in the real model, all* $\mathsf{in}_i \in \{0,1\}^*$ *and* $x \in \{0,1\}^*$*, there is probabilistic* S *in the ideal model that runs in time polynomial in* $\mathcal{A}$*'s runtime and* $\{\mathrm{IDEAL}_{f,S(x),I}(\mathsf{in}_1,\ldots,\mathsf{in}_{np})\} \equiv \{\mathrm{REAL}_{\Pi,\mathcal{A}(x),I}(\mathsf{in}_1,\ldots,\mathsf{in}_{np})\}$.

CHAPTER 4

VOICE RECOGNITIONS

Hidden Markov Model is a popular statistical tool with a large number of applications in pattern recognition. In some of these applications, such as speaker recognition, the computation involves personal data that can identify individuals and must be protected. We thus treat the problem of designing privacy-preserving techniques for HMM and companion Gaussian mixture model (GMM) computation suitable for use in speaker recognition and other applications. We provide secure solutions for both two-party and multi-party computation models and both semi-honest and malicious settings. In the two-party setting, the server does not have access in the clear to either the user-based HMM or user input (i.e., current observations) and thus the computation is based on threshold homomorphic encryption, while the multi-party setting uses threshold linear secret sharing as the underlying data protection mechanism. All solutions use floating-point arithmetic, which allows us to achieve high accuracy and provable security guarantees, while maintaining reasonable performance.

The rest of this chapter is organized as follows: We first talk about the motivation of this work and our main contributions in Sections 4.1 and 7.1. Then, we provide background information regarding HMMs and GMMs in Section 4.3. In Section 4.4, we describe the security model. We then describe our overall solution in the semi-honest model in Section 4.5. Section 4.5 reports on the results of our implementation, and in Section 4.6 we present new techniques to enable secure execution of our solution in the malicious setting.

4.1 Motivation

Hidden Markov Models (HMMs) have been an invaluable and widely used tool in the area of pattern recognition. They have applications in bioinformatics, credit card fraud detection, intrusion detection, communication networks, machine translation, cryptanalysis, robotics, and many other areas. An HMM is a powerful statistical tool for modeling sequences that can be characterized by an underlying Markov process with unobserved (or hidden) states, but visible outcomes. One important application of HMMs is voice recognition, which includes both speech and speaker recognition. For both, HMMs are the most common and accurate approach, and we use this application as a running example that guides the computation and security model for this work.

When an HMM is used for the purpose of speaker recognition, usually one party supplies a voice sample and the other party holds a description of an HMM that represents how a particular individual speaks and processes the voice sample using its model and the corresponding HMM algorithms. Security issues arise in this context because one's voice sample and HMMs are valuable personal information that must be protected. In particular, a server that stores hidden Markov models for users is in possession of sensitive biometric data, which, once leaked to insiders or outsiders, can be used to impersonate the users. For that reason, it is desirable to minimize exposure of voice samples and HMMs corresponding to individuals when such data are being used for authentication or other purposes. To this end, we design solutions for securely performing computation on HMMs in such a way that no information about private data is revealed as a result of execution other than the agreed upon output. This immediately implies privacy-preserving techniques for speaker recognition as well as other applications of HMMs.

In more detail, in the speaker recognition application the overall process consists of two phases: (i) feature extraction in the form of creating an HMM and (ii) speaker authentication in the form of evaluating an HMM. The same two phases would need to be executed in other applications as well. Feature extraction constitutes a one-time enrollment process, during which information about how user $\mathcal{U}$ speaks is extracted and privately stored at a server or servers that will later authenticate the user (i.e., the HMM is not available to the servers in the clear and prevents leakage of user information and consequently user impersonation by unauthorized parties). At the time of user authentication, an individual $\mathcal{U}'$ wanting to gain access to the system as user $\mathcal{U}$ engages in privacy-preserving user authentication by evaluating a voice sample that $\mathcal{U}'$ supplies on $\mathcal{U}$'s HMM that the server stores. This takes the form of a secure protocol run between the client $\mathcal{U}'$ and the server. Note that user authentication can never be performed locally by the client because $\mathcal{U}'$ can always return the desired value as the final outcome to the server.

4.2 Contributions

There are three different types of problems and corresponding algorithms for HMM computation: the Forward algorithm, the Viterbi algorithm, and the Expectation Maximization (EM) algorithm. Because the Viterbi algorithm is most commonly used in voice recognition, we provide a privacy-preserving solution for that algorithm, but the techniques can be used to securely execute other HMM algorithms as well. Furthermore, to ensure that Gaussian mixture models (GMMs), which are commonly used in HMM computation, can be a part of secure computation as well, we integrate GMM computation in our privacy-preserving solutions.

One significant difference between our and prior work on secure HMM computation is that we develop techniques for computation on floating-point numbers, which provide adequate precision and are most appropriate for HMM computation. We also do not compromise on security, and all of the techniques we develop are provably secure under standard and rigorous security models, while at the same time providing reasonable performance (we implement the techniques and experimentally show performance in the semi-honest setting).

To cover as wide of a range of application scenarios as possible, we consider multiple settings: (i) the two-party setting in which a client interacts with a server and (ii) the multi-party setting in which the computation is carried out by $np > 2$ parties, which is suitable for collaborative computation by several participants as well as secure outsourcing of HMM computation to multiple servers by one or more computationally limited clients. In the two-party setting, the server should have no access in the clear to either the user-based (private) HMM or user input (i.e., current observations) and thus the server stores the encrypted HMM and computation proceeds on encrypted data (see Section 4.4.1 for justification of this setup). In the multi-party setting, on the other hand, threshold linear secret sharing is employed as the underlying mechanism for privacy-preserving computation.

We provide techniques for both semi-honest and malicious security models using secure floating-point operations from [11]. Because [11] treats only the semi-honest setting, equivalent solution secure in the stronger malicious model are not available for the two-party case based on homomorphic encryption. We thus develop necessary protocols to support general secure floating-point operations in the two-party computation setting based on homomorphic encryption.

To summarize, our contributions consist of developing provably secure HMM and GMM computation techniques based on Viterbi algorithm using floating-point arithmetic. Our techniques are suitable for homomorphic encryption and secret sharing computations in a variety of settings and are designed with their efficiency in mind, which we evaluate through experimental results of an implementation.

4.3 Hidden Markov Models and Gaussian Mixture Models

A Hidden Markov Model (HMM) is a statistical model that follows the Markov property with hidden states, but visible outcomes. The inputs are a sequence of observations, and for each sequence of observations (or outcomes), the computation consists of determining a path of state transitions which is the likeliest among all paths that could produce the given observations. An HMM consists of: 1) N states, 2) M possible outcomes, 3) a vector $\pi = \langle \pi_1, \ldots, \pi_N \rangle$ that contains the initial state probability distribution, 4) a matrix A of size $N \times N$ that contains state transition probabilities, 5) a matrix B of size $N \times M$ that contains output probabilities. In most cases, matrix B is computed based on observations $(X_1, \ldots, X_T)$. Therefore, we refer to the elements of B chosen based on, or computed from, the current observations as $N \times T$ matrix β. More information can be found in [9].

GMMs are mixtures of Gaussian distributions that represent the overall distribution of observations. In the case of HMMs, we use a GMM to compute the output probability of state number j (S_j) producing an observation at time k as follows:

$$\beta_{jk} = \sum_{i=1}^{\alpha} w_i e^{-\frac{1}{2}(X_k-\mu_i)^T \Sigma_i^{-1}(X_k-\mu_i)} \tag{4.1}$$

where X_k is a vector of size f that represents the random variable corresponding to the observation at time k. The parameter α is the total number of mixture components. μ_i is a mean vector of ith component with size f; also, ith component has a covariance matrix Σ_i of size $f \times f$. The components are added together, each weighted by a mixture weight w_i, to produce the probability distribution of state S_j when the observed random variable is X_k. We use notation μ, Σ, and w to refer to the sequence of μ_i, Σ_i, and w_i, respectively, for $i = 1, \ldots, \alpha$.

In this work, we focus on the Viterbi algorithm (with complexity of $O(TN^2)$) used in speaker recognition. In Algorithm 1, we provide a brief description of HMM

```
Algorithm 1: HMM algorithm
  Input: $N, T, \pi, A, \alpha, w, \mu, \Sigma$, and $X$
  Output: $P^*$ that is the probability of the most likely path for a given sequence
    of observations and $q^* = \langle q_1^*, \ldots, q_T^* \rangle$ that denotes the most likely path itself.
    1. For $j = 1$ to $N$ and $k = 1$ to $T$, compute $\beta_{jk}$ as in Equation 4.1 using $\alpha$, $w_i$'s,
       $\mu_i$'s, $\Sigma_i$'s, and $X_k$.
    2. Set $\lambda = \langle N, T, \pi, A, \beta \rangle$.
       (a) Initialization Step: for $i = 1$ to $N$ do
           i. $\delta_1(i) = \pi_i \beta_{i1}$
           ii. $\psi_1(i) = 0$
       (b) Recursion Step: for $k = 2$ to $T$ and $j = 1$ to $N$ do
           i. $\delta_k(j) = \left( \max_{1 \le i \le N} [\delta_{k-1}(i) a_{ij}] \right) \beta_{jk}$
           ii. $\psi_k(j) = \arg\max_{1 \le i \le N} [\delta_{k-1}(i) a_{ij}]$
       (c) Termination Step:
           i. $P^* = \max_{1 \le i \le N} [\delta_T(i)]$
           ii. $q_T^* = \arg\max_{1 \le i \le N} \delta_T(i)$
           iii. for $k = T - 1$ to 1 do $q_k^* = \psi_{k+1}(q_{k+1}^*)$
    3. Return $\langle P^*, q^* \rangle$
```

computation by computing the Viterbi algorithm (step 2). The computation uses dynamic programming to store intermediate probabilities in δ, after which the path of the maximum likelihood is computed and placed in q^*. For additional information, we refer the reader to [9].

4.4 Framework

In this section, we introduce two categories of secure computation that we consider in this work (two-party and multi-party), and precisely define the computation to be carried out.

4.4.1 Two-Party Computation

The first category of secure computation that we consider is secure two-party computation. Without loss of generality, we will refer to the participants as the client and the server. Using speaker recognition as the example application, the setting can be described as follows: the client possesses a voice sample, the server stores a model that represents how a registered user speaks, and user authentication is performed by conducting HMM computation on the client's and server's inputs. Therefore, for the purposes of this work, we assume that the client owns the observations to an HMM, i.e., $X_1, \ldots, X_T$, and the server holds the parameters of the HMM and GMM, i.e., N, vector π, matrix A, α, mixture weights w, vectors μ, and matrices Σ. Because even the parameters of HMM might reveal information about the possible input observations, to build a fully privacy-preserving solution in which the server does not learn information about user biometrics, the server should not have access to the HMM parameters in the clear. For that reason, we assume that the server holds the parameters π, A, B, w, μ, and Σ in an encrypted form and computation proceeds on encrypted data. While there are other underlying techniques for secure two-party computation (such as garbled circuit evaluation), we view storing HMM data encrypted at the server and evaluating the function on encrypted data as the best option, despite high computational overhead associated with this approach. If encryption is not used, the HMM values will need to be split into random shares, with one share of each value stored by the client and the other share stored by the server. This creates multiple issues, one of which is that the client's state is large and the shares of the HMM must be present on each device from which the client wants to authenticate. The second issue is that a malicious user will need to be forced to enter the original HMM data into each authentication session to avoid tampering with the authentication process, which is generally not known how to do.

To permit the computation to take place on encrypted data, we resort to an encryption scheme with special properties, namely, semantically secure additively homomorphic public-key encryption scheme. Furthermore, to ensure that neither the server can decrypt the data it stores, nor the (untrusted) client can decrypt the data (or a function thereof) without the server's consent, we utilize a (2, 2)-threshold encryption scheme. Informally, it means that the decryption key is partitioned between the client and the server, and each decryption requires that both of them participate. This means that the client and the server can jointly carry out the HMM computation and make the result available to either or both of them. For concreteness of exposition, we will assume that the server learns the outcome.

We obtain that in the two-party setting, the client and the server share the decryption key to a semantically secure additively homomorphic (2, 2)-threshold public-key encryption scheme. The client has private input $X_1, \ldots, X_T$ and its share of the decryption key sk; the server has input $\mathsf{Enc}_{pk}(\pi_i)$ for $i \in [1, N]$, $\mathsf{Enc}_{pk}(a_{ij})$ for $i \in [1, N]$ and $j \in [1, N]$, $\mathsf{Enc}_{pk}(w_i)$ for $i \in [1, \alpha]$, encryption of each element of μ_i and Σ_i for $i \in [1, \alpha]$, and its share of sk. The computation consists of executing the Viterbi algorithm on their inputs, at the end of which the server learns P^* and q_i^* for $i = 1, \ldots, T$. The size of the problem, i.e., parameters N, T, α, and f, are assumed to be known to both parties.

4.4.2 Multi-Party Computation

The second category of secure computation that we consider is secure multi-party computation on HMMs. In this setting, either a number of parties hold inputs to a multi-observer HMM or one or more clients wish to outsource HMM computations to a collection of servers. More generally, we divide all participants into three groups: (i) the input parties who collectively possess the private inputs, (ii) the computational parties who carry out the computation, and (iii) the output parties who receive the result(s) of the computation. These groups can be arbitrarily overlapping, which gives great flexibility in the setup and covers all possible cases of joint multi-party computation (where the input owners carry out the computation themselves, select a subset of them, or seek help of external computational parties) and outsourcing scenarios (by either a single party or multiple input owners).

To conduct computation on protected values in this setting, we utilize an information-theoretically secure threshold linear secret sharing scheme (such as Shamir secret sharing scheme [110]).

We then obtain that in this setting the input parties share their private inputs among $np > 2$ computational parties, the computational parties execute the Viterbi algorithm on secret-shared values, and communicate shares of the result to the output parties, who reconstruct the result from their shares. As before, the size of the problem – namely, the parameters N, T, α, and f – is known to all parties.

4.5 Secure HMM and GMM Computation in the Semi-Honest Model

We use privacy-preserving solution for HMM and GMM computation based on Viterbi algorithm using floating-point numbers from [9]. For this security solutoion, we use floating-point building blocks in Section 3.2.2. We recommend to read Chapter 5 of [9] for more information about the computations.

The secure solution has the same asymptotic complexity as the original algorithm, expressed as a function of parameters N, T, α, and f. In particular, the GMM computation involves $O(\alpha f^2 NT)$ floating-point operations and the Viterbi algorithm (step 2 of Algorithm 1) itself uses $O(N^2T)$ floating-point operations (the recursion step dominates the complexity of the overall algorithm). In the two-party setting, the solution that employs homomorphic encryption additionally has dependency on the computational security parameter κ and the bitlength representation of the underlying floating-point values, while in the multi-party setting based on secret sharing, the complexity has dependency only on the bitlength representation of the floating-point values. More precisely, using the complexities of the building blocks as specified in Chapter 3 and [10], we obtain that the GMM computation in the homomorphic encryption setting involves $O(\alpha f^2 NT(\ell \log \ell + k))$ modulo exponentiations (which depend on the security parameter κ) and communicates $O(\alpha f^2 NT(\ell \log \ell + \log k))$ ciphertexts and/or decryption shares (which likewise depend on the security parameter κ). In the multi-party setting based on secret sharing, the complexity becomes $O(\alpha NT(f^2 \ell \log \ell + f^2 k + \ell^2))$ interactive operations (which depend on the number of participating parties n). The Viterbi computation involves $O(N^2 Tk)$ modulo exponentiations and communicates $O(N^2 T \log \ell)$ ciphertexts/decryption shares in the homomorphic encryption setting, and it uses $O(N^2 T(\ell + k))$ interactive operations in the secret sharing setting.

In the following, we report on the results of implementation of the HMM solution. Note that because of numerous applications of HMMs, their use in certain contexts might differ, and we therefore chose to implement only the core HMM computation without the GMM component. The output probabilities matrix β can be computed based on observations via different means (one of which is GMM) and for the purposes of our implementation we choose discrete assignment of β_{jk}'s based on the sequence of observations $X_1, \ldots, X_T$. In particular, for each $j = 1, \ldots, N$ and $k = 1, \ldots, T$, we set $\beta_{jk} = b_{j,i}$ using matrix B, where i corresponds to the index that the value X_k takes (out of M possible outcomes). In the homomorphic encryption setting, this means that the client who possesses the observations $X_1, \ldots, X_T$ receives encrypted matrix B from the server and sets $\mathsf{Enc}(\beta_{jk}) = \mathsf{Enc}(b_{j,i}) \cdot \mathsf{Enc}(0)$ according to X_k, where $\mathsf{Enc}(0)$ is used for randomization purposes. In the secret sharing case, the parties jointly hold X and B in protected form and obliviously set β_{jk} based on X_k (i.e., without knowing what cell of B was used to set each β_{jk}).

All our implementations in this thesis were built in C/C++. All machines used in the experiments had identical hardware with four-core 3.2GHz Intel i5-3470 pro-

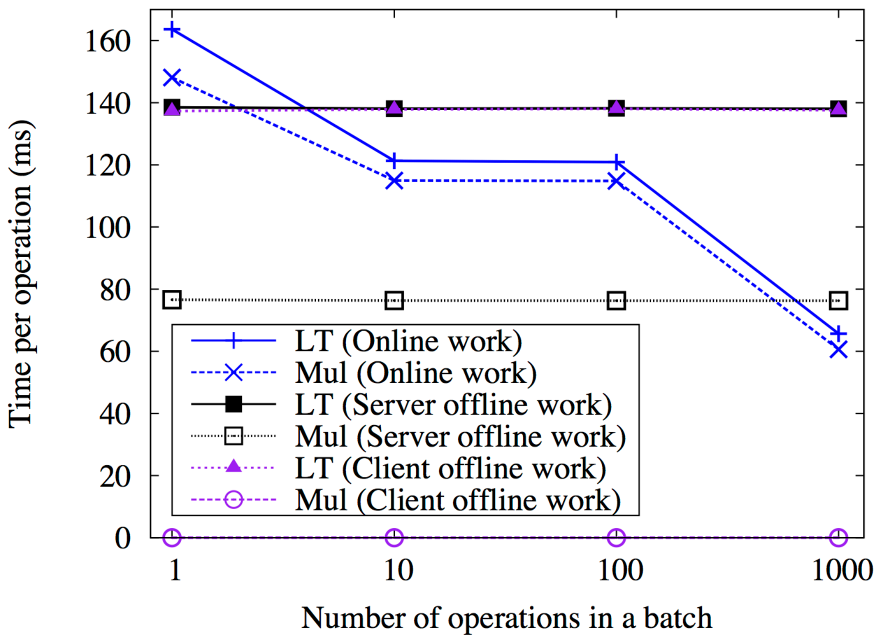


Figure 4.1. Performance of integer HMM's building blocks in the homomorphic encryption setting.

cessors with Red Hat Linux 2.6.32 and were connected via a 1Gb LAN. Only one core was used during the experiments (i.e., multi-threading was not used).

In what follows, we first describe our homomorphic encryption experiments followed by the experiments in the secret sharing setting. In our implementations we represent floating-point numbers using 32-bit significands and (signed) 9-bit exponents (plus sign and zero bits as described earlier in this work).

In the homomorphic encryption setting, we utilized $(2, 2)$-threshold Paillier encryption, which was implemented using Miracl library [47] for large number arithmetic. The experiments we report were conducted using a 1536-bit modulus for Paillier encryption. Because performance of our building blocks is not available in prior literature, we provide runtimes of integer and floating-point operations used in our implementation in Figures 4.1 and 4.2 and overall HMM computation in Fig-

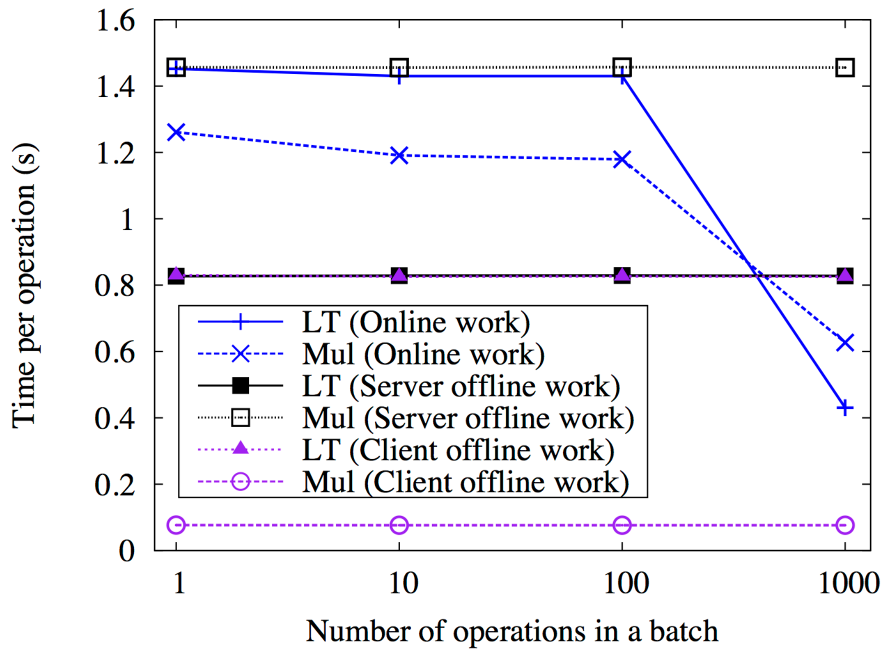


Figure 4.2. Performance of floating-point HMM's building blocks in the homomorphic encryption setting.

ure 4.3. The parameters N and T for HMM experiments were chosen as suggested in the speaker recognition literature [21, 93]. That is, a typical value for N is 3 and T is around 100 (using 32 ms per frame in [93]). We separately vary N and T to illustrate how performance depends on these values. All experiments were run 5 times and the mean value is given.

Because techniques based on homomorphic encryption are computationally intensive, we separate all work into offline and online, where the offline work consists of computation that can be performed before the inputs become available (e.g., generating random values and encrypting them). We thus measure offline work for client and server and the overall online runtime. In our experiments with identical machines, the server performs somewhat more work and thus takes longer, but in practice the server is expected to be a more powerful machine with client's performance being the

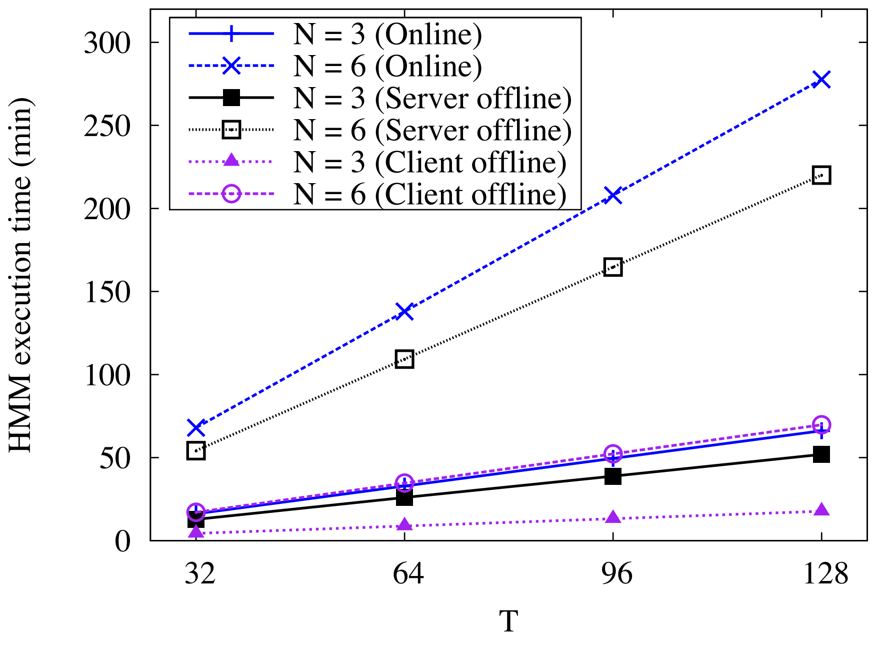


Figure 4.3. Performance of HMM computation for varying N and T in the homomorphic encryption setting.

bottleneck. For integer and floating-point operations we report the time per operation when a number of operations are executed in a single batch. Batch execution reduces communication overhead when simultaneous execution of a number of operations is possible (as is the case for HMMs). This results in reducing the total online time.

Figures 4.1 and 4.2 present performance of integer and floating-point operations in the homomorphic encryption setting as described above. The homomorphic encryption performance of HMM computation in Figure 4.3 is consistent with the complexity of the algorithm, which has linear dependency on T and quadratic dependency on N (i.e., the slope for $N = 3$ is different from the slope for $N = 6$). In both figures, most of offline work is done by the server, which benefits overall execution time.

In the secret sharing setting, we utilized three computational parties that operate on shares of the data formed using a (3,1)-threshold linear secret sharing scheme. The implementation was built using the PICCO compiler [117], in which an HMM

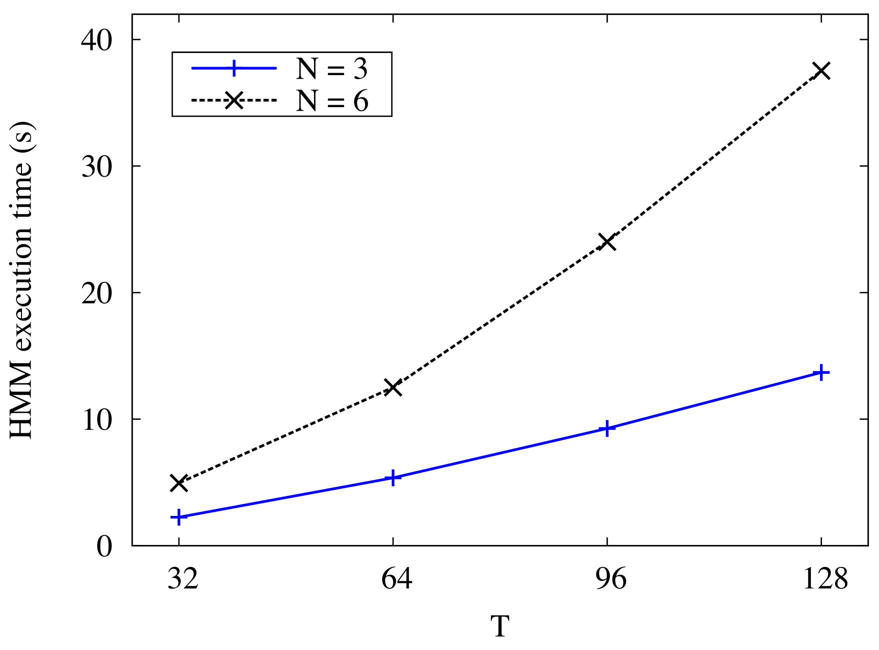


Figure 4.4. Performance of HMM computation for varying N and T in the secret sharing setting.

program written in an extension of C was compiled into its secure distributed implementation. PICCO is a source-to-source compiler that produces a C program and utilizes the GMP [4] library for the underlying arithmetic and OpenSSL [6] implementation of AES for protecting communication. All arithmetic was performed in field $\mathbb{F}_p$ for a 114-bit prime p (the modulus size was determined as described in [117]). The results of HMM experiments are given in Figure 4.4. While separation between online and offline work is also possible in this setting (e.g., the parties generate a large number of random values throughout the protocol execution), we do not distinguish between these types of work and list the overall performance in the online category. This is in part because we use an existing tool for our experiments and in part because the secret sharing performance is orders of magnitude faster than encryption-based two-party computation and is already practical.

In conclusion, we note that our two-party setting was constrained in terms of the tools that could be employed for secure computation. That is, in order to provably protect HMMs from the server, we have to resort to strong protection mechanisms such as homomorphic encryption, the threshold version of which was required to ensure that no individual party could independently decrypt ciphertexts and learn unauthorized information. In general, alternative techniques of better performance (such as garbled circuit) are possible and even additively homomorphic encryption can be substantially faster (see, e.g., [33]) when threshold decryption is not required. One suggestion for improving performance of the two-party setting for this application is to involve neutral third parties, which would allow for the use of multi-party techniques.

4.6 Secure HMM and GMM Computation in the Malicious Model

We next show how strengthen the solution to maintain security in the presence of malicious participants who can arbitrarily deviate from the prescribed behavior. The solution for the multi-party setting based on secret sharing, covered in Section 4.6.1. Most of the section is dedicated to the two-party setting based on homomorphic encryption, where in Section 4.6.2 we describe the necessary components for building a solution secure against malicious adversaries and in Sections 4.6.2.1–4.6.2.6 present protocols for two-party multiplication, comparison, truncation, inversion, prefix multiplication, and bit decomposition, respectively, together with their malicious-model security analysis.

4.6.1 Multi-Party Setting

The security of the solution in the secret sharing setting can be extended to the malicious security model. In that case, to show security in the presence of malicious adversaries, we need to ensure that (i) all participants prove that each step of their computation was performed correctly and that (ii) if some dishonest participants quit, others will be able to reconstruct their shares and proceed with the rest of the computation. The above is normally achieved using a verifiable secret sharing scheme (VSS), and a large number of results have been developed over the years. More information can be found in [9].

4.6.2 Two-Party Setting

To show security of our solution in the presence of malicious adversaries in the homomorphic encryption setting, we likewise need to enforce correct execution of all operations. However, unlike the secret sharing setting, this time security no longer follows from prior work and requires new tools.

To ensure that both participants follow all steps of the computation, we employ zero-knowledge proofs of knowledge from Section 3.4. Because such proofs are usually tied to the internal workings of the underlying homomorphic encryption scheme, we develop our solution based on the Paillier encryption scheme.

Our approach consists of using designed protocols secure in the presence of malicious adversaries for a number of building blocks used to build floating-point operations. Then after applying Canetti's composition theorem [44], we can guarantee security of larger building blocks and the overall solution. To determine which building blocks need to be implemented in the stronger security model with fully malicious participants, we analyze each floating-point operation used in this work.

- FLMul is implemented using protocols Trunc, LT, OR, XOR, and Mul as the building blocks. Trunc in turn depends on $\mathsf{TruncPR}$ and LT protocols[1]. OR and XOR protocols are built directly from Mul. This means that we need to realize malicious versions of multiplication Mul, truncation $\mathsf{TruncPR}$, and comparison LT.
- Besides some of the building blocks listed above, FLDiv additionally uses SDiv, which in turn is built using Mul and $\mathsf{TruncPR}$ protocols. Thus, no additional protocols are needed.

[1]Throughout this description we don't describe the functionality of each building block. Such description will be given only for the building blocks that we need to implement in the malicious model.

- FLAdd calls new building blocks EQ, $\mathsf{Pow2}$, BitDec, PreOR, and Inv. Our implementation of EQ is built on LT and Mul. $\mathsf{Pow2}$ calls BitDec and PreMul. PreOr calls PreMul and $\mathsf{Mod2}$, which is equivalent to Trunc.
- FLLT does not call any new building blocks.
- Similarly, FLExp calls only integer building blocks discussed above and FLMul that can be assembled from integer building blocks.

To summarize, we need to provide implementations of six functionalities secure in the malicious model, which are described next:

- $\mathsf{Enc}(xy) \leftarrow \mathsf{MalMul}(\mathsf{Enc}(x), \mathsf{Enc}(y))$ is a fundamental building block, which performs multiplication of its two encrypted input values x and y.
- $\mathsf{Enc}(b) \leftarrow \mathsf{MalLT}(\mathsf{Enc}(x), \mathsf{Enc}(y), \ell)$ performs comparison of two encrypted values x and y of size ℓ and outputs encrypted bit b, where $b = 1$ iff $x < y$.
- $\mathsf{Enc}(y) \leftarrow \mathsf{MalTruncPR}(\mathsf{Enc}(x), \ell, k)$ truncates k bits of encrypted x, which has bitlength ℓ. The output is probabilistic, where the least significant bit of the result y may differ from that of $\lfloor x/2^k \rfloor$.[2]
- $\mathsf{Enc}(y) \leftarrow \mathsf{MalInv}(\mathsf{Enc}(x))$ produces (encrypted) inversion $y = x^{-1}$ of encrypted x.
- $\mathsf{Enc}(y_1), ..., \mathsf{Enc}(y_k) \leftarrow \mathsf{MalPreMul}(\mathsf{Enc}(x_1), ..., \mathsf{Enc}(x_k))$ performs prefix multiplication of k non-zero encrypted values $x_1, \ldots, x_k$, where the result is computed as $y_i = \prod_{j=1}^{i} x_j$ for each $i \in [1, k]$.
- $\mathsf{Enc}(x_{k-1}), ..., \mathsf{Enc}(x_0) \leftarrow \mathsf{MalBitDec}(\mathsf{Enc}(a), \ell, k)$ extracts k least significant bits of (encrypted) x, where ℓ is the bitlength of x.

In the rest of this section we treat one protocol (two-party setting based on homomorphic encryption) at a time and report performance of each new protocol in the malicious model in Table 4.1.

4.6.2.1 Secure Multiplication

Now we describe a two-party multiplication protocol secure in the presence of a malicious participant. In this protocol, both parties hold $\mathsf{Enc}(x), \mathsf{Enc}(y)$ without any

[2]We note that such probabilistic version is sufficient in some cases, while in others the function can be changed to always produce correct truncation with the use of extra comparison.

knowledge of x or y and receive $\mathsf{Enc}(xy)$ as their output. The protocol is very similar to the one given in [53] for the secret sharing setting. The intuition is that P_1 and P_2 blind encryption of x with their respective random values r_1 and r_2 and decrypt $c = x + r_1 + r_2$. This allows them to compute $\mathsf{Enc}(y)^c$, from which they subtract encrypted yr_1 and yr_2 to recover the result. Doing so securely will require that P_1 and P_2 prove correctness of $\mathsf{Enc}(yr_1)$ and $\mathsf{Enc}(yr_2)$, respectively, using PKPM.

$\mathsf{Enc}(xy) \leftarrow \mathsf{MalMul}(\mathsf{Enc}(x), \mathsf{Enc}(y))$

Public inputs include public key pk for (2,2)-threshold Paillier encryption scheme and private inputs include shares of the corresponding secret key.

1. Each P_j chooses at random $r_j \in \mathbb{Z}_N$.
2. Each P_j computes $r'_j = \mathsf{Enc}(r_j, \rho_j)$, $z_j = \mathsf{Enc}(y)^{r_j} \cdot \mathsf{Enc}(0) = \mathsf{Enc}(y \cdot r_j)$, and executes $\mathsf{PKPM}((r_j, \rho_j), (\mathsf{Enc}(y), r'_j, z_j))$ to prove correctness of z_j to the other party.
3. Both parties locally compute $c' = \mathsf{Enc}(c) = \mathsf{Enc}(x) \cdot r'_1 \cdot r'_2$ and decrypt c' to recover c.
4. Each party computes $z = \mathsf{Enc}(y)^c = \mathsf{Enc}(yx + yr_1 + yr_2)$ and $\mathsf{Enc}(xy) = z \cdot z_1^{-1} \cdot z_2^{-1}$.

Following [50], we show security of this and other protocols in a hybrid model where decryption is replaced with ideal functionality. That is, instead of producing partial decryptions of a ciphertext and combining them to recover the corresponding plaintext, decryption is performed by submitting the ciphertext to a black-box which outputs the underlying plaintext. Then following the arguments of [50], we also obtain security in the real model when ideal decryption is instantiated with a real threshold decryption algorithm.

Theorem 1 *Assuming semantic security of the homomorphic encryption scheme and* *security of the building blocks, the above* MalMul *is secure in the malicious setting in the hybrid model with ideal decryption.*

Proof: We prove security of MalMul based on Definition 2. Because this protocol is completely symmetric, without loss of generality we assume P_1 is malicious. In that case, in the ideal world a simulator S_1 locates between P_1 and the TTP and simulates P_1's view after receiving $\mathsf{Enc}(x)$ and $\mathsf{Enc}(y)$, as well as $\mathsf{Enc}(xy)$ from the TTP. During step 2, S_1 receives r'_1 and z_1, and acts as a verifier for PKPM (extracting r_1). Then, S_1 sets $r'_2 = \mathsf{Enc}(w - r_1 - x)$ for a randomly chosen $w \in \mathbb{Z}_N$, computes $z_2 = (\mathsf{Enc}(xy))^{-1}(\mathsf{Enc}(y))^{w-r_1}$, and uses the simulator of PKPM to interact with P_1 and prove validity of r'_2, z_2. During step 3, S_1 sends w as the decrypted value c to P_1.

To show that P_1's output at the end of the simulation corresponds to correct $\mathsf{Enc}(xy)$, recall that the simulator sets $r_2 = w - r_1 - x$. Thus, $w = x + r_1 + r_2$ in step 3 is exactly what the simulator needs to decrypt. Therefore, P_1 correctly computes the output in step 4.

To show that the simulated view is indistinguishable from the view in the hybrid model (with ideal decryption), we note that the simulator produces encrypted values in step 2, which are indistinguishable from ciphertexts sent during the protocol execution because of semantic security of the encryption scheme. Similarly, the simulation of PKPM is indistinguishable from real execution due to its security properties. Lastly, the value w returned by the simulator in step 3 is distributed identically to the value c decrypted during protocol execution. We also note that both during the simulation and real execution the computation aborts only when a malicious party fails to correctly complete. □

4.6.2.2 Secure Comparison

Our comparison protocol secure in the malicious model is given next. In what follows, we use the fact that $(-1)^b = 1 - 2b$ and $(-1)^{1-b} = 2b - 1$ when b is a bit.

We also express a'_1 as $\mathsf{Enc}(1+2b_1b_2-b_1-b_2)$ and a'_2 as $\mathsf{Enc}(b_1+b_2-2b_1b_2)$.

$\mathsf{Enc}(b) \leftarrow \mathsf{MalLT}(\mathsf{Enc}(x), \mathsf{Enc}(y), \ell)$

Public inputs include public key pk for a (2,2)-threshold Paillier encryption scheme and private inputs consist of shares of the corresponding secret key.

1. Each P_j sets $e_1 = \mathsf{Enc}(1,0)$, $e_{-1} = \mathsf{Enc}(-1,0) = (e_1)^{-1}$, and $\mathsf{Enc}(c) = \mathsf{Enc}(x-y) = \mathsf{Enc}(x) \cdot \mathsf{Enc}(y)^{-1}$.

2. Each P_j chooses $b_j \in \{0,1\}$, $r_j, r'_j \in \{0,1\}^{\ell+\kappa}$ at random s.t. $r_j > r'_j$, and sends to P_{3-j} $z_{(j,1)} = \mathsf{Enc}(b_j, \rho_j)$, $z_{(j,2)} = \mathsf{Enc}(r_j, \rho'_j)$ and $z_{(j,3)} = \mathsf{Enc}(r'_j, \rho''_j)$. P_j executes $\mathsf{PK12}((b_j, \rho_j), (z_{(j,1)}, 0, 1))$, $\mathsf{RangeProof}(r_j, 0, H, z_{(j,2)})$, $\mathsf{RangeProof}(r'_j, 0, H, z_{(j,3)})$, and $\mathsf{RangeProof}(r_j - r'_j, 1, H, z_{(j,2)} \cdot z_{(j,3)}^{-1})$ to prove that $z_{(j,1)}$, $z_{(j,2)}$, and $z_{(j,3)}$ are well-formed, where $H = 2^{\ell+\kappa} - 1$.

3. Each party locally computes $z_{(j,4)} = \mathsf{Enc}(1-2b_j) = e_1 \cdot (z_{(j,1)})^{-2}$ and $z_{(j,5)} = \mathsf{Enc}(2b_j - 1) = (z_{(j,1)})^2 \cdot e_{-1}$.

4. P_1 computes $z_6 = \mathsf{Enc}((1-2b_1)c)) = \mathsf{Enc}(c)^{1-2b_1} \cdot \mathsf{Enc}(0, \alpha_1)$, where α_1 is newly selected as randomness during encryption, $z_7 = \mathsf{Enc}(r_1(1-2b_1)c) = z_6^{r_1}$, and $z_8 = \mathsf{Enc}((2b_1-1)r'_1) = z_{(1,5)}^{r'_1}$, and sends to P_2 z_6, z_7, and z_8. P_1 also executes $\mathsf{PKPM}((1-2b_1 \bmod N, \rho_1^{-2}), (\mathsf{Enc}(c), z_{(1,4)}, z_6))$, $\mathsf{PKPM}((r_1, \rho'_1 + \alpha_1), (z_6, z_{(1,2)}, z_7))$, and $\mathsf{PKPM}((r'_1, \rho''_1), (z_{(1,5)}, z_{(1,3)}, z_8))$ to prove that z_6, z_7, and z_8 are well-formed.

5. Each party locally computes $a_3 = \mathsf{Enc}(r_1(1-2b_1)c) + (2b_1-1)r'_1) = z_7 \cdot z_8$.

6. P_2 computes $z'_6 = \mathsf{Enc}(b_1b_2) = z_{(1,1)}^{b_2} \cdot \mathsf{Enc}(0, \alpha_2)$, $z'_7 = a_3^{1-2b_2} \cdot \mathsf{Enc}(0, \alpha_3)$, where α_2 and α_3 are newly selected as randomness during encryption, $z'_8 = (z'_7)^{r_2}$, and $z'_9 = z_{(2,5)}^{r'_2}$, and sends to P_1 z'_6, z'_7, z'_8, and z'_9. P_2 executes $\mathsf{PKPM}((b_2, \rho_2 + \alpha_2), (z_{(1,1)}, z_{(2,1)}, z'_6))$, $\mathsf{PKPM}((1-2b_2 \bmod N, \rho_2^{-2} + \alpha_3), (a_3, z_{(2,4)}, z'_7))$, $\mathsf{PKPM}((r_2, \rho'_2), (z'_7, z_{(2,2)}, z'_8))$, and $\mathsf{PKPM}((r'_2, \rho''_2), (z_{(2,5)}, z_{(2,3)}, z'_9))$ to prove that z'_6, z'_7, z'_8, and z'_9 are well-formed.

7. Each party locally computes $a'_3 = z'_8 \cdot z'_9$.

8. The parties decrypt a'_3. If the decrypted value is $< N^2/2$, output $z_{(1,1)} \cdot z_{(2,1)} \cdot (z'_6)^{-2}$; otherwise, output $e_1(z'_6)^2 \cdot (z_{(1,1)} \cdot z_{(2,1)})^{-1}$.

Our protocol closely follows the logic of the semi-honest solution [9], where we additionally need to employ zero-knowledge proofs to provide each party with the ability to verify that the computation was performed correctly by the other party. We do not explicitly compute both a'_1 and a'_2 as in the semi-honest version, but rather compute and return either a'_1 or a'_2 in step 8 after decrypting the content of a'_3. We also explicitly keep track of random values used for creating ciphertexts and use them as input into ZKPKs. This ensures that the protocol is fully specified.

Our security result can be stated as given next. Recall that we use a hybrid model with ideal decryption, which we then replace with a real instantiation of threshold decryption.

Theorem 2 *Assuming semantic security of the homomorphic encryption and security of the building blocks,* MalLT *protocol is secure in the presence of a malicious adversary in the hybrid model with ideal decryption.*

Proof: We prove security of MalLT based on Definition 2. We separately consider the cases when P_1 is malicious and when P_2 is malicious. In the ideal world, we build a simulator S_j that is located between malicious party P_j and the TTP and simulates P_j's view of the protocol after querying the TTP for P_j's output.

First, we treat the case of malicious P_1 and build the corresponding simulator S_1. Upon obtaining the input $\mathsf{Enc}(x), \mathsf{Enc}(y), \ell$, S_1 queries the TTP for $\mathsf{Enc}(b)$. In step 1, S_1 performs the same computation as P_2. In step 2, S_1 acts as a verifier for P_1's ZKPKs $\mathsf{PK12}$ and $\mathsf{RangeProof}$s and extracts P_1's input to the proofs. It also chooses a random bit w and computes $\mathsf{Enc}(b^*) = \mathsf{Enc}(b)^{1-w} \cdot (e_1(\mathsf{Enc}(b))^{-1})^w = \mathsf{Enc}(b \cdot (1-w) + (1-b) \cdot w) = \mathsf{Enc}(b \oplus w)$. In other words, if $w = 0$ then $b^* = b$ and otherwise $b^* = 1 - b$. S_1 similarly computes $\mathsf{Enc}(b_2) = \mathsf{Enc}(b^* \oplus b_1)$ and chooses and encrypts two random values r_2 and r'_2 according to the protocol. S_1 now simulates P_2's proofs $\mathsf{PK12}$ and $\mathsf{RangeProof}$ using $\mathsf{Enc}(b_2)$, $\mathsf{Enc}(r_2)$, and $\mathsf{Enc}(r'_2)$. In steps 3–5, S_1 acts like P_2, i.e., S_1

performs the prescribed computation in steps 3 and 5 and acts as a verifier for P_1's proofs PKPMs during step 4. During step 6, S_1 computes $z'_6 = \mathsf{Enc}(b_2)^{b_1} \cdot \mathsf{Enc}(0, \beta_1)$ and $z'_7 = a_3^{1-2w'} \cdot \mathsf{Enc}(0, \beta_2)$, where w' is a newly generated random bit and β_1, β_2 correspond to freshly chosen randomness during encryption. S_1 also computes the remaining z'_8 and z'_9 according to the protocol. Note that using random w' in place of b_2 makes the content of ciphertexts z'_7 and z'_8 inconsistent with other encrypted values, but P_1 cannot tell this fact due to security of the encryption scheme. S_1 then simulates the PKPM proofs in step 6. Finally, in step 7, S_1 sends a positive properly chosen value $\hat{c} \in [0, N^2/2)$ to P_1 if $w = 0$, and a negative properly chosen value $\hat{c} \in [N^2/2, N^2)$ otherwise. To form $\hat{c}$, S_1 randomly samples the distribution of the absolute value of $x - y$ (using application-specific knowledge of distributions of x and y); let the randomly chosen value be denoted by d. S_1 then sets $\hat{c} = (r_1 \cdot d - r'_1)r_2 - r'_2$ if $w = 0$ and $\hat{c} = N^2 - (r_1 \cdot d - r'_1)r_2 + r'_2$ otherwise. Note that ciphertexts z'_7 and z'_8 are not used by the simulator beyond step 6 and thus their contents do not affect correctness of the output.

To show that the view simulated by S_1 results in P_1 obtaining correct output, we note that $w = b_1 \oplus b_2 \oplus b$ (because $b_1 = b_2 \oplus b \oplus w$). Now, if $w = 0$, $b = b_1 \oplus b_2 = b_1 + b_2 - 2b_1b_2$. Thus, S_1 needs to produce a positive value ($< N^2/2$) as the decryption of a'_3 in step 8, so that $z_{(1,1)} \cdot z_{(2,1)} \cdot (z'_6)^{-2} = \mathsf{Enc}(b_1 + b_2 - 2b_1b_2) = \mathsf{Enc}(b)$ is produced as the output in step 8. Otherwise, if $w = 1$, $b = 1 \oplus b_1 \oplus b_2 = 1 - (b_1 \oplus b_2) = 1 - b_1 - b_2 + 2b_1b_2$. Thus, S_1 needs to produce a negative value ($\geq N^2/2$) as the decryption of a'_3 in step 8, so that P_1 uses $e_1(z'_6)^2 \cdot (z_{(1,1)}z_{(2,1)})^{-1} = \mathsf{Enc}(1 + 2b_1b_2 - b_1 - b_2) = \mathsf{Enc}(b)$ as the output.

To show that the view simulated by S_1 is indistinguishable from P_1's view in the hybrid model (with ideal decryption), we note that most values the simulator sends to P_1 (e.g., steps 2 and 6) are encrypted and thus are indistinguishable from ciphertexts sent during the protocol execution because of semantic security of the encryption scheme. Similarly, the simulations of $\mathsf{PK12}$, $\mathsf{RangeProofs}$, and PKPMs are indistinguishable from real execution due to their security properties. The only value that S_1 provides to P_1 in the clear is the decryption of a_3' in step 7. This value was chosen by S_1 in the same way as during the protocol after randomly sampling the absolute value of $x - y$ according to what is known about distributions of x and y. Thus, P_1 is unable to distinguish the value received during the simulation from the value received during the protocol execution. We also note that during the simulation S_1 aborts the computation in exactly the same circumstances when the computation is aborted in the real execution (namely, when a ZKPK does not verify) and thus the simulation cannot be distinguished from real execution on the grounds of computation termination.

Now let P_2 be malicious. We build a simulator S_2 that constructs P_2's view similar to the way S_1 did for P_1. Most of S_2's computations are the same as S_1's computations, and thus we concentrate on the differences. In this case, S_2 computes $\mathsf{Enc}(b^*)$ in step 2 in the same way S_1 did (i.e., by choosing a random bit w and using $b^* = b \oplus w$) and then sets $\mathsf{Enc}(b_1) = \mathsf{Enc}(b^* \oplus b_2)$. In step 4, S_2 selects a random bit w' and a random number ξ_1 to be used as randomness during encryption and computes $z_6 = \mathsf{Enc}(c)^{1-2w'} \cdot \mathsf{Enc}(0, \xi_1)$. S_2 also computes z_7 and z_8 according the protocol and simulates the PKPM proofs (step 4). Note that using w' instead of b_1 (which S_2 doesn't know) in the computation of z_6 makes the content of ciphertexts z_7, a_3, z_7', z_8', and a_3' inconsistent with other encrypted values, but P_2 is unable to tell this fact because of security of the encryption scheme. In steps 5 and 6, S_2 acts like P_1. Finally, in step 7, S_2 provides a positive properly chosen value $\tilde{c} \in [0, N^2/2)$ to P_2 if w was 0, and otherwise a negative properly chosen value $\tilde{c} \in [N^2/2, N^2)$. Note that none of incorrectly formed ciphertexts (z_6, z_7, a_3, z_7', z_8', and a_3') are used in the computation of the protocol's output and correctness of the result is not affected.

Correctness of the output that P_2 learns at the end of S_2's simulation can be shown in a way similar to that of S_1's simulation. In other words, we now also have that $w = b_1 \oplus b_2 \oplus b$ and producing a positive value as the decryption of a'_3 when $w = 0$ and producing a negative value when $w = 1$ results in P_2 computing $\mathsf{Enc}(b)$.

What remains to show is that the view simulated by S_2 is indistinguishable from P_2's view in the hybrid model. Similar to S_1's case, we have that all values that S_2 sends to P_2 are either protected via semantically secure encryption (steps 2 and 6), are simulated ZKPKs, or a plaintext $\tilde{c}$ chosen in the same way as $\hat{c}$ in S_1 simulation and is indistinguishable from the value decrypted during real protocol execution. Thus, assuming security of the building blocks, the claim follows. $\square$

4.6.2.3 Secure Truncation

In this section we are going to describe our (probabilistic) truncation protocol $\mathsf{MalTruncPR}$ secure in the malicious model. Our starting point was the semi-honest $\mathsf{TruncPR}$ from [45], which we adjusted to the two-party setting based on homomorphic encryption.[3] On input of an ℓ-bit encrypted x and a positive integer $k < \ell$, the protocol computes $\lfloor x/2^k \rfloor + b$, where b is either 0 or 1. In other words, the protocol truncates k least significant bits of x, but might also increment the result by 1.

At high level, the solution proceeds by the parties jointly and privately choosing two random values r' and r'' of bitlength k and $\ell - k + \kappa$, respectively, where κ is a statistical security parameter. The parties then blind x by $(\kappa+\ell)$-bit random number $r = 2^k r'' + r'$ and decrypt the sum $c = x + r$. The encrypted output y is computed as $(x + r' - (c \bmod 2^k))2^{-k}$.

We next present our $\mathsf{MalTruncPR}$ protocol. As before, we follow the logic of the semi-honest protocol, but need to employ stronger building blocks and ZKPKs.

[3]We also note that $\mathsf{TruncPR}$ in [45] was designed to work on both positive and negative integers, while in our case supporting only non-negative integers is sufficient.

$\mathsf{Enc}(y) \leftarrow \mathsf{MalTruncPR}(\mathsf{Enc}(x), \ell, k)$

Public inputs include public key $pk = (g, N, \theta)$ for a (2,2)-threshold Paillier encryption scheme and private inputs consist of shares of the corresponding secret key.

1. Each P_j randomly chooses $r'_{(j,i)} \in \{0, 1\}$ for $i \in [1, k]$, computes $z_{(j,i)} = \mathsf{Enc}(r'_{(j,i)}, \rho_{(j,i)})$ using randomness $\rho_{(j,i)}$, sends to the other party each $z_{(j,i)}$, and executes $\mathsf{PK12}((r'_{(j,i)}, \rho_{(j,i)}), (z_{(j,i)}, 0, 1))$ to prove that $z_{(j,i)}$ encrypts a bit.

2. The parties compute $z'_i = \mathsf{Enc}(r_i) = \mathsf{Enc}(r'_{(1,i)} \oplus r'_{(2,i)}) = z_{(1,i)} \cdot z_{(2,i)} \cdot (\mathsf{MalMul}(r'_{(1,i)}, r'_{(2,i)}))^{-2}$ for $i \in [1, k]$.

3. Each party locally computes $\mathsf{Enc}(r') = \mathsf{Enc}(\sum_{i=1}^{k} r_i^{2^i}) = \prod_{i=1}^{k} (z'_i)^{2^i}$.

4. Each P_j randomly chooses $r''_j \in [0, 2^{\ell+\kappa-k} - 1]$, computes $z''_j = \mathsf{Enc}(r''_j, \rho'_j)$, sends to the other party z''_j, and executes $\mathsf{RangeProof}((r''_j, \rho'_j), (0, 2^{\ell+\kappa-k} - 1, z''_j))$ to prove that z''_j is well-formed.

5. Each party locally computes $\mathsf{Enc}(r'') = \mathsf{Enc}(r''_1 + r''_2) = \mathsf{Enc}(r''_1) \cdot \mathsf{Enc}(r''_2)$.

6. The parties locally compute $\mathsf{Enc}(c) = \mathsf{Enc}(x + 2^k r'' + r') = \mathsf{Enc}(x) \cdot \mathsf{Enc}(r'')^{2^k} \cdot \mathsf{Enc}(r')$ and jointly decrypt c.

7. Each party locally computes $c'' = \lfloor \frac{c}{2^k} \rfloor$ and and produces $\mathsf{Enc}(y) = \mathsf{Enc}(c'' - r'') = \mathsf{Enc}(c'', 0) \cdot \mathsf{Enc}(r'')^{-1}$ as the output.

One significant difference from the semi-honest protocol is the way random k-bit value r' is generated. In the semi-honest version, r' is set to the sum $r'_1 + r'_2$, where r'_i is a $(k-1)$-bit random value chosen by P_i. To make this stronger for the malicious model, we could enforce that each party chooses its respective r'_i from the correct range using a range proof. Unfortunately, this is not sufficient for security. In the malicious model, the parties are not guaranteed to draw random values uniformly at random from the specified range and we can no longer expect that the sum $r'_1 + r'_2$ is k bits long. Suppose that a malicious party P_i sets its r'_i to 0, which guarantees that the sum r' is $k-1$ bits long. Then after the sum $c = x + 2^k r'' + r'$ is decrypted, the adversary can learn unintended information about the kth bit of x. In particular, if the kth bit of c is 0, the malicious party knows that the kth bit of x is 0. To eliminate this vulnerability, we instead require that both participants select their r'_i's to be k bits long and r' is computed via XOR as $r'_1 \oplus r'_2$.

Another conceptual difference from the semi-honest solution is that instead of using $c \bmod 2^k$ in computing the result, the parties now use $\lfloor c/2^k \rfloor$. This simplifies computation of the output, but results in identical outcome. As before, we explicitly keep track of random values used for creating ciphertexts and use them as input into ZKPKs. We next show security of this protocol.

Theorem 3 *Assuming semantic security of the homomorphic encryption and security of the building blocks,* MalTruncPR *protocol is secure in the presence of a malicious adversary in the hybrid model with ideal decryption.*

Proof: We prove security of MalTruncPR based on Definition 2. When P_j is malicious, we need to construct simulator S_j that provides a view for P_j in the ideal world, which is indistinguishable from the protocol execution in the hybrid model. In what follows, without loss of generality, let us assume that P_1 is malicious; a very similar proof can be given for the case of malicious P_2 because of the protocol's symmetry.

In step 1, S_1 acts similar to what the protocol prescribes for P_2: it receives P_1's ciphertexts $z_{(1,i)}$'s, acts as a verifier for P_1's ZKPKs (extracting P_1's inputs), forms P_2's random bits and corresponding ciphertexts, and acts as a prover in ZKPKs to show their correctness. During step 2, S_1 invokes MalMul's simulator. In step 4, S_1 receives z''_1 from P_1 and acts as a verifier for P_1's RangeProof for z''_1 (extracting r''_1). S_1 also chooses random $\hat{c} \in \{0,1\}^{\ell+\kappa}$, computes $\tilde{c} = \hat{c} + 2^k r''_1$ and $z''_2 = \mathsf{Enc}(\lfloor \hat{c}/2^k \rfloor - y) = \mathsf{Enc}(\lfloor \hat{c}/2^k \rfloor)\mathsf{Enc}(y)^{-1}$, sends z''_2 to P_1, and simulates the RangeProof for z''_2. In step 6, S_1 outputs $\tilde{c}$ as the decrypted value. We note that $\tilde{c}$ is formed by the simulator inconsistently with the values used for computing r'. This is not a problem because

P_1 does not use r' in producing its output and inconsistency of encrypted values cannot be detected as well.

To see that P_1 obtains the correct (encrypted) output at the end of S_1's simulation, recall that S_1 sets $z_2'' = \mathsf{Enc}(\lfloor \hat{c}/2^k \rfloor - y)$ in step 4. This means that P_1 computes in step 5 encryption of $r'' = \lfloor \hat{c}/2^k \rfloor - y + r_1''$. P_1 also learns $c = \tilde{c} = \hat{c} + 2^k r_1''$ in step 6 and consequently sets $c'' = \lfloor \hat{c}/2^k \rfloor + r_1''$. P_1 then sets the (encrypted) output to $c'' - r'' = \lfloor \hat{c}/2^k \rfloor + r_1'' - (\lfloor \hat{c}/2^k \rfloor - y + r_1'') = y$, as desired.

To show that the view simulated by S_1 is indistinguishable from the view in the hybrid model execution, we note that indistinguishability of encrypted data and all building blocks (i.e., ZKPKs, and MalMul) follows security of the building blocks. The only value revealed to P_1 in the clear is $c = \hat{c}$ in step 6. The value produced by the simulator, however, is statistically indistinguishable from the value of c used during real execution (using statistical security parameter κ). In addition, both during the simulation and real execution the computation aborts in identical circumstances when the malicious party fails to correctly complete ZKPKs as the prover. Thus, indistinguishability of simulated and real views follows. □

4.6.2.4 Secure Inversion

The next protocol that we treat is computation of a multiplicative inverse of an encrypted integer x, where x is treated as a group element. As before, our starting point was a semi-honest inversion protocol, which we adapt to the two-party setting based on homomorphic encryption. The main idea of this protocol is for the parties to jointly generate a random element r of the group, compute and decrypt $c = r \cdot x$, invert plaintext c, and then compute the inverse of x as $r \cdot c^{-1} = x^{-1}$ in the encrypted form.

Our protocol in the malicious model follows the logic of the semi-honest solution, but we modify the way $\mathsf{Enc}(rx)$ is computed from $\mathsf{Enc}(x)$. In particular, instead of having the parties compute $\mathsf{Enc}(r)$ and call multiplication on $\mathsf{Enc}(r)$ and $\mathsf{Enc}(x)$, we avoid calling relatively costly MalMul. We instead have each party P_j compute (and prove correctness of) $\mathsf{Enc}(x)^{r_j} = \mathsf{Enc}(r_j x)$ for its respective share r_j of r. The parties then locally compute $\mathsf{Enc}(rx) = \mathsf{Enc}(r_1 x + r_2 x)$ and proceed with the rest of the protocol as before. Security of the protocol is stated after protocol description.

$\mathsf{Enc}(y) \leftarrow \mathsf{MalInv}(\mathsf{Enc}(x))$

Public inputs include public key $pk = (g, N, \theta)$ for a (2,2)-threshold Paillier encryption scheme and private inputs consist of shares of the corresponding secret key.

1. Each P_j chooses at random $r_j \in \mathbb{Z}_N^*$, computes $z_j = \mathsf{Enc}(r_j, \rho_j)$ using fresh randomness ρ_j, sends z_j to the other party, and executes $\mathsf{PKP}((r_j, \rho_j), (z_j))$ to prove that z_j was formed correctly.

2. Each P_j computes $z'_j = \mathsf{Enc}(r_j x) = \mathsf{Enc}(x)^{r_j}$, sends z'_j to the other party, and executes $\mathsf{PKPM}((r_j, \rho_j), (\mathsf{Enc}(x), z_j, z'_j))$ to prove correctness of z'_j.

3. Each party locally computes $\mathsf{Enc}(c) = \mathsf{Enc}((r_1 x + r_2 x) = \mathsf{Enc}(r_1 x) \cdot \mathsf{Enc}(r_2 x)$ and the parties jointly decrypt c.

4. Each party locally computes and outputs $\mathsf{Enc}(y) = \mathsf{Enc}((r_1 + r_2)c^{-1}) = (z_1 z_2)^{c^{-1}}$.

Theorem 4 *Assuming semantic security of the homomorphic encryption and security of the building blocks,* MalInv *protocol is secure in the presence of a malicious adversary in the hybrid model with ideal decryption.*

Proof: We prove security of MalInv based on Definition 2. Because the protocol is symmetric, we assume without loss of generality that P_1 is malicious and build the corresponding simulator S_1. In the beginning of the protocol (step 1), S_1 receives z_1, chooses a random number $\hat{c} \in \mathbb{Z}_N^*$, computes $z_2 = \mathsf{Enc}(\hat{c} \cdot y - r_1) = \mathsf{Enc}(y)^{\hat{c}} \cdot z_1^{-1}$ using output $\mathsf{Enc}(y)$ received from the TTP, and sends z_2 to P_1. S_1 also simulates its PKP proof and acts as a verifier for P_1's proof obtaining r_1. In step 2, S_1 receives z'_1, chooses a random number $r_2 \in \mathbb{Z}_N^*$, computes $z'_2 = \mathsf{Enc}(r_2 x) = \mathsf{Enc}(x)^{r_2}$, and sends z'_2 to P_1. Both parties also execute their respective PKPM proofs, where S_1 uses simulation. Note that now z_2 and z'_2 have inconsistent contents, but this fact is not known to P_1 due to security of encryption. In step 3, S_1 output $\hat{c}$ as the result of decryption.

To show that P_1 computes correct output $\mathsf{Enc}(x^{-1})$, recall that the simulator outputs $c = \hat{c}$ and P_1 computes the result as $(z_1 z_2)^{c^{-1}}$. In the simulated view, we have $(z_1 z_2)^{c^{-1}} = (\mathsf{Enc}(r_1) \cdot \mathsf{Enc}(\hat{c}y - r_1))^{\hat{c}^{-1}} = \mathsf{Enc}(\hat{c} \cdot y \cdot \hat{c}^{-1}) = \mathsf{Enc}(y)$, as desired.

To show that the view simulated by S_1 is indistinguishable from the execution view in the hybrid model, notice that all information that P_1 receives is indistinguishable in both views due to security of the underlying building blocks with the exception of plaintext c that P_1 learns in step 3, which we need to analyze. During the simulation, S_1 outputs $\hat{c}$ chosen uniformly at random from the group. In the real execution, P_1 learns $(r_1 + r_2)x$, which is also a random element of the group. Thus, the values produced in the two worlds are indistinguishable. Lastly, in both worlds the execution aborts only when the malicious party fails to correctly complete ZKPKs, which completes this proof. □

4.6.2.5 Secure Prefix Multiplication

We next present prefix multiplication protocol, which on input of integers $x_1, \ldots, x_k$, outputs $y_1, \ldots, y_k$, where each $y_i = \prod_{j=1}^{i} x_j$. We provide the semi-honest prefix multiplication protocol adapted to the two-party setting based on homomorphic encryption from [45]. We used the protocol as our starting point and modified it to be secure in the stronger security model with malicious participants.

The main idea behind PreMul protocol is for the parties to compute and open $\mathsf{Enc}(m_i) = \mathsf{Enc}(r_i \cdot x_i \cdot r_{i-1}^{-1})$ for $i \in [2, k]$ and $\mathsf{Enc}(m_1) = r_1 x_1$, where each r_i is a random element of the group and the revealed values completely hide each input x_i. Then, each party can compute the output as $y_i = r_i^{-1} \cdot (\prod_{j=1}^{i} m_j) = r_i^{-1} \cdot r_i \cdot$

$x_i \cdot r_{i-1}^{-1} \cdots r_2 \cdot x_2 \cdot r_1^{-1} \cdot r_1 \cdot x_1$ in the encrypted form using encryptions of r_i^{-1}'s and plaintext m_i's. Each r_i is jointly chosen by the parties at random and computation of each r_i^{-1} proceeds similar to the inversion protocol. Namely, the parties also generate encryptions of random values s_i's, decrypt products $u_i = r_i \cdot s_i$, and use inverses of u_i's in the consecutive computation.

$\mathsf{Enc}(y_1), \ldots, \mathsf{Enc}(y_k) \leftarrow \mathsf{MalPreMul}(\mathsf{Enc}(x_1), \ldots, \mathsf{Enc}(x_k))$

Public inputs include public key $pk = (g, N, \theta)$ for a (2,2)-threshold Paillier encryption scheme and private inputs consist of shares of the corresponding secret key.

1. Each P_j chooses $r_{(j,i)}, s_{(j,i)} \in \mathbb{Z}_N^*$ at random for $i \in [1, k]$, computes $z_{(j,i)} = \mathsf{Enc}(r_{(j,i)}, \rho_{(j,i)})$ and $z'_{(j,i)} = \mathsf{Enc}(s_{(j,i)}, \rho'_{(j,i)})$, and sends each $z_{(j,i)}$ and $z'_{(i,j)}$ to the other party. P_j also executes $\mathsf{PKP}((r_{(j,i)}, \rho_{(j,i)}),\ (z_{(j,i)}))$ and $\mathsf{PKP}((s_{(j,i)}, \rho'_{(j,i)}),\ (z'_{(j,i)}))$ for each i to prove that $z_{(j,i)}$'s and $z'_{(j,i)}$'s are well-formed.

2. Each P_j locally computes $z_i = \mathsf{Enc}(r_i) = \mathsf{Enc}(r_{(1,i)} + r_{(2,i)}) = z_{(1,i)} \cdot z_{(2,i)}$ and $z'_i = \mathsf{Enc}(s_i) = \mathsf{Enc}(s_{(1,i)} + s_{(2,i)}) = z'_{(1,i)} \cdot z'_{(2,i)}$ for $i \in [1, k]$.

3. Each P_j computes $a_{(j,i)} = \mathsf{Enc}(r_i \cdot s_{(j,i)}) = (z_i)^{s_{(j,i)}}$ for $i \in [1, k]$, sends to the other party $a_{(j,i)}$'s, and executes $\mathsf{PKPM}((s_{(j,i)}, \rho'_{(j,i)}), (z_i, z'_{(j,i)}, a_{(j,i)})$ to prove that each $a_{(j,i)}$ is well-formed.

4. Each P_j computes $b_{(j,i)} = \mathsf{Enc}(r_{i+1} \cdot s_{(j,i)}) = (z_{i+1})^{s'_{(j,i)}}$ for $i \in [1, k-1]$, sends to the other party $b_{(j,i)}$'s, and executes $\mathsf{PKPM}((s_{(j,i)}, \rho'_{(j,i)}), (z_{i+1}, z'_{(j,i)}, b_{(j,i)}))$ to prove that $b_{(j,i)}$ is well-formed.

5. The parties locally compute $\mathsf{Enc}(u_i) = \mathsf{Enc}(r_i \cdot s_i) = a_{1,i} \cdot a_{2,i}$ for $i \in [1, k]$ and jointly decrypt each u_i.

6. Each party locally computes $\mathsf{Enc}(v_i) = \mathsf{Enc}(r_{i+1} \cdot s_i) = b_{(1,i)} \cdot b_{(2,i)}$ for $i \in [1, k-1]$.

7. Each party locally sets $\mathsf{Enc}(w_1) = \mathsf{Enc}(r_1) = z_1$ and for $i \in [2, k]$ computes $\mathsf{Enc}(w_i) = \mathsf{Enc}(v_{i-1} \cdot (u_{i-1})^{-1}) = \mathsf{Enc}(v_{i-1})^{(u_{i-1})^{-1}}$.

8. Each party also locally computes $\mathsf{Enc}(t_i) = \mathsf{Enc}(s_i \cdot (u_i^{-1})) = (z'_i)^{(u_i)^{-1}}$ for $i \in [1, k]$.

9. For $i \in [1, k]$, the parties compute $\mathsf{Enc}(m_i) = \mathsf{MalMul}\ (\mathsf{Enc}(w_i), \mathsf{Enc}(x_i))$ and

decrypt each m_i.

10. Each party sets $\mathsf{Enc}(y_1) = \mathsf{Enc}(x_1)$ and locally computes $\mathsf{Enc}(y_i) = \mathsf{Enc}(t_i \prod_{j=1}^{i} m_j) = (\mathsf{Enc}(t_i))^{\prod_{j=1}^{i} m_j}$ for $i \in [2, k]$ as the output.

The high-level logic of our solution is the same as in the semi-honest setting, but we modify how some encrypted values are computed to result in a faster solution. In particular, we avoid the use of the multiplication protocol for computing encrypted u_i's and v_i's and instead employ local multiplications and proofs of correctness using PKPM's. The computed values are the same, but the mechanism for their computation differs resulting in computational savings. We next show security of this protocol:

Theorem 5 *Assuming semantic security of the homomorphic encryption and security of the building blocks,* $\mathsf{MalPreMul}$ *protocol is secure in the presence of a malicious adversary in the hybrid model with ideal decryption.*

Proof: As before, we proceed according to the security notion from Definition 2 and build a simulator S_j that creates a view for P_j in the ideal model, which is indistinguishable from P_j's view in protocol's real execution. Because $\mathsf{MalPreMul}$ is symmetric, we assume without loss of generality that P_1 is malicious and build a corresponding simulator S_1.

In the beginning, S_1 submits inputs to the TTP and receives the output $\mathsf{Enc}(y_i)$'s. S_1 also chooses random $\hat{m}_i, d_i \in \mathbb{Z}_N^*$ and computes $\hat{u}_i = d_i(\prod_{j=1}^{i} \hat{m}_j)$ for $i \in [1, k]$. In step 1, S_1 receives $z_{(1,i)}$ and $z'_{(1,i)}$ from P_1 for each i. It chooses its own random $r_{(2,i)}$'s, encrypts them as $z_{(2,i)} = \mathsf{Enc}(r_{(2,i)}, \rho_{(2,i)})$, computes $z'_{(2,i)} = \mathsf{Enc}(y_i \cdot t_i - s_{(1,i)}) = \mathsf{Enc}(y_i)^{t_i} \cdot (z'_{(1,i)})^{-1}$, re-randomizes each $z'_{(2,i)}$ (by multiplying it to a fresh encryption of 0), and sends to P_1 each $z_{(2,i)}$ and $z'_{(2,i)}$. S_1 invokes simulator for its own and P_1's PKP proofs (extracting P_1's inputs). S_1 doesn't perform any computation in step 2. In step 3, S_1 receives $a_{(1,i)}$'s from P_1, chooses random elements $a_{(2,i)}$ from the ciphertext space, and sends these $a_{(2,i)}$'s to P_1. S_1 uses simulation for PKPM interaction. Similarly, in step, S_1 receives $b_{(1,i)}$'s from P_1, chooses random elements $b_{(2,i)}$ from the ciphertext space, and sends these $a_{(2,i)}$'s to P_1. S_1 also uses simulation for PKPM interactions. Note that using random $a_{(2,i)}$'s and $b_{(2,i)}$'s makes the content of ciphertexts $\mathsf{Enc}(u_i)$ and $\mathsf{Enc}(v_i)$ in consecutive steps inconsistent with other encrypted values, but P_1 cannot tell this fact. In step 5, S_1 uses $\hat{u}_i$'s as decryptions and then skips steps 6–8. In step 9, S_1 invokes simulator for MalMul to interact with P_1, and provides $\hat{m}_i$'s to P_1 as decrypted values.

To show that P_1 computes correct output during S_1's simulation, first notice that each $\hat{u}_i = d_i(\prod_{j=1}^{i} \hat{m}_j)$ and thus $\prod_{j=1}^{i} \hat{m}_j = \hat{u}_i \cdot d_i^{-1}$, where $\hat{u}_i$'s and $\hat{m}_i$'s are used as u_i's and m_i's, respectively. In addition, the simulator sets $s_{(2,i)} = d_i \cdot y_i - s_{(1,i)}$, so that $s_i = s_{(1,i)} + s_{(2,i)} = d_i \cdot y_i$. Now, when P_1 computes the ith component of the output, it uses computation (on encrypted values) $t_i(\prod_{j=1}^{i} m_j) = s_i \cdot u_i^{-1}(\prod_{j=1}^{i} m_i) = d_i \cdot y_i \cdot \hat{u}_i^{-1}(\prod_{j=1}^{i} \hat{m}_i) = d_i \cdot y_i \cdot \hat{u}_i^{-1} \cdot \hat{u}_i \cdot d_i^{-1} = y_i$, as required.

To show that the view simulated by S_1 is indistinguishable from P_1's view in the hybrid model, we only need to show that plaintexts u_i's and m_i's that the simulator outputs do not violate indistinguishability, as the remaining portions of the protocol are indistinguishable because of the assumption that all building blocks and ZKPKs are secure. Similarly, indistinguishability cannot be violated if the execution aborts in the ideal or real model, but not in the other because the only time the execution terminates in either world is when the malicious party does not follow the computation and fails to complete a ZKPK.

Regarding the release of $\hat{m}_i$'s and $\hat{u}_i$'s by the simulator, we first note that the release of $\hat{m}_i$'s only reveals no information to P_1 because each $\hat{m}_i$ was chosen uniformly at random. Each $\hat{u}_i$, on the other hand, is a function of $\hat{m}_i$'s, but each u_i was randomized by a new random value d_i and thus $\hat{u}_i$ is also a random element of the group. In the real protocol execution, each u_i is formed as $r_i \cdot s_i$ and each m_i (except m_1) is formed as $r_i \cdot x_i \cdot r_{i-1}^{-1}$, which are also distributed as random elements of the group. Thus, we obtain that P_1 cannot tell the difference between the simulated and real protocol execution with a non-negligible probability. $\square$

4.6.2.6 Secure Bit Decomposition

Finally, we describe our last, bit decomposition, protocol secure in the malicious model. Our starting point was the bit composition protocol in the semi-honest setting from [46], which we adapted to the two-party setting based on homomorphic encryption. On input of an ℓ-bit encrypted integer a, the protocol performs bit decomposition of k least significant bits of a.

The main idea of BitDec protocol for the parties to compute $\mathsf{Enc}(c) = \mathsf{Enc}(2^{\ell+k} + a - r)$, where r is a random $(\ell + \kappa)$-bit value and the k least significant bits of r are available to the parties in encrypted form, and decrypt c. The plaintext lets each party to compute the bits of $2^{\ell+\kappa} + a - r$ while providing statistical hiding of a. The random r is created in the same way as in the truncation protocol, where the parties separately create k least significant bits of r and choose a single random r' for the remaining bits of r. The parties then call a protocol called BitAdd that takes k least significant (plaintext) bits of c and k least significant (encrypted) bits of r and performs addition of the values provided by their bitwise representation (i.e., addition of two k-bit quantities). BitAdd outputs k encrypted bits of the sum, which are used as the output of the BitDec protocol.

In our MalBitDec protocol we need to employ a stronger version of BitAdd, which was provided for the semi-honest setting. We, however, notice that BitAdd is composed entirely of addition and multiplication operations [46] and we can obtain a protocol secure in the malicious model, which we denote by MalBitAdd, by employing protocol MalMul in place of ordinary multiplications. Adding two integers x and y in bitwise form involves computing sum and carry bits s_i and e_i, which can be sequentially computed as $e_0 = x_0 \wedge y_0 = x_0 \cdot y_0$, $s_0 = x_0 \oplus y_0 = x_0 + y_0 - 2e_0$, and

$e_i = (x_i \wedge y_i) \vee ((x_i \oplus y_i) \wedge e_{i-1}) = x_i \cdot y_i + (x_i \oplus y_i)e_{i-1}$, $s_i = x_i + y_i + e_{i-1} - 2e_i$ for $i \geq 1$. Bitwise addition protocol [8] used to implement bit decomposition uses concurrent execution to compute all bits of the sum (and carry bits) using a smaller (than linear in the size of the input) number of rounds, but still implements the formulas given above. This will be relevant for our security proof.

$\mathsf{Enc}(x_{k-1}), ..., \mathsf{Enc}(x_0) \leftarrow \mathsf{MalBitDec}(\mathsf{Enc}(a), \ell, k)$

Public inputs include public key pk for a (2,2)-threshold Paillier encryption scheme and private inputs consist of shares of the corresponding secret key.

1. For $i \in [0, k-1]$, each P_j chooses random bits $r_{(j,i)} \in \{0, 1\}$, encrypts them as $z_{(j,i)} = \mathsf{Enc}(r_{(j,i)}, \rho_{(j,i)})$, and sends each $z_{(j,i)}$ to the other party. P_j also executes $\mathsf{PK12}((r_{(j,i)}, \rho_{(j,i)}), (z_{(j,i)}, 0, 1))$ to prove that each $z_{(j,i)}$ is well-formed.

2. The parties compute $z_i = \mathsf{Enc}(r_i) = \mathsf{Enc}(r_{(1,i)} \oplus r_{(2,i)})) = \mathsf{Enc}(r_{(1,i)} + r_{(2,i)} - 2r_{(1,i)}r_{(2,i)}) = z_{(1,i)} \cdot z_{(2,i)} \cdot (\mathsf{MalMul}(z_{(1,i)}, z_{(2,i)}))^{-2}$ for $i \in [0, k-1]$.

3. Each P_j chooses random $r'_j \in [0, 2^{\ell+\kappa-k} - 1]$, encrypts it as $z'_j = \mathsf{Enc}(r'_j, \rho'_j)$, and sends it to the other party. P_j also executes $\mathsf{RangeProof}((r'_j, \rho'_i)(0, 2^{\ell+\kappa-k} - 1, z'_j))$ to prove that z'_j is well-formed.

4. Each party locally computes $\mathsf{Enc}(r) = \mathsf{Enc}(2^k(r'_1 + r'_2) + \sum_{i=0}^{k-1} r_i \cdot 2^i) = (z'_1 \cdot z'_2)^{2^k} \prod_{i=0}^{k-1} z_i^{2^i}$ and $\mathsf{Enc}(c) = \mathsf{Enc}(2^{\ell+\kappa+1} + a - r) = \mathsf{Enc}(2^{\ell+\kappa+1}, 0) \cdot \mathsf{Enc}(a) \cdot \mathsf{Enc}(r)^{-1}$.

5. The parties jointly decrypt $\mathsf{Enc}(c)$ to learn c.

6. The parties compute and output $(\mathsf{Enc}(x_{k-1}), \ldots, \mathsf{Enc}(x_0)) = \mathsf{MalBitAdd}((c_{k-1}, ..., c_0), (\mathsf{Enc}(r_{k-1}), \ldots, \mathsf{Enc}(r_0)))$, where $c_0, \ldots, c_{k-1}$ are k least significant bits of c.

Our protocol closely follows the logic of the semi-honest solution. We employ zero-knowledge proofs to verify that the computation was performed correctly and building blocks secure in the stronger security model. We show security of this protocol as follows:

Theorem 6 *Assuming semantic security of the homomorphic encryption and security of the building blocks,* $\mathsf{MalBitDec}$ *protocol is secure in the presence of a malicious adversary in the hybrid model with ideal decryption.*

Proof: We prove security of $\mathsf{MalBitDec}$ based on Definition 2. We construct P_j's view in the ideal model by building simulator S_j, and we show it is indistinguishable form view of P_j in protocol's real execution. We assume without loss of generality that P_1 is malicious, and we build simulator S_1. We can use similar proof in case P_2 is malicious because $\mathsf{MalBitDec}$ is symmetric.

In step 1, S_1 receives $z_{(1,i)}$'s, and acts as a verifier for $\mathsf{PK12}$'s (extracting $r_{(1,i)}$'s). S_1 then chooses a random number $\tilde{c} \in \{0,1\}^{\ell+\kappa}$ and computes

1. $\mathsf{Enc}(r_i) = \mathsf{Enc}(x_i - \tilde{c}_i) = \mathsf{Enc}(x_i) \cdot \mathsf{Enc}(-\tilde{c}_i)$ for $i \in [2, k-1]$, where $\tilde{c}_i$ denotes ith least significant bit of $\tilde{c}$,
2. $\mathsf{Enc}(r_1) = \mathsf{Enc}(x_1 - \tilde{c}_1 - \tilde{c}_0 r_0)$ (if $k > 1$) as $\mathsf{Enc}(x_1) \cdot \mathsf{Enc}(c_1) \cdot \mathsf{Enc}(r_0)^{-1}$ if $\tilde{c}_0 = 1$ and $\mathsf{Enc}(x_1) \cdot \mathsf{Enc}(c_1)$ otherwise, and
3. $\mathsf{Enc}(r_0) = \mathsf{Enc}(x_0 \oplus \tilde{c}_0)$ as $\mathsf{Enc}(x_0) \cdot \mathsf{Enc}(0)$ if $\tilde{c}_0 = 0$ and $\mathsf{Enc}(1 - x_0) = \mathsf{Enc}(x_0)^{-1} \cdot \mathsf{Enc}(1)$ otherwise

using fresh randomness for each newly formed encryption. Note that as a result of this computation the value that r_i takes may no longer be a bit (e.g., when $x_i = 0$ and $\tilde{c}_i = 1$ for $i \geq 2$). For each $i \in [0, k-1]$, if $r_{(1,i)} = 0$, S_1 computes $z_{(2,i)} = \mathsf{Enc}(r_i) \cdot \mathsf{Enc}(0)$ and otherwise computes $z_{(2,i)} = \mathsf{Enc}(1 - r_i) = \mathsf{Enc}(r_i)^{-1} \cdot \mathsf{Enc}(1)$ (using a freshly formed encryption of 0 or 1). S_1 then sends each $z_{(2,i)}$ to P_1 and simulates $\mathsf{PK12}$'s as a prover. During step 2, S_1 uses MalMul's simulator to produce P_1's view. In step 3, S_1 follows the protocol similar to P_2's computation: it receives z'_1, verifies P_1's $\mathsf{RangeProof}$ (extracting r'_1), produces r'_2 and its corresponding ciphertext, and acts as a prover in $\mathsf{RangeProof}$. S_1 skips step 4. In step 5, S_1 outputs $\tilde{c} + 2^{\ell+\kappa} - 2^k r'_1$ as decrypted value. As a result, in the consecutive step $\mathsf{MalBitAdd}$ will be called on k least significant bits of $\tilde{c}$ and k ciphertexts $\mathsf{Enc}(r_i)$. In step 6, S_1 uses $\mathsf{MalBitAdd}$'s simulator to interact with P_1, but introduces changes in the simulation. In particular, S_1 forces each encrypted carry bit (for $i \geq 1$) to become 0 as follows. The computation in $\mathsf{MalBitAdd}$ consists of computing $p_i = x_i + y_i - 2x_iy_i$ and $g_i = x_iy_i$ for each bit i of inputs x and y, followed by computing carry bits as $e_0 = g_0$ and $e_i = g_i + p_ie_{i-1}$ for $i \in [1, k-1]$ (the sum bits are computed from x_i's, y_i's and e_i's as previously described). Because one of the arguments to $\mathsf{MalBitAdd}$ is given in the plaintext form, computation of p_i's and g_i's is local and beyond the simulator's control. Computing each e_i (for $i \geq 1$), however, involves a call to MalMul, where we instruct S_1 to deviate from MalMul's simulation. In particular, when S_1 simulates P_1's view during a call to $\mathsf{MalMul}(\mathsf{Enc}(p_i), \mathsf{Enc}(e_{i-1}))$, instead of setting z_2 to $\mathsf{Enc}(p_ie_{i-1})^{-1}\mathsf{Enc}(e_{i-1})^{w-r_1}$ in step 2 as MalMul's simulation prescribes, S_1 sets z_2 to $\mathsf{Enc}(g_i)\mathsf{Enc}(e_{i-1})^{w-r_1}$. This will cause the product to evaluate to $-g_i$ and consequently result in e_i being 0 for each $i \geq 1$. (We note that e_i's are not computed sequentially in BitAdd to reduce the number of rounds, but this does not affect how we instruct the simulator to work.) The remaining multiplications are simulated according to MalMul's simulator.

Now we show that P_1's output at the end of simulation is computed correctly. During the simulation, S_1 sets each $r_{2,i}$ such that $r_{(2,i)} = r_i \oplus r_{(1,i)}$ and consequently $r_i = r_{(1,i)} \oplus r_{(2,i)}$, where $r_i = x_i - \tilde{c}_i$ for $i \geq 2$, $r_0 = x_0 \oplus \tilde{c}_0$, and $r_1 = x_1 - \tilde{c}_1 - \tilde{c}_0r_0$. Then because k least significant bits of $c = \tilde{c} + 2^{\ell+\kappa} - 2^kr'_1$ correspond to k least significant bits $\tilde{c}$, $\mathsf{MalBitAdd}$ is going to be called on arguments $(\tilde{c}_{k-1}, \ldots, \tilde{c}_0)$ and $(\mathsf{Enc}(r_{k-1}), \ldots, \mathsf{Enc}(r_0))$. As part of $\mathsf{MalBitAdd}$ P_1 then computes the carry bit e_0 as $\tilde{c}_0r_0$, while all other carry bits e_i for $i \geq 1$ will be forced to be 0 (by changing what MalMul returns) as described earlier. Recall that the output bits of $\mathsf{MalBitAdd}$ (and the output bits of BitDec are computed as $s_0 = \tilde{c}_0 + r_0 - 2e_0$ and $s_i = \tilde{c}_i + r_i + e_{i-1} - 2e_i$. Because all $e_i = 0$ for $i \geq 1$, but e_0 can be set to 1, we obtain that

1. $s_0 = \tilde{c}_0 + r_0 - 2\tilde{c}_0r_0 = \tilde{c}_0 \oplus r_0 = x_0$ as required;
2. s_1 is supposed to be computed as $s_1 = \tilde{c}_1 + r_1 + e_0 - 2e_1$, but we instead have

$s_1 = \tilde{c}_1 + r_1 + e_0$. Recall, however, that r_1 was set to $r_1 = x_1 - \tilde{c}_1 - \tilde{c}_0 r_0 = x_1 - \tilde{c}_1 - e_0$, which gives us $s_1 = \tilde{c}_1 + x_1 - \tilde{c}_1 - e_0 + e_0 = x_1$ as required;

3. s_i for $i \geq 2$ becomes $\tilde{c}_i + r_i$ as a result of S_1's simulation. Because r_i was set to $x_i - \tilde{c}_i$, we obtain that $s_i = \tilde{c}_i + x_i - \tilde{c}_i = x_i$ as required as well.

The last piece that we wanted to demonstrate is that P_1 will compute each r_i according to the value that S_1 expects even when r_i is not a bit (which would be a violation of the real protocol execution). During the simulation, r_0 is always computed as a bit, r_1 may take values -1 and -2 (in addition to 0 and 1), and r_i may take value -1 (in addition to 0 and 1). S_1 sets each $r_{2,i}$ as XOR of r_i and $r_{(1,i)}$ using the formula $r_i + r_{(1,i)} - 2r_i r_{(1,i)}$ (i.e., $r_{(2,i)}$ is either r_i or $1 - r_i$ based on the value of the bit $r_{(1,i)}$) and later P_1 computes $r_i = r_{(1,i)} + r_{(2,i)} - 2r_{(1,i)} r_{(2,i)}$. The crucial fact that we are using here is that $r_i \oplus r_{(1,i)} \oplus r_{(1,i)} = r_i$ for any value of r_i as long as $r_{(1,i)}$ is a bit. In other words, during the simulation $r_{(2,i)} = r_i$ and then $r_i = r_{(2,i)}$ when $r_{(1,i)} = 0$; and $r_{(2,1)} = 1 - r_i$ and then $r_i = 1 - r_{(2,i)} = 1 - (1 - r_i) = r_i$ when $r_{(1,i)} = 1$. We conclude that P_1 learns the correct (encrypted) output bits $x_0, \ldots, x_{k-1}$ as a result of this simulation.

To show that the view simulated by S_1 is indistinguishable from P_1's view in the hybrid model, we need to show the plaintext value the simulator produces in step 5 is indistinguishable from the value c in real execution (as the remaining building blocks have been shown to guarantee indistinguishability and all encrypted values achieve indistinguishability as well). Recall that in the real protocol execution c is formed as $2^{\ell+\kappa+1} - r + x = 2^{\ell+\kappa+1} - 2^k(r'_1 + r'_2) - \sum_{i=0}^{k-1} 2^i r_i + x$, while in the simulation S_1 outputs $c = \tilde{c} + 2^{\ell+k} - 2^k r'_1$. Let $\tilde{c} = 2^{\ell+\kappa} - 2^k r'_2 - \sum_{i=0}^{k-1}$. Because no information about r'_2 and r_i's is available to P_1, $\tilde{c}$ in the simulation and $2^{\ell+\kappa} - 2^k r'_2 - \sum_{i=0}^{k-1}$ during real execution are distributed identically. We obtain that during the real execution P_1 observes $2^{\ell+\kappa+1} - r + x$, while during the simulation P_1 observes $2^{\ell+\kappa+1} - r$. These two values are statistically indistinguishable using statistical security parameter κ. Note that we have to take the value of r'_1 into account when forming c during the simulation to ensure that c that the simulator outputs falls in the correct range (according to P_1's knowledge of r'_1). Lastly, we note that both during the simulation and real execution the computation aborts only when a malicious party fails to correctly complete ZKPKs as the prover. Therefore, simulated and real views are indistinguishable. □

4.6.2.7 Performance of the New Building Blocks

With MalMul, MalLT, RangeProof, MalTruncPR, MalInv, MalPreMul, MalBitDec, and previously mentioned prior work, we achieve security of the HMM and GMM protocols in the malicious model in the homomorphic encryption setting. The complexities of the new protocols are provided in Table 4.1.

TABLE 4.1

COMPLEXITY OF BUILDING BLOCKS IN THE TWO-PARTY SETTING BASED ON HOMOMORPHIC ENCRYPTION

Protocol	Rounds	Communication size	Computation complexity	
			Client	Server
MalMul	2	$13C + 2D$	15	15
MalLT	4	$(52.5 + 36(\ell + \kappa))C + 2D$	$43 + 33(\ell + \kappa)$	$43 + 33(\ell + \kappa)$
MalTruncPR	5	$(23k + 12(\ell + \kappa - k)$ $+2)C + (2k + 2)D$	$4 + 11(\ell + \kappa$ $-k) + 22k$	$4 + 11(\ell + \kappa$ $-k) + 22k$
MalInv	2	$18C + 2D$	18	18
MalPreMul	6	$44kC + 6kD$	$45k - 2$	$45k - 2$
MalBitAdd	$2 \log k$	$13k \log kC + 2k \log kD$	$k(15 \log k + 2)$	$k(15 \log k + 2)$
MalBitDec	$2 \log k + 4$	$((13 \log k + 23)k$ $+12(\ell + \kappa) - 11)C$ $+(2 \log k + 2)kD$	$k(15 \log k + 24)$ $+11(\ell + \kappa)$ -10	$k(15 \log k + 24)$ $+11(\ell + \kappa)$ -9

CHAPTER 5

DNA COMPUTATIONS

Computation based on genomic data is becoming increasingly popular today, be it for medical or other purposes such as ancestry or paternity testing. Non-medical uses of genomic data in a computation often take place in a server-mediated setting where the server offers the ability for joint genomic testing between the users. Undeniably, genomic data is highly sensitive, which in contrast to other biometry types, discloses a plethora of information not only about the data owner, but also about his or her relatives. Thus, there is an urgent need to protect genomic data, especially when it is used in computation for what we call as recreational non-health-related purposes. Towards this goal, in this work we put forward a framework for server-aided secure two-party computation with the security model motivated by genomic applications.

In this chapter, we talk about motivation and contributions in Sections 5.1 and 5.2. Section 5.3 provides necessary information about the selected genomic tests. Sections 5.4 and 5.5 describe the security model and our proposed solutions. Then Section 5.6 provides private genomic computations based on the proposed protocols. Experimental evaluation of our secure genomic tests is given in Section 5.7.

5.1 Motivation

The motivation for this work comes from rapidly expanding availability and use of genomic data in a variety of applications and the need to protect such highly sensitive data from potential abuse. The cost of sequencing one's genome has dramatically decreased in the recent years and is continuing to decrease, which makes such data more readily available for a number of applications. Examples of such applications include:

- Personalized medicine, where genomic tests are performed prior to prescribing a drug treatment to ensure its effectiveness;
- Paternity testing, which use DNA data to determine whether one individual is the father of another individual;
- Genomic compatibility tests, which allow potential or current partners to determine whether their future children are likely to inherit genetic conditions;
- Determining ancestry and building genealogical trees by examining DNA data of many individuals and finding relationships among specific individuals.

Genomic tests are increasingly used for medical purposes to ensure the best treatment. A number of services for what we call the "leisure" use of DNA data has flourished as well (examples are [1–3, 66]) allowing for various forms of comparing DNA data, be it for the purposes of building ancestry trees, genomic compatibility or other.

It is clear that DNA is highly sensitive and needs to be protected from unintended uses. It can be viewed as being even more sensitive than other types of biometry associated with an individual, as not only it allows for unique identification of an individual, but it also allows to learn a plethora of information about the individual such as predisposition to medical conditions and relatives of the individual thus exposing information about others as well. Furthermore, our understanding of genomes is continuously growing and exposure of DNA data now can lead to consequences which we cannot even anticipate today.

5.2 Contributions

The first observation we make about such types of genomic computation is that they are normally facilitated by some service or third party. For example, both ancestry and gene-based matchmaking web sites allow participants to interact with each other through the service provider. Such service providers serve as a natural point for aiding the individuals with private computation on their sensitive genomic data. In some prior publications on genomic computation (e.g., [20]), it is assumed that computation such as paternity testing or genetic compatibility is run between a client and a server, while we believe that it is more natural to assume that such computation is carried out by two individuals through some third-party service provider. Thus, in this work we look at private genomic computation in the light of server-mediated setting and utilize the server to lower the cost of the computation for the participants. Throughout this chapter, we will refer to the participants as Alice (A), Bob (B), and the server (S).

From the security point of view, participants in a protocol that securely evaluates a function are normally assumed to be either semi-honest or malicious. In our application domain, we may want to distinguish between different security settings depending on how well Alice and Bob know each other. For example, if Alice and Bob are relatives and would like to know how closely they are related (i.e., how closely their genealogical trees overlap), it would be reasonable to assume that they will not deviate from the prescribed computation in the attempt to cheat each other, i.e., they can be assumed to be semi-honest. On the other hand, if Alice and Bob meet each other through a matchmaking web site and do know each other well, it is reasonable for them to be cautious and engage in a protocol that ensures security (i.e., correctness and privacy) even in presence of malicious participants. The server can typically be expected not to deviate from its prescribed behavior, as it would lose its reputation and consequently revenue if any attempts at cheating become known. If, however, adding protection against server's malicious actions is not very costly, it can also be meaningful to assume a stronger security model.

Another important consideration from a security point of view is enforcing correct inputs to be entered in the computation when, for instance, the inputs are certified by some authority. This requirement is outside the traditional security model for secure multi-party computation (even in presence of fully malicious actors), and to the best of our knowledge certified inputs were previously considered only for specific functionalities such as private set intersection [42, 55] or anonymous credentials and certification [43], but not for general secure function evaluation. We bring this up in the context of genomic computation because for certain types of genomic computation it is very easy for one participant to modify his inputs and learn sensitive information about genetic conditions of the other participant. For example, genetic compatibility tests evaluate the possibility of two potential or existing partners to determine the possibility of transmitting to their children a genetic disease. Such possibility is present when both partners are (silent) carriers of that disease (see Section 5.3 for more detail). Then if the partners can each separately evaluate their DNA for a fingerprint of specific disease, the joint computation can consist of a simple AND of the bits provided by both parties (for one or more conditions). Now if a malicious participant sets all of his input bits to 1 and the outcome is positive, the participant learns that the other party is a carrier for a specific medical condition (or at least one condition from the set of specific conditions). We thus would like prevent malicious participants from modifying their inputs used in genomic computation in cases such data can be certified by certification authorities such as medical facilities.

The aspect of secure computation related to security properties that we treat in this work is fairness. In particular, it is known that full fairness cannot be achieved in the case of two-party computation in the malicious security model [49], but it becomes possible in the server-aided setting. Fairness has been considered in the server-aided literature in the past [71, 79] and achieving fairness only adds minimal overhead to the solutions in the settings we consider.

Thus, we categorize our contributions in two main groups: (i) results applicable to general secure function evaluation and (ii) results specific to genomic tests, both of which we consequently describe. All constructions rely on garbled circuit evaluation typically used in the two-party setting, but which we adopt to the three-party computation between the server and two users. We present from the simplest and enabling most efficient solutions to the most complex with added security guarantees.

1. Our most efficient solution is designed for the setting where A and B are semi-honest and S can be malicious (as in the ancestry testing scenario). In this setting, the solution consists of a single circuit garbling and single evaluation of the garbled circuit and the need for oblivious transfer is eliminated all together.

2. Our second solution works in the setting where A and B can be malicious, but S is semi-honest (applicable to the paternity test) and achieves fairness for A and B. In this solution, the combined work for all participants is approximately the same as the combined work of two participants in a two-party protocol based on garbled circuit in presence of semi-honest participants only.

3. Our last solution strengthens the model of malicious A and B with input certification (applicable to the genomic compatibility test). In more detail, in addition to being able to behave arbitrarily, A and B may maliciously modify their true inputs. To combat this, the function f being evaluated is modified to mark any suitable subset of the inputs as requiring certification. At the time of secure function evaluation, A and B have to prove that the inputs that they enter in the protocol are identical to the values signed by a trusted authority (a medical facility that performs genomic tests in our case). Achieving this involves the use of additional tools such as a signature scheme and ZKPKs. Handling of the remaining inputs and the rest of the computation is not affected by the shift to a stronger security model.

All of our constructions offer conceptual simplicity and at the same time achieve highly attractive performance. To the best of our knowledge, the strongest of our models which enforces correctness of the inputs have not been treated in the context of general secure multi-party computation and computation based on garbled circuit in particular. Despite the drastic differences in the techniques for garbled circuit evaluation and data certification, we show how they can be integrated by using oblivious transfer as the connecting point or even when oblivious transfer is not used.

Based on the solutions described above, we build implementations of three genetic tests, namely, genetic common ancestry, paternity, and genetic compatibility tests.

Each test uses a different security setting. We show through experimental results that each of the implemented tests is efficient with the worst runtime being on the order of a couple of seconds. The performance favorably compares to the state of the art (as detailed in Section 5.7), in some cases achieving orders of magnitude performance improvement over existing solutions.

5.3 Genomic Testing

Genomes represent complete hereditary information of an individual. Information extracted from one's genome can take different forms. One type is called Single Nucleotide Polymorphisms (SNPs), each of which corresponds to a well known variation in a single nucleotide.[1] Because SNP mutations are often associated with how one develops diseases and responds to treatments, they are commonly used in genetic disease and disorder testing. The same set of SNPs (i.e., nucleotides in the same positions) would be extracted for each individual, but the values associated with each SNP differ from one individual to another. Normally each SNP is referenced by a specific index and its value in a individual is represented as a bit, while representations consisting of 3 values 0, 1, 2 are used as well.

Another type of data extracted from a genome is based on Short Tandem Repeats (STRs). STRs occur when a short region consisting of two or more nucleotides is repeated and the occurrences are adjacent to each other. Unrelated individuals are likely to have a different number of repeats of a given STR sequence in certain regions in their DNA and thus STRs are often used for identity testing or testing between close relatives (such as paternity testing).

Paternity test. This test is normally done based on STRs. STR profile of a person consists of an ordered sequence of N 2-element sets $S = \langle \{x_{1,1}, x_{1,2}\}, \{x_{2,1}, x_{2,2}\},$

[1]A nucleotide can be viewed as a simple unit represented by a letter A, C, G, or T.

$\ldots, \{x_{N,1}, x_{N,2}\}\rangle$, where each value corresponds to the number of repeats of a specific STR sequence at specific locations in the genome. For each STR i, one of $x_{i,1}$ and $x_{i,2}$ is inherited from the mother and the other from the father.

Thus in the paternity test with a single parent, there are two STR profiles $S = \langle\{x_{i,1}, x_{i,2}\}\rangle$ and $S' = \langle\{x'_{i,1}, x'_{i,2}\}\rangle$ corresponding to the child and the contested father, respectively. To determine whether S' corresponds to the father's child, the test computes whether for each i the child's set $\{x_{i,1}, x_{i,2}\}$ contains (at least) one element from the contested father's set $\{x'_{i,1}, x'_{i,2}\}$. In other words, the test corresponds to the computation

$$\bigwedge_{i=1}^{N}[\{x_{1,i}, x_{2,i}\} \cap \{x'_{1,i}, x'_{2,i}\} \neq \emptyset] = \text{True} \tag{5.1}$$

when testing with both parents is performed, for each STR i one of $x_{i,1}$ and $x_{i,2}$ must appear in the mother's set and the other in the father's set. Using both parents' profiles in the computation increases the accuracy of the test, but even the single parent test has high accuracy for a small number N of well-chosen STRs (e.g., the US CODIS system utilizes $N = 13$, while the European SGM Plus identification method uses $N = 10$).

Genetic compatibility test. While there is a variety of genetic tests that can be used for several purposes, we concentrate on the genetic compatibility test where potential (or existing) partners would like to determine the possibility of transmitting to their children a genetic disease with Mendelian inheritance. In particular, if a specific mutation occurs in one allele[2] (called minor), it often has no impact on one's quality of life, but when the mutation occurs in both alleles (called major), the disease manifests itself in severe forms. If both partners silently carry a single mutation, they have a noticeable chance of conceiving a child carrying the major variety. Thus, a

[2]An allele is one of the alternative versions of a gene at a given location.

genetic compatibility test for a given genetic disease would test for the presence of minor mutations in both partners.

The current practice for screening for most genetic diseases consists of testing one SNP in a specific gene. It is, however, expected that in the future tests for more complex diseases (that involve multiple genes and mutations) will become available. Thus, a genetic disease can be characterized by a set of SNP indices and the corresponding values $(i_1, b_1), \ldots, (i_t, b_t)$, where i_j is the SNP index and $b_j \in \{0, 1\}$ is the value it takes. Then if the same values are found in the appropriate SNPs of an individual, we assume that the individual is tested as positive (i.e., the individual is the disease carrier). If both partners test as positive, then the outcome of the genetic compatibility test will be treated as positive and otherwise it is negative.

Ancestry test. There are a number of tests that allow for various forms of ancestry testing, for example, tests using Y-chromosome STRs (applicable to males only), mitochondrial DNA (mtDNA) test on the maternal line, and more general SNP-based tests for common ancestry or one's genealogy. Many such tests are not standardized and in addition current ancestry and genealogy service providers often use proprietary algorithms. The advantage of STR-based tests is that normally only a relatively small number of STRs are tested, while SNP-based tests often utilize a large number of (or even all available) SNPs, but more distant ancestry can be learned from SNP-based tests. For improved accuracy it is also possible to perform one type of testing after the other. In either case, to determine the most recent common ancestor between two individuals, the markers from the two individuals are compared and their number determines how closely the individuals are related. Certain tests such as determining geographical regions of one's ancestors normally require genetic data from many individuals.

5.4 Security Model

We formulate security using the standard ideal/real model for secure multi-party computation, where the view of any adversary in the real protocol execution should be indistinguishable from its view in the ideal model where a trusted party (TP) evaluates the function. Because the server does not contribute any input, it is meaningful to consider that either A or B is honest since the goal is to protect the honest party.

We are primarily interested in the setting where the server is semi-honest, but parties A and B may either be semi-honest or fully malicious. Thus, we target security models where S complies with the computation, with the exception of the first setting of semi-honest A and B, where we get security in the presence of a malicious server for free. We similarly assume that the server will not collude with users (putting its reputation at risk) or let users affect its operation.

We obtain security settings where (1) A and B can be corrupted by a semi-honest adversary, while S can act on behalf of a fully malicious adversary and (2) A and B can be malicious, but the server is semi-honest. Because we assume that the parties (or the adversaries who corrupt them) do not collude, at any given point of time there might be multiple adversaries, but they are independent of each other. This is similar to the setting used in [78, 79]. We note that based on the security settings listed above, at most one adversary would be fully malicious. In other words, if in (2) A is malicious, the goal is to protect B who is assumed to not be malicious and S is semi-honest, while in (1) S can be malicious, while A and B are semi-honest. Kamara et al. [78], however, show that in the presence of non-cooperating adversaries who corrupt only one party, showing security can be reduced to showing that the protocol is secure in the presence of semi-honest adversaries only, followed by proving for each malicious adversary $\mathcal{A}_i$ that the solution is secure in the presence of $\mathcal{A}_i$ when all other parties are honest. More precisely, we rely on the following lemma:

Lemma 1 ([78]) *If a multi-party protocol* Π *between* n *parties* $P_1, \ldots, P_{np}$ *securely computes* f *in the presence of (i) independent and semi-honest adversaries and (ii) a malicious* $\mathcal{A}_i$ *and honest* $\{\mathcal{A}_j\}_{j \neq i}$*, then* Π *is also secure in the presence of an adversary* $\mathcal{A}_i$ *that is non-cooperative with respect to all other semi-honest adversaries.*

This implies that in our setting (2) a solution secure in the presence of malicious A or B will also remain secure when A and B are corrupted by two independent malicious adversaries.

To model fairness, we modify the behavior of the TP in the ideal model to send $\perp$ to all parties if any party chooses to abort (note that fairness is only applicable to A and B). We assume that A and B learn the result of evaluation of a predefined function f that takes input x_1 from A and x_2 from B, and the server learns nothing. Because our primary motivation is genomic computation, we consider single-output functions, i.e., both A and B learn $f(x_1, x_2)$ (but two of our constructions support functions where A's and B's outputs differ and the remaining protocol in the present form loses only fairness).

Execution in the real model. The execution of protocol Π in the real model takes place between parties A, B, S and a subset of adversaries $\mathcal{A}_A$, $\mathcal{A}_B$, $\mathcal{A}_S$ who can corrupt the corresponding party. Let $\mathcal{A}$ denote the set of adversaries present in a given protocol execution. A and B receive their respective inputs x_i and a set of random coins r_i, while S receives only a set of random coins r_3. All parties also receive security parameter 1^κ. Each adversary receives all information that the party it corrupted has and a malicious adversary can also instruct the corresponding corrupted party to behave in a certain way. For each $\mathcal{A}_X \in \mathcal{A}$, let $\text{VIEW}_{\Pi,\mathcal{A}_X}$ denote the view of the adversary $\mathcal{A}_X$ at the end of an execution of Π. Also let $\text{OUT}^{\text{hon}}_{\Pi,\mathcal{A}}$ denote the output of the honest parties (if any) after the same execution of the protocol. Then for each $\mathcal{A}_X \in \mathcal{A}$, we define the partial output of a real-model execution of Π between A, B, S in the presence of $\mathcal{A}$ by $\text{REAL}_{\Pi,\mathcal{A}_X}(\kappa, x_1, x_2, r_1, r_2, r_3) \stackrel{\text{def}}{=} \text{VIEW}_{\Pi,\mathcal{A}_X} \cup \text{OUT}^{\text{hon}}_{\Pi,\mathcal{A}}$.

Execution in the ideal model. In the ideal model, all parties interact with a TP party who evaluates f. Similar to the real model, the execution begins with A and B receiving their respective inputs x_i and each party (A, B, and S) receiving security parameter 1^κ. Each honest (semi-honest) party sends to the TP $x'_i = x_i$ and each malicious party can send an arbitrary value x'_i to the TP. If x_1 or x_2 is equal to $\perp$ (empty) or if the TP receives an abort message, the TP returns $\perp$ to all participants. Otherwise, A and B receive $f(x'_1, x'_2)$. Let $\mathrm{OUT}^{\mathrm{hon}}_{f,\mathcal{A}}$ denote the output returned by the TP to the honest parties and let $\mathrm{OUT}_{f,\mathcal{A}_X}$ denote the output that corrupted party $\mathcal{A}_X \in \mathcal{A}$ produces based on an arbitrary function of its view. For each $\mathcal{A}_X \in \mathcal{A}$, the partial output of an ideal-model execution of f between A, B, S in the presence of $\mathcal{A}$ is denoted by $\mathrm{IDEAL}_{f,\mathcal{A}_X}(\kappa, x_1, x_2) \overset{\mathrm{def}}{=} \mathrm{OUT}_{f,\mathcal{A}_X} \cup \mathrm{OUT}^{\mathrm{hon}}_{f,\mathcal{A}}$.

Definition 3 (Security) *A three-party protocol* Π *between A, B, and S securely computes* f *if for all sets of probabilistic polynomial time (PPT) adversaries* $\mathcal{A}$ *in the real model, for all* x_i *and* $\kappa \in \mathbb{Z}$*, there exists a PPT transformation* $\mathcal{S}_X$ *for each* $\mathcal{A}_X \in \mathcal{A}$ *such that* $\mathrm{REAL}_{\Pi,\mathcal{A}_X}(\kappa, x_1, x_2, r_1, r_2, r_3) \overset{c}{\approx} \mathrm{IDEAL}_{f,\mathcal{S}_X}(\kappa, x_1, x_2)$*, where each* r_i *is chosen uniformly at random and* $\overset{c}{\approx}$ *denotes computational indistinguishability.*

To model the setting where some of the inputs of A and/or B are certified, we augment the function f to be executed with the specification of what inputs are to be certified and two additional inputs y_1 and y_2 that provide certification for A's and B's inputs, respectively. Then in the ideal model execution, the TP will be charged with additionally receiving y_i's. If the TP does not receive all inputs or if upon receiving all inputs some inputs requiring certification do not verify, it sends $\perp$ to all parties. In the real model execution, verification of certified inputs is built into Π and besides using two additional inputs y_1 and y_2 the specification of the execution remains unchanged.

Definition 4 (Security with certified inputs) *A three-party protocol* Π *between A, B, and S securely computes* f *if for all sets of PPT adversaries* $\mathcal{A}$ *in the real model,* *for all x_i, y_i, and $\kappa \in \mathbb{Z}$, there exists a PPT transformation $\mathcal{S}_X$ for each $\mathcal{A}_X \in \mathcal{A}$ such that* $\text{REAL}_{\Pi,\mathcal{A}_X}(\kappa, x_1, x_2, y_1, y_2, r_1, r_2, r_3) \overset{c}{\approx} \text{IDEAL}_{f,\mathcal{S}_X}(\kappa, x_1, x_2, y_1, y_2)$*, where each r_i is chosen uniformly at random.*

5.5 Server-Aided Computation

In this section we detail our solutions for server-aided two party computation based on garbled circuit. The current description is general and can be applied to any function f. In Section 5.6 we describe how these constructions can be applied to genomic tests to result in fast performance. For the rest of this chapter, t_1 (t_2) shows the number of A's (B's) input bits, and t_3 represents the number of output bits.

5.5.1 Semi-Honest A and B, Malicious S

Our first security setting is where A and B are semi-honest and S can be malicious. The main intuition behind the solution is that when A and B can be assumed to be semi-honest and a solution based on garbled circuit evaluation is used, we will charge S with the task of evaluating a garbled circuit. That is, security is maintained in the presence of malicious server because garbled circuit evaluation techniques are secure in the presence of a malicious evaluator. Next, we notice that if A and B jointly form garbled representation of the circuit for the function f they would like to evaluate, both of them can have access to the pairs of labels (ℓ_i^0, ℓ_i^1) corresponding to the input wires. Thus, they can simply send the appropriate label ℓ_i^b to S for evaluation purposes for their value of the input bit b for each input wire. This eliminates the need for OT and results in a solution that outperforms a garbled circuit protocol in the presence of only semi-honest participants. The same idea was sketched in [58] (with the difference that S was to learn the output). The use of a pseudo-random function $\mathsf{PRF} : \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^\kappa$ with security parameter κ for deriving wire labels in the scheme is as in [97].

Input: A has private input x_1, B has private input x_2, and S has no private input.
Output: A and B learn $f(x_1, x_2)$, S learns nothing.
Protocol 1:

1. A and B jointly choose $\delta \overset{R}{\leftarrow} \{0,1\}^{\kappa-1}$, $k \overset{R}{\leftarrow} \{0,1\}^{\kappa}$, and set $\Delta = \delta || 1$. They jointly produce m pairs of garbled labels as $\ell_i^0 = \mathsf{PRF}(k, i)$ and $\ell_i^1 = \ell_i^0 \oplus \Delta$ for $i \in [1, m]$, garble the gates to produce garbled circuit $\mathcal{G}_f$ for f, and send $\mathcal{G}_f$ to S.
2. For each $i \in [1, t_1]$, A locates the ith bit b_i of her input and sends to S the label $\ell_i^{b_i}$ of the corresponding wire i in the garbled circuit.
3. Similarly, for each bit $j \in [1, t_2]$, B locates the jth bit b_j of his input and sends to S the label $\ell_{i+t_1}^{b_j}$ of the corresponding wire $i + t_1$ in the garbled circuit.
4. S evaluates the circuit on the received inputs and returns to B the computed label ℓ_i^b for each output wire $i \in [m - t_3 + 1, m]$. B forwards all received information to A.
5. For each ℓ_i^b returned by S ($i \in [m - t_3 + 1, m]$), A and B do the following: if $\ell_i^b = \ell_i^0$, set $(i - m + t_3)$th bit of the output to 0, if $\ell_i^b = \ell_i^1$, set $(i - m + t_3)$th bit of the output to 1, otherwise abort.

A more detailed description of the solution is provided in Protocol 1. In what follows, let m denote the total number of wires in a circuit (including input and output wires), wires $1, \ldots, t_1$ correspond to A's input, wires $t_1 + 1, \ldots, t_1 + t_2$ correspond to B's input, and the last t_3 wires $m - t_3 + 1, \ldots, m$ correspond to the output wires. We also use κ to denote security parameter (for symmetric key cryptography). Notation $a \overset{R}{\leftarrow} U$ means that the value of a is chosen uniformly at random from the set U. The protocol is written to utilize the free XOR technique, where $\ell_i^0 \oplus \ell_i^1$ must take the same value Δ for all circuit wires i and the last bit of Δ is 1.

In Protocol 1 (which is also illustrated in Figure 5.1), the easiest way for A and B to jointly choose random values is for one party to produces them and communicate to the other party. In this solution, the combined work of A and B is linear in the size of the circuit for f. The work, however, can be distributed in an arbitrary manner as long as S receives all garbled gates (e.g., a half of $\mathcal{G}_f$ from A and the other half

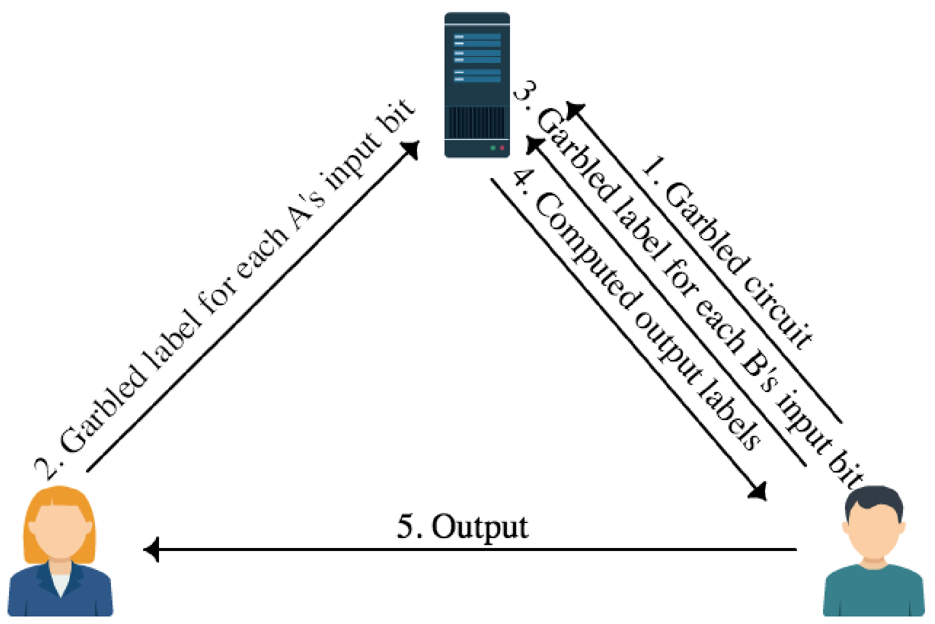

1. Garbled circuit
2. Garbled label for each A's input bit
3. Garbled label for each B's input bit
4. Computed output labels
5. Output

Figure 5.1. Illustration of Protocol 1 with weak A (who contributes only garbled labels for her input wires to the computation).

from B). Besides equally splitting the work of circuit garbling between the parties, an alternative possibility is to let the weaker party (e.g., a mobile phone user) to do work sublinear in the circuit size. Let A be a weak client, who delegates as much work as possible to B. Then B generates the entire garbled circuit and sends it to S, while A will only need to create t_1 label pairs corresponding to her input, to be used in step 2 of the protocol. Upon completion of the result, A learns the output from B (i.e., there is no need for A to know labels for the output wires). Thus, the work and communication of the weaker client is only linear in the input and output sizes. Security of this solution can be stated as follows:

Theorem 7 *Protocol 1 fairly and securely evaluates function f in the presence of semi-honest A and B and malicious S.*

Proof: Fairness is achieved based on the fact that A and B are semi-honest. This means that if B ever learns any output, he will share (the part of) the output he receives with the other party. Thus, either both of them learn the (possibly partial) output or neither party learns the output.

To show simulation-based security, we build three independent simulators $\mathcal{S}_S$, $\mathcal{S}_B$, and $\mathcal{S}_A$ for three independent adversaries $\mathcal{A}_S$, $\mathcal{A}_B$, and $\mathcal{A}_A$, respectively. Note that $\mathcal{A}_S$ can act in an arbitrary way while $\mathcal{A}_B$ and $\mathcal{A}_A$ are semi-honest.

We first consider a simulator $\mathcal{S}_S$ that communicates with malicious $\mathcal{A}_S$ pretending to be A and B (without access to their private inputs) in such a way that $\mathcal{S}_S$ is unable to distinguish it from an execution in the real model. $\mathcal{S}_S$ proceeds by garbling a circuit corresponding to f and sending it to $\mathcal{A}_S$. For each wire $i \in [1, t_1 + t_2]$, $\mathcal{S}_S$ sends label $\ell_i^{b_i}$ for a randomly chosen bit b_i. If $\mathcal{A}_S$ does not abort, $\mathcal{S}_S$ obtains from $\mathcal{A}_S$ a set of labels corresponding to the output wires. It is easy to see that the view simulated by $\mathcal{S}_S$ is indistinguishable from the view of $\mathcal{A}_S$ in the real model execution (since the view of garbled circuit evaluator is independent of the input on which the circuit is evaluated). Note that the simulator can safely use random inputs because S is not entitled to receiving any information about the result of function evaluation.

We next consider a simulator $\mathcal{S}_B$ for semi-honest $\mathcal{A}_B$. Note that $\mathcal{A}_B$'s advantage is maximized when B constructs the entire garbled circuit (or simply knows the random label pairs for all wires of the circuit), which we assume is the case. After $\mathcal{A}_B$ constructs the circuit and forms the input labels according to x_2, $\mathcal{S}_B$ queries the trusted party and obtains B's output $f(x_1, x_2)$. $\mathcal{S}_B$ then extracts from $\mathcal{A}_B$'s view the output labels corresponding to the received output $f(x_1, x_2)$ from the set of the label pairs for the output wires and sends them to $\mathcal{A}_B$. $\mathcal{S}_B$ also receives assembled $f(x_1, x_2)$ from $\mathcal{A}_B$ destined to A. This view is the same as the view of $\mathcal{A}_B$ in the real model execution given the same set of random coins.

A simulator for $\mathcal{A}_A$ is built analogously with the exception that $\mathcal{A}_A$ receives (from $\mathcal{S}_A$) $f(x_1, x_2)$ directly instead of learning labels for the output wires. $\square$

5.5.2 Semi-Honest S, Malicious A and B

To maintain efficiency of the previous solution by avoiding the cost of OT, we might want to preserve the high-level structure of the computation in the first solution. Now, however, because A and B can be malicious, neither of them can rely on the other party in garbling the circuit correctly. To address this, each of A and B may garble their own circuit for f, send it to S, and S will be in charge of evaluating both of them and performing a consistency check on the results (without learning the output). With this solution, A would create label pairs for her input bits/wires for both garbled circuit and communicate one set of pairs to B who uses them in constructing his circuit. What this achieves is that now A can directly send to S the labels corresponding to her input bits for circuit evaluation for both circuits. B performs identical operations. There is still no need to perform OT, but two security issues arise: (1) A and B must be forced to provide consistent inputs into both circuits and (2) regardless of whether the parties learn the output (e.g., whether the computation is aborted or not), a malicious party can learn one bit of information about the other party's input (by constructing a circuit that does not correspond to f) [75, 95]. While the first issue can be inexpensively addressed using the solution of [86] (which works in the presence of malicious users and semi-honest server), the second issue will still stand with this structure of the computation.

Instead of allowing for (1-bit) information leakage about private inputs, we change the way the computation takes place. If we now let the server garble the circuit and each of the remaining parties evaluate a copy of it, the need for OT (for both A and B's inputs) arises. We, however, were able to eliminate the use of OT for one of A and B and construct a solution that has about the same cost as a single garbled circuit solution in the semi-honest model. At a high-level, it proceeds as follows: A creates garbled label pairs (ℓ_i^0, ℓ_i^1) for the wires corresponding to her inputs only and sends them to S. S uses the pairs to construct a garbled circuit for f and sends it to B. S and B engage in OT, at the end of which B learns labels corresponding to his input bits. Also, A sends to B the labels corresponding to her input bits, which allows B to evaluate the circuit. We note that because A may act maliciously, she might send to B incorrect labels, which will result in B's inability to evaluate the circuit. This, however, is equivalent to A aborting the protocol. In either case, neither A nor B learn any output and the solution achieves fairness. Similarly, if B does not perform circuit evaluation correctly, neither party learns the output.

The next issue that needs to to addressed is that of fairly learning the output. We note that S cannot simply send the label pairs for the output wires to A and B as this would allow B to learn the output and deny A of this knowledge. Instead, upon completion of garbled circuit evaluation, B sends the computed labels to A. With the help of S, A verifies that the labels A possesses are indeed valid labels for the output wires without learning the meaning of the output. Once A is satisfied, she notifies S who sends the label pairs to A and B, both of whom can interpret and learn the result. We note that malicious A can report failure to S even if verification of the validity of the output labels received from B was successful. Once again, this is equivalent to A aborting the protocol, in which case neither party learns the output and fairness is maintained.

Our solution is given as Protocol 2 below (which is also illustrated in Figure 5.2) and uses a hash function $H : \{0,1\}^* \rightarrow \{0,1\}^\kappa$ that we treat as a random oracle. We show security of this solution in a hybrid model where the parties are given access to a trusted entity computing OT.

Theorem 8 *Protocol 2 fairly and securely evaluates function f in the presence of malicious A or B and semi-honest S in the hybrid model with ideal implementation of OT and where H is modeled as a random oracle.*

Proof: As before, we start by showing fairness and then proceed with security. The only way for A or B to learn any output is when A is satisfied with the verification

Input: A has private input x_1, B has private input x_2, and S has no private input.
Output: A and B learn $f(x_1, x_2)$, S learns nothing.
Protocol 2:

1. S chooses $\delta \overset{R}{\to} \{0,1\}^{\kappa-1}$, $k_1 \overset{R}{\to} \{0,1\}^{\kappa}$, $k_2 \overset{R}{\to} \{0,1\}^{\kappa}$ and sets $\Delta = \delta||1$. S sends Δ and k_1 to A.

2. S computes wire labels $\ell_i^0 = \mathsf{PRF}(k_1, i)$ for $i \in [1, t_1]$, $\ell_i^0 = \mathsf{PRF}(k_2, i - t_1)$ for $i \in [t_1 + 1, m]$, and sets $\ell_i^1 = \ell_i^0 \oplus \Delta$ for $i \in [1, m]$. S then constructs garbled gates $\mathcal{G}_f$ and sends $\mathcal{G}_f$ to B.

3. S and B engage in t_2 instances of 1-out-of-2 OT, where S assumes the role of the sender and uses t_2 label pairs $(\ell_{t_1+i}^0, \ell_{t_1+i}^1)$ for $i \in [1, t_2]$ corresponding to B's input wires as its input and B assumes the role of the receiver and uses his t_2 input bits b_i as the input into the protocol. As the result of the interaction, B learns garbled labels $\ell_{t_1+i}^{b_i}$ for $i \in [1, t_2]$.

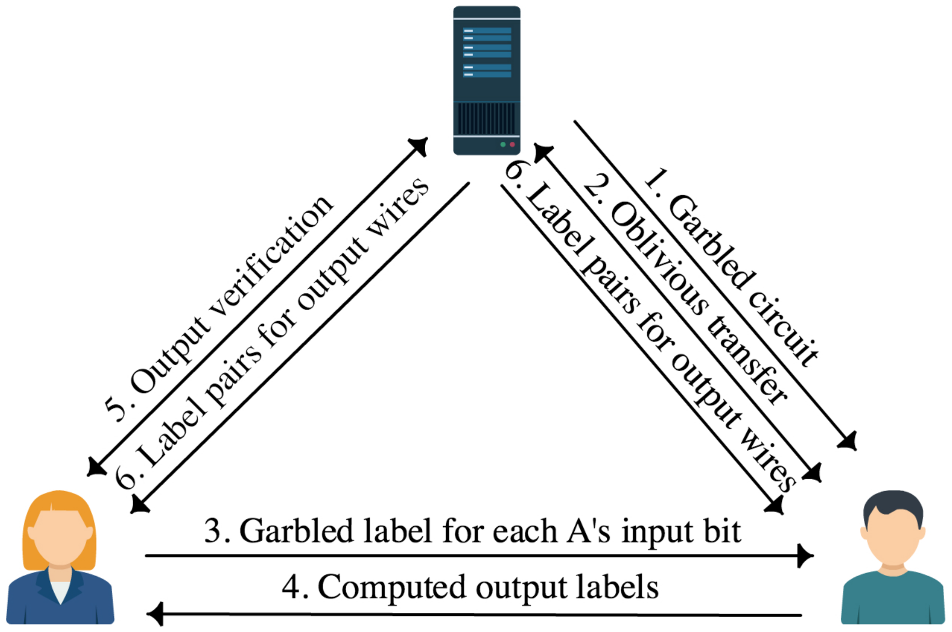
4. A computes labels $\ell_i^0 = \mathsf{PRF}(k_1, i)$ for $i \in [1, t_1]$ and sends to B $\ell_i^{b_i}$ for her input bits b_i, where $\ell_i^1 = \ell_i^0 \oplus \Delta$ for any $b_i = 1$.

5. After receiving the labels for his own and A's input, B evaluates the circuit, learns the output labels $\ell_i^{b_i}$ for $i \in [m - t_3 + 1, m]$ and sends them to A.

6. A requests from S output verification constructed as follows: For each output wire i, S computes $H(\ell_i^0), H(\ell_i^1)$, randomly permutes the tuple, and sends it to A.

7. For each label ℓ_i received from B in step 5, A computes $H(\ell_i)$ and checks whether the computed value appear among $H(\ell_i^b)$, $H(\ell_i^{1-b})$ received from S in step 6. If the check succeeds for all output wires, A notifies S of success

8. Upon receiving confirmation of success from A, S sends (ℓ_i^0, ℓ_i^1) for all output wires i to A and B, who recover the output.



1. Garbled circuit
2. Oblivious transfer
6. Label pairs for output wires
5. Output verification
6. Label pairs for output wires
3. Garbled label for each A's input bit
4. Computed output labels

Figure 5.2. Illustration of Protocol 2.

of the output labels she received from B. Recall that each received label ℓ_i is checked against $H(\ell_i^b), H(\ell_i^{1-b})$ for some bit b, where H is a collision-resistant hash function. The probability that this check succeeds for some ℓ_i that is not equal to ℓ_i^0 or ℓ_i^1 is negligible. Thus, A is guaranteed to possess the result of garbled circuit evaluation, at which point both parties are given access to the output.

We next construct simulators for all of the (independent) adversaries $\mathcal{A}_A$, $\mathcal{A}_B$, and $\mathcal{A}_S$. We start with a simulator $\mathcal{S}_A$ for malicious $\mathcal{A}_A$. $\mathcal{S}_A$ runs $\mathcal{A}_A$ and simulates the remaining parties. $\mathcal{A}_A$ produces t_1 random labels ℓ_i^0 and sends them to $\mathcal{S}_A$, while $\mathcal{S}_A$ chooses Δ and sends it to $\mathcal{A}_A$. If at least one label is of an incorrect bitlength, $\mathcal{S}_A$ aborts. If $\mathcal{S}_A$ did not abort, $\mathcal{A}_A$ sends t_1 labels to $\mathcal{S}_A$. If the ith label sent by $\mathcal{A}_A$ does not correspond to one of the labels in the ith pair of labels $(\ell_i^0, \ell_i^0 \oplus \Delta)$ corresponding to $\mathcal{A}_A$'s inputs, $\mathcal{S}_A$ aborts. If $\mathcal{S}_A$ did not abort, it interprets the meaning of the input labels received from $\mathcal{A}_A$ and stores the input as x_1'. At some point $\mathcal{S}_A$ creates a random label ℓ_i for each bit of the output and sends them to $\mathcal{A}_A$. Upon $\mathcal{A}_A$'s request,

$\mathcal{S}_A$ also chooses another random label ℓ'_i for each bit of the output. For each bit i of the output, $\mathcal{S}_A$ sends to $\mathcal{A}_A$ the pair $H(\ell_i), H(\ell'_i)$ in a randomly permuted order. If $\mathcal{A}_A$ notifies $\mathcal{S}_A$ of successful verification of the output labels, $\mathcal{S}_A$ queries the trusted party for the output $f(x'_1, x_2)$. For each ith bit b_i of the output, if $b_i = 0$, $\mathcal{S}_A$ sends to $\mathcal{A}_A$ the pair (ℓ_i, ℓ'_i), otherwise, $\mathcal{S}_A$ sends the pair (ℓ'_i, ℓ_i).

Now we examine the view of $\mathcal{A}_A$ in the real and ideal model executions and correctness of the output. After receiving the label pairs from $\mathcal{A}_A$, $\mathcal{S}_A$ performs the same checks on them as S would and thus both would abort in the same circumstances. Similarly, if $\mathcal{A}_A$ provides malformed labels for circuit evaluation, $\mathcal{S}_A$ will immediately detect this in the ideal model and abort, while B in the real world will be unable to evaluate the circuit and also abort. Otherwise, in both cases the function will be correctly evaluated on the input provided by $\mathcal{A}_A$ and B's input. In the remaining interaction, $\mathcal{A}_A$ sees only random values, which in the ideal world are constructed consistently with $\mathcal{A}_A$'s view in the real model execution. Thus, $\mathcal{A}_A$'s view is indistinguishable in the two executions.

Let us now consider malicious $\mathcal{A}_B$, for which we construct simulator $\mathcal{S}_B$ in the ideal model execution who simulates correct behavior of A and S. First, $\mathcal{S}_B$ simulates the OT. It records the input bits used by $\mathcal{A}_B$ during the simulation, which it stores as x'_2 and returns t_2 random labels to $\mathcal{A}_B$. $\mathcal{S}_B$ also sends another set of t_1 random labels to $\mathcal{S}_B$. $\mathcal{S}_B$ queries the trusted party for $\mathcal{A}_B$'s output $f(x_1, x'_2)$ and chooses a pair of random labels (ℓ_i^0, ℓ_i^1) for each bit i of the output. $\mathcal{S}_B$ gives to $\mathcal{A}_B$ a simulated garbled circuit (as described in [90]) so that the ith computed output label corresponds to the ith bit of $f(x_1, x'_2)$. If after circuit evaluation, $\mathcal{A}_B$ does not send the correct output labels to $\mathcal{S}_B$, $\mathcal{S}_B$ aborts the execution. Otherwise, $\mathcal{S}_B$ sends the pairs (ℓ_i^0, ℓ_i^1) to $\mathcal{A}_B$.

The only difference between the view of $\mathcal{A}_B$ during real model execution and the view simulated by $\mathcal{S}_B$ in the ideal model is that $\mathcal{A}_B$ evaluates a simulated circuit in the ideal model. Computational indistinguishability of the simulated circuit follows from the security proofs of Yao's garbled circuit construction [90]. Thus, $\mathcal{A}_B$ is unable to tell the two worlds apart.

It is also straightforward to simulate the view of semi-honest $\mathcal{A}_S$ because it contributes no input and receives no output. □

5.5.3 Semi-Honest S, Malicious A and B with Input Certification

We next consider an enhanced security setting in which malicious A and B are enforced to provide correct inputs in the computation. This enforcement is performed by requiring A and B to certify their inputs prior to protocol execution and prove the existence of certification on the inputs they enter.

The basic structure of our solution in this stronger security model remains the same as in Protocol 2, but we extend it with a novel mechanism for obliviously verifying correctness of the inputs. The intricate part of this problem is that signature schemes use public-key operations, while garbled circuit evaluation deals with randomly generated labels and symmetric key operations. In what follows, we describe the intuition behind our solution followed by more detailed explanation.

Suppose that the party whose inputs are to be verified participates in an OT protocol on her inputs as part of garbled circuit evaluation (i.e., the party is the circuit evaluator and acts as the receiver in the OT). Then if we use the variant of OT known as committed oblivious transfer (COT) (also called verifiable OT in some literature), the party will submit commitments to the bits of her input as part of OT computation and these commitments can be naturally tied to the values signed by a third party authority by means of ZKPKs (i.e., without revealing anything other than equality of the signed values and the values used in the commitments). Several COT schemes that we examined (such as in [77, 83]), however, had disadvantages in their performance and/or complex setup assumptions (such as requiring the sender and receiver to hold shares of the decryption key for a homomorphic public-key encryption scheme). We thus choose to integrate input certification directly with a conventional OT protocol by Naor and Pinkas [99].

Before we proceed with further description, we discuss the choice of the signature scheme and the way knowledge of a signature is proved. Between the main two candidates of signature schemes with protocols [37] and [38], we chose the one from [37] because it uses an RSA modulus. In application like ours, zero-knowledge statements are to be proved across different groups. This requires the use of statistically-hiding zero-knowledge proofs that connect two different groups through a setting in which the Strong RSA assumption (or, more generally, the difficulty of eth root extraction) holds [39, 51, 62]. Thus, the public key of the third party certification authority can be conveniently used as the common setup for other interaction between the prover and verifier. This has important implications on the use of such solutions in practice. (If multiple signatures are issued by multiple authorities, i.e., medical facilities in our application, one of the available public keys can be used to instantiate the common setup.)

Recall that in Protocol 2, B obtains the labels corresponding to his input from S via OT, while A knows all label pairs for her input wires and simply sends the appropriate labels to B. Now both of them have to prove to S that the inputs they enter in the protocol have been certified by a certain authority. For simplicity, in what follows we assume that all of A's and B's inputs are to be verified. (If this is not the case and only a subset of the inputs should be verified, the computation associated with input verification described below is simply omitted for some of the input bits.) Let us start with the verification mechanism for B, after which we treat the case of A.

B engages in the Naor-Pinkas OT in the role of the receiver. As part of OT, B forms two keys PK_0 and PK_1, where PK_σ is the key that will be used to recover m_σ. Thus, if we want to enforce that σ corresponds to the bit for which B has a

Input: Sender S has two strings m_0 and m_1, receiver R has a bit σ. Common input consists of prime p, generator $\hat{g}$ of subgroup of $\mathbb{Z}_p^*$ of prime order q, and a random element C from the group generated by $\hat{g}$ (chosen by S).
Output: R learns m_σ and S learns nothing.
Naor-Pinkas OT Protocol:

1. S chooses random $r \in \mathbb{Z}_q$ and computes C^r and $\hat{g}^r$.
2. R chooses $k \in \mathbb{Z}_q^*$, sets public keys $PK_\sigma = \hat{g}^k$ and $PK_{1-\sigma} = C/PK_\sigma$, and sends PK_0 to S.
3. After receiving PK_0, S computes $(PK_0)^r$ and $(PK_1)^r = C^r/(PK_0)^r$. S sends to R $\hat{g}^r$ and two encryptions $H((PK_0)^r, 0) \oplus m_0$ and $H((PK_1)^r, 1) \oplus m_1$, where H is a hash function (modeled as a random oracle).
4. R computes $H((\hat{g}^r)^k) = H((PK_\sigma)^r)$ and uses it to recover m_σ.

signature from a certification authority, B must prove that he knows the discrete logarithm of PK_σ where σ is the signed bit. More formally, the statement B has to prove in zero knowledge is $PK\{(\sigma, \beta) : \mathsf{Sig}(\sigma) \wedge y = \mathsf{Com}(\sigma) \wedge ((\sigma = 0 \wedge PK_0 = \hat{g}^\beta) \vee (\sigma = 1 \wedge PK_1 = \hat{g}^\beta))\}$. In other words, B has a signature on 0 and knows the discrete logarithm of PK_0 to the base $\hat{g}$ (i.e., constructed PK_0 as $\hat{g}^k$) or B has a signature on 1 and knows the discrete logarithm of PK_1 to the same base. Using a technically more precise PK statement for showing that σ is 0 or 1 would result in the PK statement above be re-written as $PK\{(\sigma, \alpha, \beta) : \mathsf{Sig}(\sigma) \wedge y = \mathsf{Com}(\sigma, \alpha) = g^\sigma h^\alpha \wedge ((y = h^\alpha \wedge PK_0 = \hat{g}^\beta) \vee (y/g = h^\alpha \wedge PK_1 = \hat{g}^\beta))\}$. We note that it is known how to realize this statement as a ZKPK as it uses only conjunction and disjunction of discrete logarithm-based sub-statements (see, e.g., [41]). Executing this ZKPK would allow S to verify B's input for a particular input wire if B has a signature on a bit. In practice, however, a signature is expected to be on messages from a larger space than $\{0, 1\}$ and thus a single signature will need to be used to provide inputs for several input wires in the circuit. This can be accomplished by, in addition to using a commitment on the signed message, creating commitments to the individual bits and showing that they correspond to the binary representation of the signed message. Then the commitments to the bits of the message are linked to the keys generated in each instance of the OT. More formally, the ZKPK statement for a t-bit signed value would become:

$$PK\{(\sigma, \sigma_1, \ldots, \sigma_t, \alpha, \alpha_1, \ldots, \alpha_t) : \mathsf{Sig}(\sigma) \wedge y = g^{\sigma} h^{\alpha} \wedge \\ y_1 = g^{\sigma_1} h^{\alpha_1} \wedge \ldots \wedge y_t = g^{\sigma_t} h^{\alpha_t} \wedge \sigma = \sum_{i=1}^{t} 2^{i-1} \sigma_i\} \quad (5.2)$$

$$PK\{(\sigma_i, \alpha_i, \beta_i) : y_i = g^{\sigma_i} h^{\alpha_i} \wedge ((y_i = h^{\alpha_i} \wedge PK_0^{(i)} = \hat{g}^{\beta_i}) \\ \vee (y_i/g = h^{\alpha_i} \wedge PK_1^{(i)} = \hat{g}^{\beta_i}))\}. \quad (5.3)$$

Notation $PK_0^{(i)}$ and $PK_1^{(i)}$ denotes the public keys used in the ith instance of Naor-Pinkas OT. [41] shows how to prove that discrete logarithms satisfy a given linear equation.

Furthermore, it is likely that signatures will contain multiple messages (e.g., a genetic disease name and the outcome of its testing). In those cases, multiple messages from a single signature can be used as inputs into the garbled circuit or, depending on the function f, there might be other arrangements. For instance, one message can be used to provide inputs into the circuit and another be opened or partially open. It is not difficult to generalize Equations 5.2 and 5.3 to cover such cases.

We now can proceed with the description of the mechanism for verifying A's inputs. Recall that for each bit i of her input, A has label pairs (ℓ_i^0, ℓ_i^1) and later sends to B the label $\ell_i^{b_i}$ corresponding to her input bit b_i. As before, consider first the case when A holds a signature on a single bit. To prove that the label $\ell_i^{b_i}$ sent to B corresponds to the bit for which she possesses a signature, we have A commit to the label $\ell_i^{b_i}$ and prove to S that either the commitment is to ℓ_i^0 and she has a signature on 0 or the commitment is to ℓ_i^1 and she has a signature on 1. Let the commitment be $c_i = \mathsf{Com}(\ell_i^{b_i}, r_i)$. Then if verification of the ZKPKs for each input bit was successful, S forwards each c_i to B together with the garbled circuit. Now when A sends her input label $\ell_i^{b_i}$ to B, she is also required to open the commitment c_i by sending r_i to B. B will proceed with circuit evaluation only if $c_i = g^{\hat{\ell}_i^{b_i}} h^{\hat{r}_i}$ for each bit i of A's input, where $\hat{\ell}_i^{b_i}$ and $\hat{r}_i$ are the values B received from A.

More formally, the statement A proves to S in ZK is $PK\{(\sigma, \alpha, \beta, \gamma) : \mathsf{Sig}(\sigma) \wedge y = g^\sigma h^\alpha \wedge z = g^\beta h^\gamma \wedge ((y = h^\alpha \wedge z/g^{\ell_i^0} = h^\gamma) \vee (y/g = h^\alpha \wedge z/g^{\ell_i^1} = h^\gamma))\}$. Similar to the case of B's input verification, this ZKPK can be generalized to use a single signature with multiple bits input into the circuit. More precisely, the statement in Equation 5.2 remains unchanged, while the second statement becomes:

$$\begin{aligned} PK\{(\sigma_i, \alpha_i, \beta_i, \gamma_i) : y_i = g^{\sigma_i} h^{\alpha_i} \wedge z_i = g^{\beta_i} h^{\gamma_i} \wedge \\ ((y_i = h^{\alpha_i} \wedge z_i/g^{\ell_i^0} = h^{\gamma_i}) \vee (y_i/g = h^{\alpha_i} \wedge z_i/g^{\ell_i^1} = h^{\gamma_i}))\}. \end{aligned} \tag{5.4}$$

We summarize the overall solution as Protocol 3 below.

The security of Protocol 3 can be stated as follows:

Theorem 9 *Protocol 3 fairly and securely evaluates function f in the presence of malicious A or B and semi-honest S in the hybrid model with ideal implementation of OT and where H is a hash function modeled as a random oracle and inputs of A and B are verified according to Definition 4.*

Because the structure of the computation in Protocol 3 is the same as in Protocol 2 and primarily only ZKPKs have been added (that have corresponding simulators in the ideal model), we do not mention the proof here.

5.6 Private Genomic Computation

For all types of genomic computation we assume that A has information extracted from her genome, which she privately stores. Similarly, B stores data associated with his genome. A and B may enter some or all of their data into the computation and

Input: A has private input x_1 and signature $\mathsf{Sig}(x_1)$, B has private input x_2 and $\mathsf{Sig}(x_2)$, and S has no private input.
Output: A and B learn $f(x_1, x_2)$, S learns nothing.
Protocol 3:

1. (a) S chooses $\delta \overset{R}{\to} \{0,1\}^{\kappa-1}$, $k_1 \overset{R}{\to} \{0,1\}^{\kappa}$, $k_2 \overset{R}{\to} \{0,1\}^{\kappa}$ and sets $\Delta = \delta||1$. S sends Δ and k_1 to A. S also computes labels $\ell_i^0 = \mathsf{PRF}(k_1, i)$ and $\ell_i^1 = \ell_i^0 \oplus \Delta$ for $i \in [1, t_1]$.

   (b) A computes labels $\ell_i^0 = \mathsf{PRF}(k_1, i)$ and $\ell_i^1 = \ell_i^0 \oplus \Delta$ for $i \in [1, t_1]$. For each bit b_i of her input, A commits $c_i = \mathsf{Com}(b_i, r_i)$ and $c_i' = \mathsf{Com}(\ell_i^{b_i}, r_i')$ using fresh randomness r_i and r_i'. A sends to S $\mathsf{Sig}(x_1)$ and c_i, c_i' for $i \in [1, t_1]$.

   (c) A proves in ZK the statement in Equation 5.2 using private inputs $x_1, b_1, \ldots, b_{t_1}, r_1, \ldots, r_{t_1}$. For each $i \in [1, t_1]$, A also proves in ZK the statement in Equation 5.4 using private inputs $b_i, r_i, \ell_i^{b_i}, r_i'$.

2. S computes wire labels $\ell_i^0 = \mathsf{PRF}(k_2, i - t_1)$ and $\ell_i^1 = \ell_i^0 \oplus \Delta$ for $i \in [t_1 + 1, m]$. S then construct garbled gates $\mathcal{G}_f$ and sends $\mathcal{G}_f$ and A's commitments c_i' for $i \in [1, t_1]$ to B.

3. S and B engage in t_2 instances of 1-out-of-2 OT as in Protocol 2 together with verification of B's input. Before B can learn labels $\ell_{t_1+i}^{b_i}$, B forms t_2 commitments $c_i'' = \mathsf{Com}(b_i, r_i'')$ using fresh randomness r_i'' and proves in ZK the statements in Equations 5.2 and 5.3 using private input $x_2, b_1, \ldots, b_{t_2}, r_1'', \ldots, r_{t_2}''$ and b_i, r_i'', k_i, respectively. Here k_i denotes the value chosen during step 2 of the ith instance of the OT protocol.

4. A opens commitments c_i' by sending to B pairs $(\ell_i^{b_i}, r_i')$ for i $\in [1, t_1]$. B checks whether $\mathsf{Com}(\ell_i, r_i') = c_i'$ for each i and aborts if at least one check fails.

5. The remaining steps are the same as in Protocol 2.

they may also compute a function of their individual data, which will be used as the input into the joint computation.

5.6.1 Ancestry Test

This test would often be invoked when A and B already know to be related or have reasons to believe to be related. Under such circumstances, they are unlikely to try to cheat each other. For that reason, we use the solution with semi-honest A and B to realize this test. Because SNP-based tests are most general and can provide information about recent as well as distant ancestry, we build a circuit that takes a large number of SNPs from two individuals and counts the number of positions with the same values. The computed value is then compared to a number of thresholds to determine the closest generation in which the individuals have the same ancestor.

To compute the number of SNPs which are equal in the DNA of two individuals, the circuit first proceeds by XORing two binary input vectors from A and B (recall that the value of each SNP is a bit) and then counts the number of bits that differ in a hierarchical manner. That is, in the first round of additions, every two adjacent bits are added and the result is a 2-bit integer. In the second round of additions, every two adjacent results from the first round are added resulting in 3-bit sums. This process continues in $\lceil \log_2 t \rceil$ rounds of additions, where t is the size of A's and B's input, and the last round performs only a single addition. As mentioned earlier, the result can be interpreted by performing a number of comparisons at the end, but the cost of final comparisons is insignificant compared to the remaining size of the circuit.

5.6.2 Paternity Test

We assess that the security setting with malicious users A and B is the most suitable for running paternity tests. That is, the participants may be inclined to tamper with the computation to influence the result of the computation. It is, however, difficult to learn the other party's genetic information by modifying one's input into the function. In particular, recall from Equation 5.1 that the output of a paternity test is a single bit, which indicates whether the exact match was found. Then if a malicious participant engages in the computation with the same victim multiple times and modifies the input in the attempt to discover the victim's genomic data, the single bit output does not help the attacker to learn how his inputs are to be modified to be closer to the victim's input. The situation is different when the output of the computation reveals information about the distance between the inputs of A and B, but we do not consider such computation in this work. Thus, we do not use input certification for paternity tests.

This test would normally be run between an individual and a contested father of that individual according to the computation in Equation 5.1. We thus implement the computation in Equation 5.1 using a Boolean circuit. For each i, the circuit XORs the vectors $\langle x_{i,1}, x_{i,2}, x_{i,1}, x_{i,2} \rangle$ and $\langle x'_{i,1}, x'_{i,1}, x'_{i,2}, x'_{i,2} \rangle$ and compares each of the four value in the resulting vector to 0. The (in)equality to 0 testing is performed using $k-1$ OR gates, where k is the bitlength of all $x_{i,j}$'s and $x'_{i,j}$'s. Finally, we compute the AND of the results of the 4 equality tests, OR the resulting bits across i's, and output the complement of the computed bit.

5.6.3 Genetic Compatibility Test

When A and B want to perform a compatibility test, we assume that they want to evaluate the possibility of their children inheriting at least one recessive genetic disease. Thus, we assume that A and B agree on a list of genetic diseases to be included in the test (this list can be standard, e.g., suggested by S or a medical association). Because performing a test for a specific genetic disease is only meaningful if both parties wish to be tested for it, we assume that A and B can reconcile the differences in their lists.

To maximize privacy, we construct the function f to be as conservative as possible. In particular, given a list of genetic diseases L, A and B run a compatibility test for each disease $D \in L$, and if at least one test resulted in a positive outcome, the function will output 1, and otherwise it will output 0. That is, the function can be interpreted as producing 1 if A and B's children have a chance of inheriting the major variety for at least one of the tested diseases; and producing 0 means that their children will not inherit the major variety for any of the diseases in L. Evaluating this function can be viewed as the first step in A and B's interaction. If the output was 1, they may jointly decide to run more specific computation to determine the responsible disease or diseases themselves.

The above means that for each $D \in L$, A can locally run the test to determine whether she is a carrier of D. B performs the same test on his data. Thus, A's and B's input into f consists of $|L|$ bits each and the result is 1 iff $\exists i$ such that A's and B's ith input bits are both 1. This computation can be realized as a simple circuit consisting of $|L|$ AND and $|L| - 1$ OR gates.

Next, notice is that it is easy for malicious A or B to learn sensitive information about the other party by using certain inputs. That is, if a malicious user sets all his input bits to 1, he will be able to learn whether the other party is a carrier of least one disease in L. This poses substantial privacy concerns, particularly for matchmaking services that routinely run genetic compatibility tests between many individuals. Thus, we require that A and B certify the results of testing for each genetic disease on the list (e.g., by a medical facility) and enter certified inputs into the computation. (Note that the medical facility that performs sequencing can also certify the test results; alternatively, the medical facility performing test certification will require genome certification from the facility that performed sequencing.) This means that the server-aided solution with certified inputs will be used for secure computation.

For each disease $D \in L$, the signature will need to include the name of the disease D and the test outcome σ, which we assume is a bit. Then if we target efficient the computation, the disease names will not be input into the circuit, but instead S will verify that A's signature used for a particular input wire includes the same disease name as B's signature used for an equivalent input wire. A simple way to achieve this is to reveal list L to S and reveal the name of the disease including in each signature (without revealing the signature itself). If we assume that each issued signature is on the tuple (D, σ), i.e., the signature was produced as $v^e \equiv a_1^D a_2^\sigma b^s c$, all that is needed is to adjust the value used in the ZKPK of the signature by $\frac{1}{a_2^D}$ by both the sender and the verifier for each $D \in L$ (we refer the reader to [37] for details). S will need to check that all conditions appear in the same order among A's and B's inputs (i.e., the sequences of diseases are identical) before proceeding with the rest of the protocol. Revealing the set of diseases used in the compatibility test would not constitute violation of privacy if such a set of conditions is standard or suggested by S itself.

When, however, the parties compose a custom set of genetic diseases for their genetic compatibility testing and would like to keep the set private, they may be unwilling reveal the set of diseases to S. We propose that the parties instead prove that they are providing results for the same conditions without revealing the conditions themselves to the server. The difficulty in doing so arises from the fact that S interacts independently with A and B (possibly at non-overlapping times) and A and B are not proving any joint statements together. Our idea of proving that inputs of A and B correspond to the same sequence of diseases consists of forming a sequence of commitments to the diseases in L, the openings of which are known to both A and B. That is, A and B jointly generate a commitment to each disease using shared randomness and used those commitments at the time of proving that their inputs have been certified. Then if A supplies commitments $\mathsf{Com}_1, \ldots, \mathsf{Com}_t$ and proves that the committed values correspond to the diseases in her signatures, S will check that B supplies the same sequence of commitments and also proves that the committed values are equal to the diseases in the signatures he possesses. This will ensure that A and B supply input bits for the same sequence of diseases. To jointly produce the commitments, we have both A and B contribute their own randomness and the resulting commitment will be a function of A's and B's contribution. It can proceed as follows (recall that A and B can be malicious):

1. A chooses a random r_A and sends to B $c_A = \mathsf{Com}(r_A, z)$.
2. B chooses a random r_B and sends to A
3. A and B open their commitments by exchanging (r_A, z) and (r_B, z') and verify that they match c_A and c_B, resp.
4. They form joint randomness as $r = r_A \oplus r_B$ and use it to construct commitment $\mathsf{Com}(D, r)$.

Then the (high-level) statement that A and B prove about their inputs is $PK\{(\alpha, \sigma) : \mathsf{Sig}(\alpha, \sigma) \wedge y_1 = \mathsf{Com}(\alpha) \wedge y_2 = \mathsf{Com}(\sigma)\}$ using y_1 shared between A and B, while the remaining portion is specific to A and B as detailed in Section 5.5.3.

5.7 Performance Evaluation

In this section, we report on the results of our implementation. The implementation used Miracl library [47] for large number arithmetic and JustGarble library [25] for garbled circuit implementation. We provide experimental results for ancestry, paternity, and compatibility tests implemented as described in Section 5.6. The security parameters for symmetric key cryptography and statistical security were set to 128. The security parameter for public key cryptography (for both RSA modulus and discrete logarithm setting) was set to 1536. Additionally, the security parameter for the group size in the discrete logarithm setting was set to 192. Note that in practice

S is expected to have more powerful hardware and the runtimes can be significantly reduced by utilizing more cores. All experiments were run 5 times, and the mean value is reported.

To provide additional insights into which protocol component is main performance bottleneck, we separately report computation times for different parts of each solution (e.g., garbled circuit evaluation, OT, etc.). Furthermore, we separately list the times for offline and online computation, where, as in other publications, offline computation refers to all operations that can be performed before the inputs become available. Lastly, because the speed of communication channels can greatly vary, we separately report the size of communication for each party and communication time is not included in the runtimes. In several cases overlaying computation with communication is possible (e.g., S can perform OT computation and simultaneously transmit the garbled circuit) and the overall runtime does not need to be the sum of computation and communication time. Next, We discuss ancestry, paternity, and compatibility tests in their respective settings.

5.7.1 Ancestry Test

Recall that the ancestry test is implemented in the setting where A and B are semi-honest, but S can be malicious. We ran this test using 2^{17} SNPs as the input for A and B. The resulting circuit used 655,304 XOR gates and 131,072 non-XOR gates. The computation time and communication size are given in Table 5.1. We used the original JustGarble implementation as well as implement a variant with the recent half-gates optimization [116], which reduces bandwidth associated with transmitting garbled circuit.[3] Both variants are listed in Table 5.1. In the context of

[3]In both the original and half-gates implementations, garbling a non-free gate involves calling AES on 4 blocks, while evaluation of a half gate calls AES on 2 blocks and on 1 block in the original implementation. Any deviations in the run time from these expectations are due to non-cryptographic operations.

this work, the half-gates optimization has the largest impact on the performance of the first protocol, as in the remaining protocols other components of SFE are likely to dominate the overall time.

TABLE 5.1

PERFORMANCE OF ANCESTRY TEST WITHOUT/WITH HALF-GATES[4]

Party	Garbled circuit		Communication	
	garble (offline)	eval (online)	sent	received
A	1.8	–	2	0
B	19.8/18.4	–	8/6	0
S	–	12.5/15.9	0	10/8

The implementation assumes that A only creates labels for her input wires and communicates 2^{17} labels to B. B performs the rest of the garbling work and interacts with S. As expected, the time for circuit garbling and evaluation is small, but the size of communication is fairly large because of the large input size and consecutively circuit size. Nevertheless, we consider the runtimes very small for the computation of this size.

To provide insights into performance gains of our solution compared to the regular garbled circuit computation in the semi-honest setting, we additionally implement the garbled circuit-based approach in the presence of semi-honest A and B only. In

[4]Work is in MS. Communication is in MB.

addition to circuit garbling and evaluation, this also requires the use of OT, which we implement using a recent optimized OT extension construction from [14] (including optimizations specific to Yao's garbled circuit evaluation). As in [14], we use Naor-Pinkas OT for 128 base OTs [99]. The results are given in Table 5.2. Compared to the server-aided setting, computation is higher by at least two orders of magnitude for each party and communication is noticeably increased as well.

TABLE 5.2

PERFORMANCE OF ANCESTRY TEST WITHOUT SERVER WITHOUT/WITH HALF-GATES[5]

Party	Garbled circuit		OT		Total time		Comm	
	garble	eval	offline	online	offline	online	sent	received
A	21.6/20.2	–	195.2	1983	216.8/215.4	1983	10.02/8.02	2.03
B	–	12.5/15.9	2003	218.6	2003	231.1/234.5	2.03	10.02/8.02

5.7.2 Paternity Test

Next, we look at the paternity test, implemented as described in Section 5.6 in the presence of malicious A and B and semi-honest S. The inputs for both A and B consisted of 13 2-element sets, where each element is 9 bits long. We use OT extension from [14] with 128 Naor-Pinkas base OTs. The circuit consisted of 468

[5]Work is in MS. Communication is in MB.

XOR and 467 non-XOR gates. The results of this experiment are reported in Table 5.3. The computation for output verification is reported only as part of total time.

TABLE 5.3

PERFORMANCE OF PATERNITY TEST (NO HALF-GATES)[6]

Party	GC		OT		Total time		Comm	
	garble	eval	offline	online	offline	online	sent	received
A	0.003	–	–	–	0.003	–	3.7	0.06
B	–	0.01	515.5	201.7	515.5	201.7	31.67	56.88
S	0.03	–	196.1	260.9	196.1	260.9	53.32	31.66

Not surprisingly, the cost of OT dominates the overall runtime, but for A the overhead is negligible (the cost of generating input labels and verifying the output labels returned by B). Thus, it is well-suited for settings when one user is very constrained.

Compared to garbled circuit computation in the presence of malicious participants, our solution reduces both computation and communication for the participants by at least two orders of magnitude. This is because practical constructions rely on cut-and-choose (and other) techniques to ensure that the party who garbles the circuit in unable to learn unauthorized information about the other participant's input. Recent results such as [91, 96, 112] require the circuit generator to garble on

[6]Work is in MS. Communication is in KB.

the order of 125 circuits for cheating probability of at most 2^{-40}, some of which are checked (i.e., re-generated) by the circuit evaluator, while the remaining circuits are evaluated. Thus, the work of each of A and B will have to increase by at least two orders of magnitude just for circuit garbling and evaluation, not counting other techniques that deter a number of known attacks and result in increasing the input size and introducing expensive public key operations. A notable exception to the above is the work of Lindell [89] that reduces the number of circuits to 40 for the same cheating probability. The construction, however, results in savings only for circuits of large size as it introduces a large number of additional public key operations. Thus, for paternity tests using constructions with a larger number of circuits is very likely to be faster in practice, which results in a drastic difference between our solution and regular garbled circuit protocol with malicious participants. This difference in performance can be explained by the fact that in our setting one party is known not to deviate from the protocol allowing for a more efficient solution.

Baldi et al. [20] also provide a private paternity test in the two-party setting (between a client and a server). It uses a different computation based on Restriction Fragment Length Polymorphisms (RFLPs) and relies on private set intersection as a cryptographic building block. Both offline and online times for the client and the server are 3.4 ms and the communication size is 3KB for the client and 3.5KB for the server when the test is performed with 25 markers. All times and communication sizes double when the test is run with 50 markers. While the runtimes we report are higher, the implementation of [20] did not consider malicious participants. If protection against malicious A and B in our solution is removed, the work for all parties reduces to well below 0.1 millisecond and communication becomes a couple of KBs.

5.7.3 Genetic Compatibility Test

The last genetic compatibility test is run in the setting where A and B are malicious and their inputs must be certified. We choose the variant of the solution that reveals the list of diseases L to the server (i.e., a standard list is used). We implement the signature scheme, OT, and ZKPKs as described earlier. All ZKPKs are non-interactive using the Fiat-Shamir heuristic [59]. We used $|L| = 10$ and thus A and B provide 10 input bits into the circuit accompanied by 10 signatures. The circuit consisted of only 19 non-XOR gates. The performance of the test is given in Table 5.4. We divide all ZKPKs into a proof of signature possession (together with a commitment), denoted by "Sign PK" in the table, and the remaining ZK proofs, denoted by "Other PK." As it is clear from the table, input certification contributes most of the solution's overhead, but it is still on the order of 1–3 seconds for all parties.

TABLE 5.4

PERFORMANCE OF COMPATIBILITY TEST (NO HALF-GATES)[7]

Party	Garbled circuit		OT		Sign PK		Other PK		Total time		Comm	
	garble	eval	offline	online	offline	online	offline	online	offline	online	sent	received
A	0	–	–	–	1170	42.1	616.8	20.6	1790	62.7	34.35	0.06
B	–	0.001	15.4	14.6	1170	42.1	282.4	15.7	1470	72.4	36.41	2.98
S	0.003	–	29.3	15.2	0	2060	0	756	29.3	2830	2.87	70.59

[7]Work is in MS. Communication is in KB.

As mentioned earlier, we are not aware of general results that achieve input certification for comparison. However, the comparison to general garbled circuit computation in the presence of malicious parties or server-aided garbled circuit computation from Section 5.7.2 applies here as well.

Baldi et al. [20] also build a solution and report on the performance of genetic compatibility test. In [20], testing for presence of a genetic disease that client carries in the server genome consists of the client providing the disease fingerprint in the form of $(nucleotide, location)$ pairs (which is equivalent to a SNP) and both parties searching whether the disease fingerprint also appears in the server's DNA. This requires scanning over the entire genome, which our solution avoids. As a result, the solution of [20] incurs substantial offline overhead for the server (67 minutes) and large communication size (around 4GB) even for semi-honest participants. The solution utilizes authorized private set intersection, which allows inputs of one party (as opposed to both in our work) to be verified. Compared to [20], in our framework, testing for a single disease requires a fraction of a second for each party with malicious A and B, where inputs of both of them are certified. The computation is greatly simplified because the list of diseases is assumed to be known by both users. When this is the case, the cost of input certification greatly dominates the overall time.

CHAPTER 6

FINGERPRINT RECOGNITIONS

Fingerprint is one of the most accurate type of biometric data that used for versification and identification; therefore, protecting the data is crucial during the fingerprint recognition computations. In this chapter, we present three secure privacy-preserving protocols for fingerprint alignment and matching, based on what are considered to be the most precise and efficient fingerprint recognition algorithms—those based on the *geometric* matching of "landmarks" known as *minutia* points. Our protocols allow two or more honest-but-curious parties to compare their respective privately-held fingerprints in a secure way such that they each learn nothing more than a highly-accurate score of how well the fingerprints match. To the best of our knowledge, this is the first time fingerprint alignment based on minutiae is considered in a secure computation framework.

In this chapter, we briefly describe motivation and contributions in Sections 6.1 and 6.2. Section 6.3 provides necessary fingerprint background, including the three fingerprint comparison algorithms which are the focus of this work. Section 6.4 describes the problem statement which we rely to realize secure fingerprint comparisons. Then Section 6.5 describes new secure sub-protocols not available in the prior literature that we design to enable secure fingerprint comparisons. Our main protocols for the three selected fingerprint alignment and matching algorithms are given in Section 6.6. Lastly, experimental evaluation of our secure realization of the third spectral minutia representation algorithm in two different secure computation frameworks is given in Section 6.7.

6.1 Motivation

Computing securely with biometric data is challenging because biometric identification applications often require accurate metrics and recognition algorithms, but the data involved is so sensitive that if it is ever revealed or stolen the victim may be vulnerable to impersonation attacks for the rest of their life. This risk is particularly true for fingerprints, which have been used since the 19th century for identification purposes and are being used for such purposes ubiquitously today, including for cellphones, laptops, digital storage devices, safes, immigration, and physical building/room access, as well as the classic application of identifying criminals. Thus, there is a strong motivation for fast and secure fingerprint recognition protocols that protect fingerprint data but nevertheless allow for highly accurate scores for fingerprint comparison.

6.2 Contributions

The setting we consider in this work is one where the parties involved are honest-but-curious (although, in some cases, malicious parties can also be tolerated using known hardening techniques). That is, two or more parties hold private fingerprint data that they would like to compare in a fast way that provides an accurate score for the comparison but does not reveal the actual fingerprint data. For example, the computation could involve comparing multiple pairs of fingerprint representations stored in two privacy-sensitive databases (e.g., one owned by the FBI and the other owned by a university), or it could involve a single comparison of a suspected criminal's fingerprint to a fingerprint found at a crime scene.

According to accepted best practices (e.g., see [92, 98, 115]) the most accurate *fingerprint recognition* algorithms are based on the use of *minutia* points, which are fingerprint "landmarks," such as where two ridges merge or split. (See Figure 6.1.)

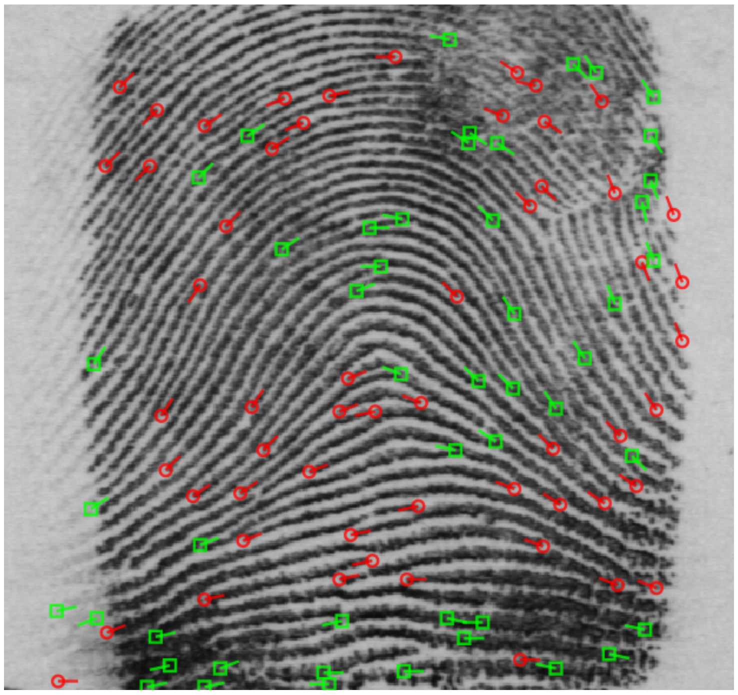


Figure 6.1. A set of minutiae with orientations. The image is generated by NIST's Fingeprint Minutiae Viewer (FpMV) software [7] using a fingerprint from NIST's Special Database 4 [5].

Such *geometric* recognition algorithms generally involve two main steps: *alignment* and *matching*. Alignment involves computing a geometric transformation to best "line up" two sets of minutiae and matching involves scoring the similarity of two sets of (hopefully overlapping) minutia points. Alignment is necessary for the meaningful and accurate application of the matching step, yet, to the best of our knowledge, all previous publications on secure fingerprint recognition, except for a paper by Kerschbaum *et al.* [82], focus exclusively on the matching step, possibly because of the computational difficulty of finding a good alignment. Instead, our approach is on the design of efficient protocols for the entire fingerprint recognition process, including both the alignment and matching steps, so as to provide secure protocols for accurate fingerprint recognition algorithms used in practice.

In this work, we focus on three algorithms that compare fingerprints using both alignment and matching:

1. A simple geometric transformation algorithm that searches for the maximum matching by considering each pair of minutia [92].
2. An algorithm aligns fingerprints based on high curvature points [98].
3. An algorithm based on a spectral minutia representation [115].

These algorithms were selected due to their precision, speed, and/or popularity, with the goal of building efficient and practical security solutions. Our main contributions can be summarized as follows:

- We design new secure sine, cosine, and arctangent sub-protocols for fixed-point arithmetic, for both two-party (garbled circuit) and multi-party (secret sharing) cases.
- We design a new secure square-root sub-protocol for fixed-point arithmetic. Our solution is for the two-party (garbled circuit) setting and is based on Goldschmidt's method with the last iteration replaced by Newton-Raphson's method to eliminate accumulated errors.
- We design a new secure sub-protocol for selecting the fth smallest element in a set of comparable elements. Our method is based on an efficient square-root sampling algorithm.
- We build three secure and efficient protocols for fingerprint alignment and matching, based on the use of the above new building blocks applied to the three well-known fingerprint recognition algorithms mentioned above. The constructions work in both two-party (garbled circuit) and multi-party (secret sharing) settings.
- We implement one of the secure fingerprint recognition protocols and show its efficiency in practice.

Our constructions are presented in the semi-honest model. However, a number of available techniques can be used to achieve security in the malicious model (e.g., [13, 54, 64] and others in the multi-party (secret sharing) setting and [87] among others in the two-party (garbled circuit) setting).

6.3 Fingerprint Background

A fingerprint represents an exterior surface of a finger. All features extracted from a fingerprint image are intended to allow one to uniquely identify the individual who owns the fingerprint. One of the features most commonly used for fingerprint recognition is *minutiae*, which are represented as a set of points in the two-dimensional plane (possibly with an added angle orientation). Minutiae points generally correspond to fingerprint ridge endings or branches, and are typically represented by the following elements [92]: 1) an x-coordinate, 2) a y-coordinate, 3) an optional orientation, θ, measured in degrees, and 4) an optional minutia type. Some algorithms might store additional information in the fingerprint representation, e.g., the relationship of each minutia point to a reference point such as the core point representing the center of a fingerprint.

All of the algorithms that we describe are based on minutia points. One of them, however, uses an alternative representation in the form of a spectrum of a fixed size. Thus, the minutiae might be not represented as points, but instead use a spectral representation as detailed later in this section.

Furthermore, one of the fingerprint alignment and matching algorithms upon which we build—namely, the approach that uses curvature information for alignment—relies on an additional type of fingerprint features. In particular, it makes use of high curvature points extracted from a fingerprint image. In addition to storing a number of minutia points, each fingerprint contains a set of high curvature points, each of which is represented as (x, y, w). Here, first two elements indicate the location of the point and the last element is its curvature value in the range 0–2 (see [98] for the details of its computation).

In the remainder of this section, we describe the three selected fingerprint recognition algorithms without security considerations. All of them take two fingerprints T and S as their input, align the fingerprints, and output the number of matching minutiae between the aligned fingerprint representations. In all algorithms, a fingerprint representation contains a set of minutiae, $(t_1, \ldots, t_m)$ for T and $(s_1, \ldots, s_n)$, for S, or an alternative representation derived from the minutiae. Additionally, the second algorithm takes a number of high curvature points stored in a fingerprint as well, and we denote them by $(\hat{t}_1, \ldots, \hat{t}_{\hat{m}})$ for T and $(\hat{s}_1, \ldots, \hat{s}_{\hat{n}})$ for S.

6.3.1 Fingerprint Recognition Using Brute Force Geometrical Transformation

The first algorithm searches for the best alignment between T and S by considering that minutia, $t_i \in T$, corresponds to minutia, $s_j \in S$, for all possible pairs (t_i, s_j) (called a reference pair) [92]. Once a new reference pair is chosen, the algorithm transforms the minutiae from S using a geometrical transformation and also rotates them, after which it counts the number of matched minutiae. After trying all possible reference pairs, the algorithm chooses an alignment that maximizes the number of matched minutiae and thus increases the likelihood of the two fingerprints to be matched. Algorithm 2 lists the details of this approach.

To implement the matching step, we use an algorithm that iterates through all points in $t_i \in T$ and pairs t_i with a minutia $s_j \in S$ (if any) within a close spatial and directional distance from t_i. When multiple points satisfy the matching criteria, the closest to t_i is chosen, as described in [34, 92]. This computation of the matching step is given in Algorithm 3.

The time complexity of Algorithm 3 is $O(nm)$, and the time complexity of Algorithm 2 is $O(n^2m^2)$ when Algorithm 3 is used as its subroutine.

6.3.2 Fingerprint Recognition Using High Curvature Points for Alignment

The second fingerprint recognition algorithm [98] uses high curvature information extracted from the fingerprints to align them. The alignment is based on the iterative closest point (ICP) algorithm [27] that estimates the rigid transformation F between

```
Algorithm 2: Fingerprint Recognition based on Geometrical Transformation
  Input: Two fingerprints $T = \{t_i = (x_i, y_i, \theta_i)\}_{i=1}^m$ and $S = \{s_i = (x'_i, y'_i, \theta'_i)\}_{i=1}^n$.
  Output: The largest number of matching minutiae, $C_{max}$, and the
   corresponding alignment.
    1. Initialize $C_{max} = 0$.
    2. For $i = 1, \ldots, m$ and $j = 1, \ldots, n$, use $(t_i, s_j)$ as a reference pair and perform
       (a) Compute transformation $\Delta x = x'_j - x_i$, $\Delta y = y'_j - y_i$, and $\Delta\theta = \theta'_j - \theta_i$.
       (b) Transfer each minutia $s_k \in S$ as $x''_k = \cos(\Delta\theta) \cdot x'_k + \sin(\Delta\theta) \cdot y'_k - \Delta x$,
           $y''_k = -\sin(\Delta\theta) \cdot x'_k + \cos(\Delta\theta) \cdot y'_k - \Delta y$, and $\theta''_k = \theta'_k - \Delta\theta$ and save the
           result as $S'' = \{s''_i = (x''_i, y''_i, \theta''_i)\}_{i=1}^n$.
       (c) Compute the number $C$ of matched minutiae between $T$ and $S''$ (e.g.,
           using Algorithm 3).
       (d) If $C_{max} < C$, then set $C_{max} = C$ and
           $(\Delta x_{max}, \Delta y_{max}, \Delta\theta_{max}) = (\Delta x, \Delta y, \Delta\theta)$.
    3. Return $C_{max}$ and its corresponding alignment $(\Delta x_{max}, \Delta y_{max}, \Delta\theta_{max})$.
```

T and S. Once this transformation is determined, it is applied to the minutia points of S. Then the algorithm computes the number of matched minutiae between T and transformed S. The ICP algorithm assumes that the fingerprints are roughly pre-aligned, which can be done, e.g., by aligning each fingerprint independently using its core point, and iteratively finds point correspondences and the transformation between them. To eliminate alignment errors when the overlap between the two sets is partial, [98] suggests using the trimmed ICP (TICP) algorithm [48]. The TICP algorithm ignores a proportion of the points in T whose distances to the corresponding points in S are large, which makes it robust to outliers in the data.

The details of this fingerprint recognition approach are given in Algorithm 4. Steps 1–10 correspond to the TICP algorithm that proceeds with iterations, aligning the fingerprints closer to each in each successive iteration. The algorithm termination parameters λ and γ correspond to the threshold for the total distance between the matched points of T and S and the limit on the number of iterations, respectively.

Algorithm 3: Fingerprint matching

```
Input: Two fingerprints $T = \{t_i = (x_i, y_i, \theta_i)\}_{i=1}^m$, $S = \{s_i = (x'_i, y'_i, \theta'_i)\}_{i=1}^n$ and
  thresholds $\lambda$ and $\lambda_\theta$ for distance and orientation.
Output: The number $C$ of matched minutia pairs between $T$ and $S$.
  1. Set $C = 0$.
  2. Mark each $s_j \in S$ as available.
  3. For $i = 1, \ldots, m$, do
     (a) Create a list $L_i$ consisting of all available points $s_j \in S$ that satisfy
         $\min(|\theta_i - \theta'_j|, 360 - |\theta_i - \theta'_j|) < \lambda_\theta$ and $\sqrt{(x_i - x'_j)^2 + (y_i - y'_j)^2} < \lambda$.
     (b) If $L_i$ is not empty, select the closest minutia $s_k$ to $t_i$ from $L_i$, mark $s_k$ as
         unavailable and set $C = C + 1$.
  4. return $C$.
```

For robustness, the alignment is performed using only a subset of the points (f closest pairs computed in step 4). The optimal motion for transforming one set of the points into the other (treated as two 3-D shapes) is computed using the union quaternion method due to Horn [73], and is given in Algorithm 5. The transformation is represented as a 3×3 matrix R and a vector v of size 3, which are consequently used to align the points in S (step 8).

Once the fingerprints are sufficiently aligned using the high curvature points, the overall transformation between the original and aligned S is computed (step 11). The resulting transformation is applied to the minutia points of S. The new x and y coordinates of the minutiae can be computed directly using the transformation (R, v), while orientation θ requires a special handling (since it is not used in computing the optimal motion). We compute the difference in the orientation between the original and aligned S as the angle between the slopes of two lines drawn using two points from the original and transformed S, respectively (step 12).

The original ICP algorithm [27] lists expected and worst case complexities of computing the closest points (step 3 of Algorithm 4) to be $O(\hat{m} \log \hat{n})$ and $O(\hat{m}\hat{n})$,

Algorithm 4: Fingerprint recognition based on high curvature points for alignment

```
Input: Two fingerprints consisting of minutiae and high curvature points
  $T = (\{t_i = (x_i, y_i, \theta_i)\}_{i=1}^m, \{\hat{t}_i = (\hat{x}_i, \hat{y}_i, \hat{w}_i)\}_{i=1}^{\hat{m}})$ and $S = (\{s_i = (x'_i, y'_i, \theta'_i)\}_{i=1}^n,$
  $\{\hat{s}_i = (\hat{x}'_i, \hat{y}'_i, \hat{w}'_i)\}_{i=1}^{\hat{n}})$; a minimum number of matched high curvature points
  $f$; and algorithm termination parameters $\lambda$ and $\gamma$.
Output: The largest number of matching minutiae, $C$, and the corresponding
  alignment.
  1. Set $S_{LTS} = 0$.
  2. Store a copy of $\hat{s}_i$'s as $\bar{s}_i = (\bar{x}_i, \bar{y}_i, \bar{w}_i)$ for $i = 1, \ldots, \hat{n}$.
  3. For $i = 1, \ldots, \hat{n}$, find the closest to $\hat{s}_i$ point $\hat{t}_j$ in $T$ and store their distance $d_i$.
     Here, the distance between any $\hat{s}_i$ and $\hat{t}_j$ is defined as
     $\sqrt{(\hat{x}'_i - \hat{x}_j)^2 + (\hat{y}'_i - \hat{y}_j)^2} + \beta|\hat{w}'_i - \hat{w}_j|$.
  4. Find $f$ smallest $d_i$'s and calculate their sum $S'_{LTS}$.
  5. If $S'_{LTS} \leq \lambda$ or $\gamma = 0$, proceed with step 11.
  6. Set $S_{LTS} = S'_{LTS}$.
  7. Compute the optimal motion $(R, v)$ for the selected $f$ pairs using Algorithm 5.
  8. For $i = 1, \ldots, \hat{n}$, transform point $\hat{s}_i$ according to $(R, v)$ as $\hat{s}_i = R\hat{s}_i + v$.
  9. Set $\gamma = \gamma - 1$.
  10. Repeat from step 3.
  11. Compute the optimal motion $(R, v)$ for $\hat{n}$ pairs $(\bar{s}_i, \hat{s}_i)$ using Algorithm 5. Let
      $r_{i,j}$ denote $R$'s cell at row $i$ and column $j$ and $v_i$ denote the $i$th element of $v$.
  12. Compute $c_1 = (\bar{y}_2 - \bar{y}_1)/(\bar{x}_2 - \bar{x}_1)$, $c_2 = (\hat{y}'_2 - \hat{y}'_1)(\hat{x}'_2 - \hat{x}'_1)$, and
      $\Delta\theta = \arctan((c_1 - c_2)/(1 + c_1 c_2))$.
  13. For $i = 1, \ldots, n$, apply the transformation to minutia $s_i$ by computing
      $x'_i = r_{1,1}x'_i + r_{1,2}y'_i + v_1$,
      $y'_i = r_{2,1}x'_i + r_{2,2}y'_i + v_2$,
      and $\theta'_i = \theta'_i - \Delta\theta$.
  14. Compute the number, $C$, of matched minutiae between $t_i$'s and transformed
      $s_i$'s (e.g., using Algorithm 3).
  15. Return $C$ and the transformation $(R, v)$.
```

```
Algorithm 5: Union quaternion method for computing optimal motion
  Input: $n$ pairs $\{(t_i = (x_i, y_i, z_i), s_i = (x'_i, y'_i, z'_i))\}_{i=1}^{n}$.
  Output: Optimal motion $(R, v)$.
    1. For $i = 1, \ldots, n$, compute unit quaternion
       $q_i = (q_{(i,1)}, q_{(i,2)}, q_{(i,3)}, q_{(i,4)}) = \left(\sqrt{\frac{1+k_i}{2}}, u_i\sqrt{\frac{1-k_i}{2}}\right)$, where
       $k_i = \frac{x_i \cdot x'_i + y_i \cdot y'_i + z_i \cdot z'_i}{\sqrt{x_i^2+y_i^2+z_i^2} \cdot \sqrt{x_i'^2+y_i'^2+z_i'^2}}$, $u_i = \left(\frac{y_i \cdot z'_i - z_i \cdot y'_i}{||t_i \times s_i||}, \frac{z_i x'_i - x_i z'_i}{||t_i \times s_i||}, \frac{x_i \cdot y'_i - y_i \cdot x'_i}{||t_i \times s_i||}\right)$, and
       $||t_i \times s_i|| = \sqrt{(y_i \cdot z'_i - z_i \cdot y'_i)^2 + (z_i \cdot x'_i - x_i \cdot z'_i)^2 + (x_i \cdot y'_i - y_i \cdot x'_i)^2}$.
    2. Compute the overall unit quaternions $q = [q_1, q_2, q_3, q_4] = q_1 q_2 \ldots q_{n-1} q_n$ by
       executing multiplication from left to right, where
       $q_j q_{j+1} = [q_{(j,1)} q_{(j+1,1)} - v_j \cdot v_{j+1}, q_{(j,1)} v_{j+1} + q_{(j+1,1)} v_j + v_j \times v_{j+1}]$,
       $v_j = (q_{(j,2)}, q_{(j,3)}, q_{(j,4)})$, and $v_{j+1} = (q_{(j+1,2)}, q_{(j+1,3)}, q_{(j+1,4)})$ for
       $j = 1, \ldots, n-1$.
    3. Compute rotation matrix
       $R = \begin{bmatrix} q_1^2 + q_2^2 - q_3^2 - q_4^2 & 2(q_2 q_3 - q_1 q_4) & 2(q_2 q_4 + q_1 q_3) \\ 2(q_3 q_2 + q_1 q_4) & q_1^2 - q_2^2 + q_3^2 - q_4^2 & 2(q_3 q_4 - q_1 q_2) \\ 2(q_4 q_2 - q_1 q_3) & 2(q_4 q_3 - q_1 q_2) & q_1^2 - q_2^2 - q_3^2 + q_4^2 \end{bmatrix}$.
    4. Compute transformation vector $v = t - Rs$, where $t = (\sum_{i=1}^{n} t_i)/n$ and
       $s = (\sum_{i=1}^{n} s_i)/n$.
    5. Return $(R, v)$.
```

respectively. The approach, however, becomes prone to errors in the presence of outliers, and thus in the TICP solution [48] a bounding box is additionally used to eliminate errors. In step 4, we can use fth smallest element selection to find the smallest distances, which runs in linear time (the TICP paper suggests sorting, but sorting results in higher asymptotic complexities). The complexity of Algorithm 5 is $O(k)$ when run on k input pairs. Thus, the overall complexity of Algorithm 4 (including Algorithms 5 and 3) is $O(\gamma \hat{m} \hat{n} + mn)$, where γ is the upper bound on the number of iterations in the algorithm.

6.3.3 Fingerprint Recognition based on Spectral Minutiae Representation

The last fingerprint recognition algorithm by Xu et al. [115] uses minutia-based representation of fingerprints, but offers greater efficiency than other algorithms because of an alternative form of minutia representation. The spectral minutia representation [114] that it uses is a fixed-length feature vector, in which rotation and scaling can be easily compensated for, resulting in efficient alignment of two fingerprints.

In the original spectral minutia representation [114], a set of minutiae corresponds to a real-valued vector of a fixed length D, which is written in the form of a matrix of dimensions $M \times N$ $(= D)$. The vector is normalized to have zero mean and unit energy. Thus, we now represent T and S as matrices of dimensions $M \times N$ and denote their individual elements as $t_{i,j}$'s and $s_{i,j}$'s, respectively. In [114], there are two types of minutia spectra (location-based and orientation-based), each with $M = 128$ and $N = 256$, and a fingerprint represented by their combination consists of 65,536 real numbers.

To compare two fingerprints in the spectral representation, two-dimensional correlation is used as a measure of their similarity. Furthermore, to compensate for fingerprint image rotations, which in this representation become circular shifts in the horizontal direction, the algorithm tries a number of rotations in both directions and computes the highest score among all shifts. This alignment and matching algorithm is presented as Algorithm 6. It is optimized to perform shifts from -15 to 15 units (which correspond to rotations from -10° to $+10^\circ$) and computes only 9 similarity scores instead of all 31 of them. Later in this section, we show how the algorithm can be generalized to work with any amount of shift λ resulting in only $O(\log \lambda)$ score computations.

Xu et al. [115] apply feature reduction to the spectral minutia representation to reduce the size of the feature vector and consequently improve the time of fingerprint comparisons without losing precision. That work describes two types of feature re-

```
Algorithm 6: Fingerprint recognition based on spectral minutia representation
Input: Two real-valued matrices $T = \{t_{i,j}\}_{i=1,j=1}^{M,N}$ and $S = \{s_{i,j}\}_{i=1,j=1}^{M,N}$ and
  parameter $\lambda = 15$ indicating the maximum amount of rotation.
Output: The best matching score $C_{max}$ and the corresponding alignment.
  1. Set $C_{max} = 0$.
  2. For $\alpha = -12, -6, 0, 6, 12$, do
     (a) Compute the similarity score between $T$ and $S$ horizontally shifted by $\alpha$
         positions as $C_\alpha = \frac{1}{MN} \sum_{i=1}^{M} \sum_{j=1}^{N} t_{i,j} \cdot s_{i,(j+\alpha) \bmod N}$.
     (b) If $C_\alpha > C_{max}$, set $C_{max} = C$, $k = \alpha$, and $\alpha_{max} = \alpha$.
  3. For $\alpha = k-2, k+2$, do
     (a) Compute the similarity score between $T$ and $S$ shifted by $\alpha$ positions as
         in step 2(a).
     (b) If $C > C_{max}$, set $C_{max} = C$, $k' = \alpha$, and $\alpha_{max} = \alpha$.
  4. For $\alpha = k'-1, k'+1$, do
     (a) Compute the similarity score between $T$ and $S$ shifted by $\alpha$ positions as
         in step 2(a).
     (b) If $C > C_{max}$, set $C_{max} = C$ and $\alpha_{max} = \alpha$.
  5. Return $C_{max}$ and $\alpha_{max}$.
```

duction: Column Principle Component Analysis (CPCA) and Line Discrete Fourier Transform (LDFT) feature reductions. Then one or both of them can be applied to the feature vector, with the latter option providing the greatest savings.

The first form of feature reduction, CPCA, reduces the minutia spectrum feature in the vertical direction. After the transformation, the signal is concentrated in the top rows and the remaining rows that carry little or no energy are removed. As a result, a minutia spectrum is represented as a matrix of dimension $M' \times N$ for some $M' < M$. Because the rotation operation commutes with this transformation, the score computation in Algorithm 6 remain unchanged after CPCA feature reduction.

The second form of feature reduction, LDFT, reduces the minutia spectrum in the horizontal direction and returns a feature matrix of dimension $M \times N'$ (or $M' \times N'$ if applied after the CPCA feature reduction) for some $N' < N$. The matrix consists of complex numbers as a result of applying Fourier transform. After the LDFT, the energy is concentrated in the middle columns and other columns are consequently removed. Shifting matrix S in the resulting representation by α positions now translates into setting its cell at location k, j to $e^{-i\frac{2\pi}{N}j\alpha} \cdot s_{k,j}$, where i indicates the imaginary part of a complex number. Note that circular shifts are no longer applied to matrix rows (including when the LDFT is applied after the CPCA feature reduction).

The resulting matrices of complex numbers T and S are then converted to real-valued matrices and processed using Algorithm 6. If we express a cell of T or S at location k, j as $a_{k,j} + ib_{k,j}$ by separating its real and imaginary parts, then the kth row of T and S is now expressed as a real-valued vector

$$\sqrt{\frac{1}{N}}a_{k,1}, \sqrt{\frac{2}{N}}a_{k,2}, \ldots, \sqrt{\frac{2}{N}}a_{k,N}, \sqrt{\frac{2}{N}}b_{k,2}, \ldots, \sqrt{\frac{2}{N}}b_{k,N'}.$$

Computing the score between T and S then corresponds to computing the dot product of each kth row of T and S and adding the values across all k. When we use Algorithm 6 to compute the best similarity score, we need to adjust the computation in step 2(a) for the new matrix dimensions and also implement rotation as a multiplication instead of a shift. If we expand the quantity $e^{-i\frac{2\pi}{N}j\alpha}$ using the formula $e^{i\varphi} = \cos(\varphi) + i\sin(\varphi)$, the real part of each score between T and S rotated by α positions now becomes:

$$C_\alpha = \frac{1}{MN^2}\sum_{k=1}^{M'}\left(a_{k,1}\left(a'_{k,1}\cos\left(-\frac{2\pi\alpha}{N}\right) - b'_{k,1}\sin\left(-\frac{2\pi\alpha}{N}\right)\right) + \right. \tag{6.1}$$
$$\left. + 2\sum_{j=2}^{N'}\left(\cos\left(-\frac{2\pi j\alpha}{N}\right)\left(a_{k,j}a'_{k,j} + b_{k,j}b'_{k,j}\right) + \sin\left(-\frac{2\pi j\alpha}{N}\right)\left(a'_{k,j}b_{k,j} - a_{k,j}b'_{k,j}\right)\right)\right)$$

where $t_{k,j} = a_{k,j} + ib_{k,j}$ and original $s_{k,j} = a'_{k,j} + ib'_{k,j}$.

Returning to Algorithm 6, we generalize the algorithm to work for any value of λ so that only $O(\log \lambda)$ score computations are necessary. Let $\lambda = c \cdot 3^k$ for some constants $c \geq 2$ and $k \geq 1$. Our algorithm proceeds in $k+1$ iterations computing c scores in the first iteration and 2 scores in each consecutive iteration, which results in the total of $c+2k$ score computations. In the initial iteration 0, we compute c scores at indices $\lambda = \lceil 3^k/2 \rceil, \ldots, \lceil 3^k(2c-1)/2 \rceil$, then determine their maximum C_{max} and the index of the maximum score α_{max}. In iteration $i = 1, \ldots, k$, the algorithm computes two scores at indices $\alpha_{max} \pm 3^{k-i}$, determines the maximum of the computed scores and C_{max}, and updates the value of α_{max} as needed. Compared to Algorithm 6, this approach covers 54 shifts using 8 score computations instead of 9 score computations for 31 shifts. The best performance is achieved when $\lambda = 2 \cdot 3^k$ for some k. If λ is not of that form, we could use a different value of c, cover a wider range of shifts than required, or decrease the step size of the algorithm slower than by a factor of 3 (as was done in Algorithm 6), which results in redundant coverage.

In, [115], parameters M' and N' are chosen to retain most of the signal's energy after the transformations (e.g., 90%) and may be dependent on the data set. The complexity of Algorithm 6 for two fingerprints after both types of feature reduction is $O(M'N' \log \lambda)$.

6.4 Problem Statement

Because of the variety of settings in which fingerprint recognition may be used, we distinguish between different computation setups and the corresponding security settings.

1. There will be circumstances when two entities would like to compare fingerprints that they respectively possess without revealing any information about their data to each other. This can correspond to the cases when both entities own a fingerprint database and would like to find entries common to both of them or when one entity would like to search a database belonging to a different entity for a specific fingerprint. In these settings, the computation takes the form of

secure two-party computation, where the participants can be regarded as semi-honest or fully malicious depending on the application and their trust level.

2. There will also be circumstances when one or more data owners are computationally limited and would like to securely offload their work to more powerful servers or cloud providers (this applies even if all fingerprints used in the computation belong to a single entity). Then multi-party settings that allow for input providers to be different from the computational parties apply. In such settings, the computational parties are typically treated as semi-honest, but stronger security models can also be used for added security guarantees.

The focus of this work is on the semi-honest adversarial model, although standard techniques for strengthening security in the presence of fully malicious participants can be used as well.

In addition, to realize our constructions in a variety of settings, two secure computation frameworks are of interest to us. In the two-party setting, we build on garbled circuit evaluation techniques, while in the multi-party setting, we employ linear secret sharing techniques.

6.5 Secure Building Blocks

To be able to build secure protocols for different fingerprint recognition algorithms, we will need to rely on secure algorithms for performing a number of arithmetic operations. Most of the computation described in Section 6.3 is performed on real numbers, while for some of its components integer arithmetic will suffice (e.g., Algorithm 3 can be executed on integers when its inputs are integer values). Complex numbers are represented as a pair (a real part and an imaginary part) of an appropriate data type.

For the purposes of this work, we choose to use fixed-point representation for real numbers. Operations on fixed-point numbers are generally faster than operations using floating-point representation (see, e.g., [11]), and fixed-point representation provides sufficient precision for this application.

In Section 3.2.1, we listed building blocks from prior literature that we utilize in our solutions. Now, we proceed with presenting our design for a number of new building blocks for which we did not find secure realizations in the literature (Section 6.5.1).

6.5.1 New Building Blocks

To build secure fingerprint recognition algorithms, we also need a number of secure building blocks for rather complex operations that previously have not been sufficiently treated in the literature. Thus, in this section, we present secure constructions for a number of operations, including trigonometric operations, square root, and selection of the fth smallest element of a set.

We explore the available algorithms for computing these functions (e.g., Chebyshev polynomials for trigonometric functions) and build protocols that optimize performance in each of the chosen settings of garbled circuit and secret sharing computation. Our optimizations for designing the building blocks as well as the overall protocols focus on the specific costs of the underlying techniques for secure arithmetic and ensure that we achieve efficiency together with precision and provable security guarantees.

6.5.1.1 Sine, Cosine, and Arctangent

We now give new secure building blocks for three trigonometric functions, which are sine, cosine, and arctangent. We denote secure protocols as $[b] \leftarrow \mathsf{Sin}([a])$, $[b] \leftarrow \mathsf{Cos}([a])$, and $[b] \leftarrow \mathsf{Arctan}([a])$ for each respective function. The output b is a fixed-point value, while the input a (for sine and cosine) represented in degrees can be either integer or fixed-point. For generality, we will assume that a used with trigonometric functions is also a fixed-point value, while slight optimizations are possible when a is known to be an integer (as in the fingerprint algorithms considered in this work).

There are a variety of different approximation algorithms that can be used for trigonometric functions, many of which take the form of polynomial evaluation. Upon examining the options, we chose to proceed with the polynomials described in [68, Chapter 6], as they achieve good precision using only small degree polynomials. Note, that Taylor series of trigonometric functions achieve an approximation when the input is very close to a fixed point. However, in secure computation, we can only assume the input value belongs to a known range (e.g., $[0, 2\pi]$). Therefore, polynomial approximation over a range for inputs from [68] was chosen. The polynomials used in [68] for trigonometric functions take the form $P(x^2)$ or $xP(x^2)$ for some polynomial P over variable x (i.e., use only odd or even powers), which requires one to compute only half as many multiplications as in a polynomial with all powers present.

The approach used in [68] offers two types of polynomial approximations. The first type uses a regular polynomial $P(x)$ of degree N to approximate the desired function. The second type uses a rational function of the form $P(x)/Q(x)$, where P and Q are polynomials of degree N and M, respectively. For the same desired precision, the second option will yield lower degrees N and M and thus fewer multiplications to approximate the function. This option, however, is undesirable when the division operation is much costlier than the multiplication operation. In our setting, the rational form is preferred in the case of garbled circuit evaluation (at least for higher precisions) where multiplication and division have similar costs. However, with secret sharing, the cost of division is much higher than that of multiplication, and we use the first option with regular polynomial evaluation.

It is assumed in [68] that the initial range reduction to $[0, \pi/2]$ is used for the input to trigonometric functions. This means that if input a to sine or cosine is given in degrees, it needs to be reduced to the range $[0, 90]$ and normalized to the algorithm's expectations. Note that it is straightforward to extend the result of the computation to cover the entire range $[0, 2\pi]$ given the output of this function and the original input.

Furthermore, because evaluating trigonometric functions on a smaller range of inputs offers higher precision, it is possible to apply further range reduction and evaluate the function on even a smaller range of inputs, after which the range is expanded using an appropriate formula or segmented evaluation. We refer the reader to [68] for additional detail. For that reason, the tables of polynomial coefficients provided in [68] are for $\sin(\alpha\pi x)$ and $\cos(\beta\pi x)$, where $0 \leq x \leq 1$ and α and β are fixed constants equal to 1/2 or less. To see the benefit provided by smaller α or β, for example, consider the sine function. Then by using $\alpha = 1/2$ and approximating the function using a regular polynomial of degree 7, we obtain precision to 15.85 decimal places, while with $\alpha = 1/6$ and the same polynomial degree 25.77 decimal places of the output can be computed. However, for simplicity, in this work we suggest to choose α and β equal to 1/2, as this option will not require non-trivial post-computation to compensate for the effect of computing the function on a different angle.

Based on the desired precision, one can retrieve the minimum necessary polynomial degree used in the approximation to achieve the desired precision, then look up the polynomial coefficients and evaluate the polynomial or polynomials on the input. As mentioned before, for trigonometric functions only polynomials with even or odd powers are used, and we have that sine is approximated as $xP(x^2)$ or $xP(x^2)/Q(x^2)$ (odd powers) and cosine as $P(x^2)$ or $P(x^2)/Q(x^2)$ (even powers). Thus, in when a rational function is used to evaluate sine, we obtain the following secure protocol. It assumes that the input is given in degrees in the range [0, 360).

$[b] \leftarrow \mathsf{Sin}([a])$

---

1. Compute $[s] = \mathsf{LT}(180, [a])$.
2. If $([s])$ then $[a] = [a] - 180$.

```
3. If ($\mathsf{LT}(90, [a])$) then $[a] = 180 - [a]$.
4. Compute $[x] = \frac{1}{90}[a]$ and then $[w] = [x]^2$.
5. Lookup the minimum polynomial degrees $N$ and $M$ using $\alpha = 1/2$ for which
   precision of the approximation is at least $k$ bits.
6. Lookup polynomial coefficients $p_0, \ldots, p_N$ and $q_0, \ldots, q_M$ for sine approximation.
7. Compute $([z_1], \ldots, [z_{\max(N,M)}]) \leftarrow \mathsf{PreMul}([w], \max(N, M))$.
8. Set $[y_P] = p_0 + \sum_{i=1}^{N} p_i[z_i]$.
9. Set $[y_Q] = q_0 + \sum_{i=1}^{M} q_i[z_i]$.
10. Compute $[y] \leftarrow \mathsf{Div}([y_P], [y_Q])$.
11. If ($[s]$) then $[b] = 0 - [x]$ else $[b] = [x]$.
12. Compute and return $[b] = [b] \cdot [y]$.
```

Recall that we recommend using a rational function in the form of $P(x)/Q(x)$ as in the above protocol for garbled circuit based implementation of high precision. With secret sharing, we modify the protocol to evaluate only a single polynomial P of (a different) degree N and skip the division operation in step 10. Also, based on our calculations, the single polynomial variant is also slightly faster in the garbled circuit setting when precision is not high (e.g., with 16 bits of precision).

As far as optimizations go, we, as usual, simultaneously execute independent operations in the secret sharing setting. Also, note that because coefficients p_i's and q_j's are not integers, evaluations of polynomials in steps 8 and 9 is not local in the secret sharing setting and requires truncation. We, however, can reduce the cost of truncating N (respectively, M) values to the cost of one truncation. This is because we first add the products and truncate the sum once. This also applies to polynomial evaluation in other protocols such as cosine and arctangent.

When input a is known to lie in the range $[0, 90]$ (as in the fingerprint algorithms we consider in this work), steps 1, 2, 3, 11, and 12 are not executed and instead we return $[y] \cdot [x]$ after evaluating step 10. Lastly, when input a is known to be an integer, the comparisons in steps 1 and 3 and the multiplication of $[a]$ and a fixed-point constant $\frac{1}{90}$ in step 4 become more efficient in both secret sharing and garbled circuit based implementations (i.e., both settings benefit from the reduced bitlength of a if only integer part is stored and the cost of the multiplication in the secret sharing setting becomes 1 interactive operation).

To show security of this protocol, we need to build simulators according to Definition 1. The main argument here is that because we only use secure building blocks that do not reveal any private information, we apply Canetti's composition theorem [44] to result in security of the overall construction. More precisely, to simulate the adversarial view, we invoke simulators corresponding to the building blocks. In more detail, if we assume an implementation based on a (np, t)-threshold linear secret sharing, our protocols inherit the same security guarantees as those of the building blocks (i.e., perfect or statistical security in the presence of secure channels between the parties with at most t corrupt computational parties) because no information about private values is revealed throughout the computation. More formally, to comply with the security definition, it is rather straightforward to build a simulator for our protocols by invoking simulators of the corresponding building blocks to result in the environment that will be indistinguishable from the real protocol execution by the participants. The same argument applies to other protocols presented in this work and we do not include explicit analysis.

The cosine function is evaluated similarly to sine. The main difference is in the way the input is pre- and post-processed for polynomial evaluation due to the behavior of this function. When cosine is evaluated using a rational function, we have the following secure protocol:

$[b] \leftarrow \mathsf{Cos}([a])$

```
1. If ($\mathsf{LT}(180, [a])$) then $[a] = 360 - [a]$.
2. $[s] = \mathsf{LT}(90, [a])$.
3. If ($[s]$) then $[a] = 180 - [a]$.
4. Compute $[x] = \frac{1}{90}[a]$ and then $[x] = [x]^2$.
5. Lookup the minimum polynomial degrees $N$ and $M$ using $\beta = 1/2$ for which
   precision of the approximation is at least $k$ bits.
6. Lookup polynomial coefficients $p_0, \ldots, p_N$ and $q_0, \ldots, q_M$ for cosine approxima-
   tion.
7. Compute $([z_1], \ldots, [z_{\max(N,M)}]) \leftarrow \mathsf{PreMul}([x], \max(N, M))$.
8. Set $[y_P] = p_0 + \sum_{i=1}^{N} p_i[z_i]$.
9. Set $[y_Q] = q_0 + \sum_{i=1}^{M} q_i[z_i]$.
10. Compute $[y] \leftarrow \mathsf{Div}([y_P], [y_Q])$.
11. If ($[s]$) then $[b] = 0 - [y]$ else $[b] = [y]$.
12. Return $[b]$.
```

The same optimizations as those described for the sine protocol apply to the cosine computation as well. Furthermore, to use a single polynomial to evaluate cosine, we similarly lookup coefficients for a single polynomial of (a different) degree N and skip steps 9 and 10 in the Cos protocol.

The complexities of Sin and Cos are provided in Tables 6.1 and 6.2. To achieve 16, 32, or 64 bits of precision, N needs to be set to 3, 5, or 9, respectively, for both Sin and Cos using a single polynomial, and (N, M) need to be set to (2, 1), (3, 2), and (6, 2), respectively, for Sin using the rational function approach.

TABLE 6.1

PERFORMANCE OF PROPOSED SECURE BUILDING BLOCKS BASED ON SECRET SHARING FOR FIXED-POINT VALUES

Protocol	Rounds	Interactive operations
Sin	$2\log N + 16$	$2Nk + 8\ell + 2N + 6k + 4$
Cos	$2\log N + 15$	$2Nk + 8\ell + 2N + 4k + 2$
Arctan	$2\log N$ $+3\log\ell + 2\times$ $\log(\frac{\ell}{3.5}) + 22$	$1.5\ell\log\ell + 2\ell\log(\frac{\ell}{3.5})$ $+2Nk + 18.5\ell + 2N$ $+4\log(\frac{\ell}{3.5}) + 6$
Sqrt	$0.5\ell + 2\log\ell$ $+6\xi + 24$	$2\ell^2 + \ell\log\ell$ $+3k(\xi+1) + 5\ell + 6\xi + 12$
Select	$c(2\log^2 t_1$ $+2\log^2 t_2$ $+6\log m + 2\times$ $\log\log m + 17)$	$c(\ell(t_1 - 0.25)(\log^2 t_1 + \log t_1 + 4)$ $+\ell(t_2 - 0.25)(\log^2 t_2 + \log t_2 + 4)$ $+2m\log m\log\log m + 8m\log m+$ $18m\ell - 5.5m + 2\log m - 8\ell + 9)$

In the case of inverse trigonometric functions and arctangent in particular, the function input domain is the entire $(-\infty, \infty)$ and the range of output is $(-\pi/2, \pi/2)$. We recall that arctangent is an odd function (i.e., $\arctan(-x) = -\arctan(x)$) and for all positive inputs we have $\arctan(x) + \arctan(\frac{1}{x}) = \pi/2$. It is, therefore, sufficient to approximate arctangent over the interval of $[0, 1]$. To this end, we use the technique introduced by Medina in [94] and its formal proof in [63]. In particular, [94] defines a sequence of polynomials over the input domain of $[0, 1]$, denoted as $h_N(x)$, with the property that $|h_N(x) - \arctan(x)| \leq \left(\frac{1}{4^{5/8}}\right)^{\deg(h_N)+1}$.

TABLE 6.2

PERFORMANCE OF PROPOSED SECURE BUILDING BLOCKS BASED ON GARBLED CIRCUIT FOR FIXED-POINT VALUES

Protocol	XOR gates	Non-XOR gates
Sin	$(\max(N,M)+N+M+2)$ $\times(4\ell^2-4\ell)+7\ell^2$ $+4\ell(N+M)+31\ell$	$(\max(N,M)+N+M+2)$ $\times(2\ell^2-\ell)+3\ell^2$ $+\ell(N+M)+11\ell$
Cos	$(\max(N,M)+N+M+1)$ $\times(4\ell^2-4\ell)+7\ell^2$ $+4\ell(N+M)+31\ell$	$(\max(N,M)+N+M$ $\times(2\ell^2-\ell)+3\ell^2$ $+\ell(N+M)+11\ell$
Arctan	$8N\ell^2+3\ell^2-4N\ell+43\ell$	$4N\ell^2+\ell^2-N\ell+15\ell$
Sqrt	$12\xi\ell^2+12.5\ell^2-k^2$ $+\ell k-8\xi\ell-7.5\ell+k-2$	$6\xi\ell^2+6.5\ell^2-k^2$ $+\ell k-2\xi\ell-0.5\ell+k-4$
Select	$c(1.5t_1\ell(\log^2 t_1+\log t_1+4)$ $+1.5t_2\ell(\log^2 t_2+\log t_2+4)$ $+(2\ell+20.5)m\log m+m$ $+12.5m\ell-6\log m-8\ell)$	$c(0.5t_1\ell(\log^2 t_1+\log t_1+4)$ $+0.5t_2\ell(\log^2 t_2+\log t_2+4)$ $+(4\ell+5)m\log m+m$ $+4m\ell+(2\ell-1)\log m)$

We use this formula to determine the degree N for any k-bit precision. This degree N is logarithmic with respect to the desired precision. Afterwards, the coefficients of $h_N(x)$ are computed from the recursive definitions in [94]. We choose this approach over other alternatives such as [68] for its efficiency. As an example, in [68] the authors propose to divide $[0,\infty)$ into $s+1$ segments for some value of s and perform a search to discover in which interval the input falls. Afterwards, the input is transformed into another variable (involving division) whose value is guaranteed to be within a small fixed interval and arctangent of this new variable is approximated using standard polynomial approximation techniques. In this approach, the search can be performed using binary search trees whose secure realization is non-trivial or alternatively will involve s steps. This extra computation makes the overall performance of the solution in [68] worse than that of [94]'s approach used here. Another candidate for arctangent evaluation is the well-known Taylor series of arctangent. However, Taylor series provides possibly reasonable precision if the input is bound to be very close to one point from the input domain, which we cannot assume. In addition, arctangent Taylor series converges very slowly. For example, for a three decimal precision on inputs around 0.95 Taylor series requires a polynomial of degree 57, whereas [94]'s approach requires a 7th-degree polynomial [63]. Our secure protocol to approximate arctangent based on the approach from [94] is given next:

```
$[b] \leftarrow \mathsf{Arctan}([a])$

1. Compute $[s] \leftarrow \mathsf{LT}([a], 0)$.
2. If $([s])$ then $[x] = 0 - [a]$ else $[x] = [a]$.
3. Compute $[c] \leftarrow \mathsf{LT}(1, [x])$.
4. If $([c])$ then $[d] = \pi/2$, $[y] \leftarrow \mathsf{Div}(1, [x])$; else $[d] = 0$, $[y] = [x]$.
5. Lookup the minimum polynomial degree $N$ for which precision of the approximation is at least $k$ bits.
6. Lookup polynomial coefficients $p_0, \ldots, p_N$ for arctangent approximation from [94].
7. Compute $([z_1], \ldots, [z_N]) \leftarrow \mathsf{PreMul}([y], N)$.
8. Set $[z] = p_0 + \sum_{i=1}^{N} p_i[z_i]$.
```

```
9. If ($[c]$) then $[z] = [d] - [z]$ else $[z] = [d] + [z]$.
10. If ($[s]$) then $[b] = 0 - [z]$ else $[b] = [z]$.
11. Return $[b]$.
```

The complexity of this protocol can be found in Tables 6.1 and 6.2.

6.5.1.2 Square Root

We now proceed with the square root computation defined by the interface $[b] \leftarrow \mathsf{Sqrt}([a])$, where a and b are fixed-point values to cover the general case.

Secure multi-party computation of the square root based on secret sharing has been treated by Liedel in [88]. The approach is based on the Goldschmidt's algorithm, which is faster than the Newton-Raphson's method. However, to eliminate the accumulated errors, the last iteration of the algorithm is replaced with the self-correcting iteration of the Newton-Raphson's method. For an ℓ-bit input a with k bits after the radix point, this protocol uses $2\ell^2 + \ell \log \ell + 3k(\xi + 1) + 5\ell + 6\xi + 12$ interactive operations in $0.5\ell + 2 \log \ell + 6\xi + 24$ rounds, where ℓ is assumed to be a power of 2 and $\xi = \lceil \log_2(\ell/5.4) \rceil$ is the number of algorithm iterations. For the purposes of this work, we use the protocol from [88] for the secret sharing setting. We also optimize the computation to be used with garbled circuit based on the specifics of that technique as described next.

Goldschmidt's method starts by computing an initial approximation for $\frac{1}{\sqrt{a}}$, denoted by b_0, that satisfies $\frac{1}{2} \leq ab_0^2 < \frac{3}{2}$. It then proceeds in iterations increasing the precision of the approximation with each consecutive iteration. To approximate $\frac{1}{\sqrt{a}}$, [88] uses an efficient method (a linear equation) that expects that input a is in the range $[\frac{1}{2}, 1)$. Thus, there is a need to first normalize a to a' such that $\frac{1}{2} \leq a' < 1$ and $a = a' \cdot 2^w$. Note that $\frac{1}{\sqrt{a}} = \frac{1}{\sqrt{a'}}\sqrt{2^{-w}}$, therefore, once we approximate $\frac{1}{\sqrt{a'}}$, we can multiply it by $\sqrt{2^{-w}}$ to determine an approximation of $\frac{1}{\sqrt{a}}$.

In our (ℓ, k)-bit fixed-point representation, a normalized input $a' \in [\frac{1}{2}, 1)$ has the most significant non-zero bit exactly at position $k-1$. We also express $\sqrt{2^{-w}}$ as $\frac{1}{\sqrt{2}} 2^{-\lfloor \frac{w}{2} \rfloor}$ when w is odd and as $2^{-\lfloor \frac{w}{2} \rfloor}$ when w is even. Our normalization procedure Norm that we present next thus takes input a and returns $\frac{1}{2} \le a' < 1$ (normalized input a in (ℓ, k)-bit fixed-point representation), $2^{-\lfloor \frac{w}{2} \rfloor}$ and bit c set to 1 when w is odd and to 0 otherwise. The protocol is optimized for the garbled circuit approach with cheap XOR gates [84]. It assumes that ℓ and k are even.

$\langle [a'], [2^{-\lfloor \frac{w}{2} \rfloor}], [c] \rangle \leftarrow \mathsf{Norm}([a])$

```
1. $([a_{\ell-1}], \ldots, [a_0]) \leftarrow [a]$.
2. Set $[x_{\ell-1}] = [a_{\ell-1}]$.
3. For $i = \ell-2, \ldots, 0$ do $[x_i] = [a_i] \vee [x_{i+1}]$.
4. Set $[y_{\ell-1}] = [x_{\ell-1}]$.
5. For $i = 0, \ldots, \ell-2$ do in parallel $[y_i] = [x_i] \oplus [x_{i+1}]$.
6. For $i = 0, \ldots, \ell-1$ do in parallel $[z^{(i)}] \leftarrow ([a_{\ell-1}] \wedge [y_i], \ldots, [a_0] \wedge [y_i])$.
7. Compute $[a'] = \left(\bigoplus_{i=0}^{k-1} \left([z^{(i)}] \ll (k-1-i)\right)\right) \oplus \left(\bigoplus_{i=k}^{\ell-1} \left([z^{(i)}] \gg (i-(k-1))\right)\right)$.
8. Let $[u_0] = [y_0]$, $[u_{\frac{\ell}{2}}] = [y_{\ell-1}]$ and for $i = 1, \ldots, \frac{\ell}{2}-1$ do in parallel $[u_i] =$
   $[y_{2i-1}] \oplus [y_{2i}]$.
9. Set $[2^{-\lfloor \frac{w}{2} \rfloor}] \leftarrow ([d_{\ell-1}], \ldots, [d_0]) = (0^{\ell-\frac{3k}{2}-1}, [u_0], [u_1], \ldots, [u_{\frac{\ell}{2}}], 0^{\frac{3k-\ell}{2}})$, where $0^x$
   corresponds to $x$ zeros.
10. Set $[c] = \bigoplus_{i=0}^{\frac{\ell}{2}-1} [y_{2i}]$.
11. Return $\langle [a'], [2^{-\lfloor \frac{w}{2} \rfloor}], [c] \rangle$.
```

Here, steps 2–3 preserve the most significant zero bits of a and set the remaining bits to 1 in variable x (i.e., all bits following the most significant non-zero bit are

1). Steps 4–5 compute y as a vector with the most significant non-zero bit of a set to 1 and all other bits set to 0. On step 6, each vector $z^{(i)}$ is either filled with 0s or set to a depending on the ith bit of y (thus, all but one $z^{(i)}$ can be non-zero). Step 7 computes the normalized value of a by aggregating all vectors $z^{(i)}$ shifted an appropriate number of positions. Here operation $x \ll y$ shifts ℓ-bit representation of x y positions to the left by discarding y most significant bits of x and appending y 0s in place of least significant bits. Similarly, $x \gg y$ shifts x to the right by prepending y 0s in place of most significant bits and discarding y least significant bits of x. Note that we can use cheap XOR gates for this aggregation operation because at most one y_i can take a non-zero value.

Steps 8 and 9 compute $[2^{-\lfloor \frac{w}{2} \rfloor}]$. Because a pair of consecutive i's results in the same value of w, we first combine the pairs on step 8 and shift them in place on step 9. As before, we can use cheap XOR and free shift operations to accomplish this task because at most one y_i is set. Lastly, step 10 computes the bit c, which is set by combining all flags y_i's at odd distances from $k-1$.

Note that for simplicity of exposition, we AND and XOR all bits in steps 6 and 7. There is, however, no need to compute the AND for the bits discarded in step 7 or compute the XOR with newly appended or prepended 0s. We obtain that this protocol can be implemented as a circuit using $0.5\ell^2 - k^2 + \ell k + 1.5\ell - 4$ non-XOR and $0.5\ell^2 - k^2 + \ell k + 0.5\ell - 3$ XOR gates.

Once we determine normalized input a', Liedel [88] approximates $\frac{1}{\sqrt{a'}}$, denoted by b'_0, by a linear equation $b'_0 = \alpha a' + \beta$, where α and β are precomputed. The values of these coefficients are set to $\alpha = -0.8099868542$ and $\beta = 1.787727479$ in [88] to compute an initial approximation b'_0 with almost 5.5 bits of precision (when a' is in the range $[\frac{1}{2}, 1)$). We then use b'_0 to compute b_0 that approximates $\frac{1}{\sqrt{a}}$ as described above.

After the initialization, Goldschmidt's algorithm sets $g_0 = xb_0$ and $h_0 = 0.5b_0$ and proceeds in ξ iterations that compute: $g_{i+1} = g_i(1.5 - g_i h_i)$, $h_{i+1} = h_i(1.5 - g_i h_i)$. The last iteration is replaced with one iteration of Newton-Raphson's method to eliminate accumulated errors that computes the following: $h_{i+1} = h_i(1.5 - 0.5ah_i^2)$. This gives us the following square root protocol, which we present optimized for the garbled circuit.

$[b] \leftarrow \mathsf{Sqrt}([a])$

```
1. Let $\xi = \lceil \log_2(\frac{\ell}{5.4}) \rceil$.
2. Execute $\langle [a'], [2^{-\lfloor \frac{w}{2} \rfloor}], [c] \rangle \leftarrow \mathsf{Norm}([a])$.
3. Let $\alpha = -0.8099868542$, $\beta = 1.787727479$ and compute $[b'_0] = \alpha \cdot [a'] + \beta$.
4. Compute $[b_0] = \Big( \Big( [c] \wedge \frac{1}{\sqrt{2}} \Big) \oplus \neg[c] \Big) \cdot [2^{-\lfloor \frac{w}{2} \rfloor}] \cdot [b'_0]$.
5. Compute $[g_0] = [a] \cdot [b_0]$ and $[h_0] = ([b_0] \gg 1)$.
6. For $i = 0, \ldots, \xi - 2$ do
    (a) $[x] = 1.5 - [g_i] \cdot [h_i]$.
    (b) if $(i < \xi - 2)$ then $[g_{i+1}] = [g_i] \cdot [x]$.
    (c) $[h_{i+1}] = [h_i] \cdot [x]$.
7. Compute $[h_\xi] = [h_{\xi-1}](1.5 - ([a] \cdot [h_{\xi-1}]^2 \gg 1))$.
8. Return $[b] = [h_\xi]$.
```

Here multiplication by 0.5 is replaced by shift to the right by 1 bit that has no cost. Complexity of this protocol can be found in Table 6.2.

6.5.1.3 Selection

There are multiple ways to find the fth smallest element of a set. The most straightforward method is to obliviously sort the input set and return the fth element of the sorted set. This approach is often beneficial when the input set is small. It is usually faster to use a selection algorithm, and the goal of this section is to design an oblivious selection protocol for finding the fth smallest element of a set.

The regular non-oblivious selection algorithms run in $O(m)$ time on m-element sets. We considered both deterministic and probabilistic algorithms and settled on a probabilistic algorithm that performs the best in the secure setting in practice. Its complexity is $O(m \log m)$ due to the use of data-oblivious compaction. Goodrich [67] proposed a probabilistic oblivious selection algorithm that works in linear time. Our starting point, however, is a different simpler algorithm that performs well in practice. Our algorithm proceeds by scanning through the input set and selects each element with probability $\frac{c_1}{\sqrt{m}}$ for some constant c_1. The expected number of selected elements is $O(\sqrt{m})$. The algorithm then sorts the selected elements and determines two elements x and y between which the fth smallest element is expected to lie based on f, m, and the number of selected elements. Afterwards, we scan the input set again retrieving all elements with values between x and y and simultaneously computing the ranks of x and y in the input set; let r_x and r_y denote the ranks. The expected number of retrieved elements is $O(\sqrt{m})$. We sort the retrieved elements and return the element at position $f - r_x$ in the sorted set. Let t_1 denote the number of elements randomly selected in the beginning of the algorithm and t_2 denote the number of elements with values between x and y.

Because of the randomized nature of our algorithm, it can fail at multiple points of its execution. In particular, if the number of elements selected in the beginning of the algorithm is too large or too small (and the performance guarantees cannot be met), we abort. Similarly, if the number of the retrieved elements that lie between x and y is too small, we abort. Lastly, if f does not lie between the ranks of x and y, the fth smallest element will not be among the retrieved elements and we abort. It can be shown using the union bound that all of the above events happen with a probability negligible in the input size. By using the appropriate choice of constants we can also ensure that faults are very rare in practice (see below). Thus, in the rare event of failure, the algorithm needs to be restarted using new random choices, and the expected number of times the algorithm is to be executed is slightly above 1.

Our protocol Select is given below. Compared to the algorithm structure outlined above, we need to ensure high probability of success through the appropriate selection of x and y as well as some constant parameters. We also need to ensure that the execution can proceed obliviously without compromising private data. To achieve this, we use three constants c_1, c_2, and $\hat{c}$. The value of c_1 influences the probability with which an element is selected in the beginning of the algorithm. The value of $\hat{c}$ influences the distance between x and y in the set of t_1 selected elements. In particular, we first determine the position where f is expected to be positioned among the t_1 selected elements as $(ft_1)/m$. We then choose x to be $\hat{c}$ elements before that position and choose y to be $3\hat{c}$ elements after that position. Lastly, c_2 is used to control the size of t_2. When t_2 is too small due to the random choices made during the execution, we abort.

To turn this logic into a data-oblivious protocol, we resort to secure and oblivious set operations. In particular, after we mark each element of the input set as selected or not selected, we use data-oblivious compaction to place all selected elements in the beginning of the set. Similarly, we use data-oblivious sorting to sort t_1 and t_2 elements.

$[b] \leftarrow \mathsf{Select}(\langle [a_1], \ldots, [a_m] \rangle, f)$

```
1. Set constants $c_1$, $c_2$, and $\hat{c}$; also set $n = \sqrt{m}$, $[t_1] = 0$, and $[t_2] = 0$.
2. For $i = 1, \ldots, m$ set $[b_i] = 0$.
3. For $i = 1, \ldots, m$ do in parallel
    (a) Generate random $[r_i] \in \mathbb{Z}_n$.
    (b) If $\mathsf{LT}([r_i], c_1)$ then $[b_i] = 1$.
4. For $i = 1, \ldots, m$ do $[t_1] = [t_1] + [b_i]$.
5. Open the value of $t_1$.
6. If $((t_1 > 2c_1 n) \vee (t_1 < \frac{1}{2}c_1 n))$ then abort.
7. Execute $\langle [a'_1], \ldots, [a'_m] \rangle \leftarrow \mathsf{Comp}(\langle [b_1], [a_1] \wedge [b_1] \rangle, \ldots, \langle [b_m], [a_m] \wedge [b_m] \rangle)$.
8. Execute $\langle [a''_1], \ldots, [a''_{t_1}] \rangle \leftarrow \mathsf{Sort}([a'_1], \ldots, [a'_{t_1}])$.
9. Compute $k$ as the closest integer to $(ft_1)/m$.
10. If $(k - \hat{c} < 1)$ then $[x] \leftarrow \mathsf{Min}([a_1], \ldots, [a_m])$; else $[x] = [a''_{k-\hat{c}}]$.
11. If $(k + 3\hat{c} > t_1)$ then $[y] \leftarrow \mathsf{Max}([a_1], \ldots, [a_m])$; else $[y] = [a''_{k+3\hat{c}}]$.
12. Set $[r_x] = 0$ and $[r_y] = 0$.
13. For $i = 1, \ldots, m$ do in parallel $[g_i] = \mathsf{LT}([a_i], [x])$ and $[g'_i] = \mathsf{LT}([a_i], [y])$.
14. For $i = 1, \ldots, m$ do in parallel $[b'_i] = \neg[g_i] \wedge [g'_i]$.
15. For $i = 1, \ldots, m$ do $[r_x] = [r_x] + [g_i]$ and $[t_2] = [t_2] + [b'_i]$.
16. Open the values of $r_x$ and $t_2$.
17. If $((t_2 < 4\hat{c}c_2 n) \vee ((f - r_x) < 0) \vee ((f - r_x) > t_2))$ then abort.
18. Execute $\langle [d_1], \ldots, [d_m] \rangle \leftarrow \mathsf{Comp}(\langle [b'_1], [a_1] \wedge [b'_1] \rangle, \ldots, \langle [b'_m], [a_m] \wedge [b'_m] \rangle)$.
19. Execute $\langle [d'_1], \ldots, [d'_{t_2}] \rangle \leftarrow \mathsf{Sort}([d_1], \ldots, [d_{t_2}])$.
20. Return $[b] = [d'_{f-r_x}]$.
```

Note that because maximum and minimum functions are evaluated on the same set, we reduce the overhead of evaluating them simultaneously compared to evaluating them individually. In particular, when min and max functions are evaluated in a tree-like fashion on an m-element set, we use the $m/2$ comparison in the first layer to set both minimum and maximum elements of each pair using the same cost as evaluating only one of them. The rest is evaluated as before, and we save about 1/2 overhead of one of min or max.

Complexity of this protocol is given in Tables 6.1 and 6.2 as a function of parameters m, t_1, t_2, c, and bitlength ℓ. The expected value of t_1 is $c_1\sqrt{m}$, the expected value of t_2 is $(4\hat{c}\sqrt{m})/c_1$, and c indicates the average number of times the algorithm needs to be invoked, which is a constant slightly larger than 1 (see below).

We experimentally determined the best values for c_1, c_2, and $\hat{c}$ for different values of n to ensure high success of the algorithm. Based on the experiments, we recommend to set $c_1 = 2$, $c_2 = 2$, and $\hat{c} = 10$. Using these values, the probability of the protocol's failure on a single run is at most a few percent up to input size 10,000. With larger input sizes, a larger value of $\hat{c}$ might be preferred.

Unlike all other protocols presented in this work, three values t_1, t_2, and r_x privately computed in Select are opened during its execution. Thus, to guarantee security, we need to show that these values are independent of the private input set and result in no information leakage. First, observe that the value of t_1 is computed purely based on random choices made in step 3(a) of the protocol. Thus, its value is data-independent. Second, the rank of x; r_x, is determined by random choices, but not the data values themselves. Similarly, the value of t_2 depends only on the ranks of the randomly chosen pivots, but not on the actual data values. Therefore, execution is data-oblivious and security is maintained.

6.6 Secure Fingerprint Recognition

We are now ready to proceed with the description of our solutions for secure fingerprint recognition using the building blocks introduced in Sections 3.2.1 and 6.5.1. We provide three constructions, one for each fingerprint recognition approach described in Section 6.3.

6.6.1 Secure Fingerprint Recognition Using Brute Force Geometrical Transformation

The easiest way to execute all (non-secure) fingerprint recognition algorithms in Section 6.3 is to use floating-point representation. We, however, choose to utilize integer and fixed-point arithmetic in their secure counterparts to reduce overhead associated with secure execution. In particular, typically minutia coordinates (x_i, y_i) as well as their orientation θ_i are represented as integers. This means that all inputs in Algorithm 2, the first fingerprint matching algorithm based on brute force geometrical transformation, are integers and computing the number of matched minutiae using Algorithm 3 can also be performed on integers. The output of sine and cosine functions in step 2(b) of Algorithm 2, however, are not integers and we utilize fixed-point representation for their values. Moreover, after the minutia points are transformed, we can truncate the transformed coordinates x''_i and y''_i and use their integer representation in Algorithm 3.

Our secure implementation of Algorithm 2 is given below as protocol $\mathsf{GeomTransFR}$. The secure computation uses the same logic as Algorithm 2 with the difference that the computation of the maximum matching is performed outside the main for-loop to reduce the number of rounds in the secret sharing setting.

$([C_{max}], \langle[\Delta x_{max}], [\Delta y_{max}], [\Delta\theta_{max}]\rangle) \leftarrow \mathsf{GeomTransFR}(T = \{t_i = ([x_i], [y_i], [\theta_i])\}_{i=1}^{m}, S = \{s_i = ([x'_i], [y'_i], [\theta'_i])\}_{i=1}^{n})$

---

1. $[C_{max}] = [0]$;
2. For $i = 1, \ldots, m$ and $j = 1, \ldots, n$, compute in parallel
   (a) $[\Delta x_{i,j}] = [x'_j] - [x_i]$, $[\Delta y_{i,j}] = [y'_j] - [y_i]$,and $[\Delta\theta_{i,j}] = [\theta'_j] - [\theta_i]$.
   (b) $[c_{i,j}] = \mathsf{Sin}([\Delta\theta_{i,j}])$ and $[c'_{i,j}] = \mathsf{Cos}([\Delta\theta_{i,j}])$.

(c) For $k = 1, \ldots, n$, compute in parallel $[x_{i,j}^{(k)}] = [c'_{i,j}] \cdot [x'_k] + [c_{i,j}] \cdot [y'_k] - [\Delta x_{i,j}]$, $[y_{i,j}^{(k)}] = [c'_{i,j}] \cdot [y'_k] - [c_{i,j}] \cdot [x'_k] - [\Delta y_{i,j}]$, and $[\theta_{i,j}^{(k)}] = [\theta'_k] - [\Delta \theta_{i,j}]$ and save the computed points as $S_{i,j} = \{([x_{i,j}^{(k)}], [y_{i,j}^{(k)}], [\theta_{i,j}^{(k)}])\}_{k=1}^{n}$.

(d) Compute the number $[C_{i,j}]$ of matched minutiae between T and $S_{i,j}$ using protocol Match.

3. Compute the largest matching $\langle [C_{max}], [\Delta x_{max}], [\Delta y_{max}], [\Delta \theta_{max}] \rangle = \mathsf{Max}$ $(\langle [C_{1,1}], [\Delta x_{1,1}], [\Delta y_{1,1}], [\Delta \theta_{1,1}] \rangle, \ldots, \langle [C_{m,n}]), [\Delta x_{m,n}]), [\Delta y_{m,n}], [\Delta \theta_{m,n}] \rangle)$.

4. Return $[C_{max}]$ and the corresponding alignment $\langle [\Delta x_{max}], [\Delta y_{max}], [\Delta \theta_{max}] \rangle$.

We obtain that all steps use integer arithmetic except step 2(b) (where sine and cosine have integer inputs, but fixed-point outputs) and multiplications in step 2(c) take one integer and one fixed-point operand (after which $x_{i,j}^{(k)}$ and $y_{i,j}^{(k)}$ are truncated to integer representation). Note that the latter corresponds to some optimization of the computation, where instead of converting integers x'_k and y'_k to fixed-point values and multiplying two fixed-point numbers, we can reduce the overhead by multiplying an integer to a fixed-point value. This eliminates the cost of truncating the product in the secret sharing setting and reduces the cost of multiplication in the garbled circuit setting. Furthermore, we can also optimize the point at which the fixed-point values are converted back to integers in the computation of $x_{i,j}^{(k)}$ and $y_{i,j}^{(k)}$ in step 2(c) of the algorithm. In particular, in the secret sharing setting we add the fixed-point products and $\Delta x_{i,j}^{(k)}$, $\Delta y_{i,j}^{(k)}$ converted to fixed-point representation (all of which have 0 cost), after which the sum is converted to an integer (paying the price of truncation). In the garbled circuit setting, on the other hand, we could convert the products to integers first (which has 0 cost) and then perform addition/subtraction on shorter integer values.

Our secure implementation of Algorithm 3 is given as protocol Match. All computation is performed on integer values. Compared to the structure of Algorithm 3, there are a few notable differences. First, note that instead of checking the

$\sqrt{(x_i - x'_j)^2 + (y_i - y'_j)^2} < \lambda$ constraint, the protocol checks the equivalent constraint $(x_i - x'_j)^2 + (y_i - y'_j)^2 < \lambda^2$ to avoid using a costly square root operation (and it is assumed that λ^2 is supplied as part of the input instead of λ). Second, to evaluate the constraint $\min(|\theta_i - \theta'_j|, 360 - |\theta_i - \theta'_j|) < \lambda_\theta$, instead of computing $|\theta_i - \theta'_j|$ the protocol computes $\theta_i - \theta'_j$ or $\theta'_j - \theta_i$ depending on which value is positive and uses that value in the consecutive computation. In addition, instead of computing the minimum, the computation proceeds as $(|\theta_i - \theta'_j| < \lambda_\theta) \vee (360 - |\theta_i - \theta'_j| < \lambda_\theta)$ in steps 3(c) and 3(d) of the protocol, which allows us to make the computation slightly faster. Because this computation is performed a large number of times, even small improvements can have impact on the overall performance.

Similar to restructuring protocol GeomTransFR to maximize parallelism and lower the number of rounds, in Match all distance and orientation constraints are evaluated in parallel in step 3. Then step 4 iterates through all minutiae in T and constructs a matching between a minutia of T and an available minutia from S within a close proximity to it (if any). Variable l_j indicates whether the jth minutia of fingerprint S is currently available (i.e., has not yet been paired up with another minutia from T). For each minutia, s_j, marked as unavailable, its distance to the ith minutia in T is set to λ^2 to prevent it from being chosen for pairing the second time. Because step 4(d) is written to run all loop iterations in parallel, there is a single variable C_j for each loop iteration, all values of which are added together at the end of the loop (which is free using secret sharing). With garbled circuit this parallelism is often not essential, and if the loop is executed sequentially, C can be incremented on step 4(d) directly instead of using C_j's. (And if parallel computation is used, the sum on step 4(e) will need to be implemented as the (free) XOR of all C_j.)

$[C] \leftarrow \mathsf{Match}(T = \{t_i = ([x_i], [y_i], [\theta_i])\}_{i=1}^m, S = \{s_i = ([x'_i], [y'_i], [\theta'_i])\}_{i=1}^n, \lambda^2, \lambda_\theta)$

```
1. Set $[C] = [0]$.
2. For $j = 1, \ldots, n$, set in parallel $[l_j] = [0]$.
3. For $i = 1, \ldots, m$ and $j = 1, \ldots, n$, compute in parallel
   (a) $[d_{i,j}] = ([x_i] - [x'_j])^2 + ([y_i] - [y'_j])^2$.
   (b) If $\mathsf{LT}([\theta_i], [\theta'_j])$, then $[a_{i,j}] = [\theta'_j] - [\theta_i]$, else $[a_{i,j}] = [\theta_i] - [\theta'_j]$.
   (c) $[c_{i,j}] = \mathsf{LT}([d_{i,j}], [\lambda^2])$, $[c'_{i,j}] = \mathsf{LT}([a_{i,j}], [\lambda_\theta])$, and $[c''_{i,j}] = \mathsf{LT}((360 - [a_{i,j}]), [\lambda_\theta])$.
   (d) $[v_{i,j}] = [c_{i,j}] \wedge ([c'_{i,j}] \vee [c''_{i,j}])$.
4. For $i = 1, \ldots, m$, do
   (a) For $j = 1, \ldots, n$, do in parallel if $([l_j] \vee \neg[v_{i,j}])$, then $[d_{i,j}] = \lambda^2$.
   (b) Execute $([d_{min}], [j_{min}]) = \mathsf{Min}([d_{i,1}], \ldots, [d_{i,n}])$.
   (c) Set $[u] = \mathsf{LT}([d_{min}], \lambda^2)$.
   (d) For $j = 1, \ldots, n$, compute in parallel if $(\mathsf{EQ}([j_{min}], j) \wedge [u])$, then $[C_j] = [1]$
       and $[l_j] = [1]$, else $[C_j] = [0]$.
   (e) $[C] = [C] + \sum_{j=1}^{n} [C_j]$.
5. Return $[C]$.
```

The asymptotic complexity of GeomTransFR and Match remains similar to their original non-secure counterparts. In particular, if we treat the bitlength of integer and fixed-point values as constants, the complexity of GeomTransFR is $O(n^2m^2)$ and Match is $O(nm)$. Their round complexity (for the secret sharing setting) is $O(m \log n)$ and $O(m \log n)$, respectively. If we wish to include dependency on the bitlengths ℓ and k, the complexity increases by at most a factor of $O(\ell)$ in the secret sharing setting and at most a factor of $O(\ell^2)$ in the garbled circuit setting due to the differences in the complexity of the underlying building blocks.

6.6.2 Secure Fingerprint Recognition Using High Curvature Points for Alignment

We next treat our secure realization of fingerprint recognition using high curvature points from Algorithm 4. The corresponding secure computation is provided as protocol HighCurvatureFR below. It makes calls to a secure version of Algorithm 5, which we consequently call as protocol OptimalMotion. Also, because of the complexity of the computation associated with finding the closest points in step 3 of Algorithm 4, we provide the corresponding secure computation as a separate protocol ClosestPoints. All computation is performed on fixed-point values with the exception of step 7 of HighCurvatureFR, where a call to the minutia pairing protocol Match is made.

$([C],(R,v)) \leftarrow \mathsf{HighCurvatureFR}(T = (\{t_i = ([x_i],[y_i],[\theta_i])\}_{i=1}^{m}, \{\hat{t}_i = ([\hat{x}_i],[\hat{y}_i],[\hat{w}_i])\}_{i=1}^{\hat{m}}),$
$S = (\{s_i = ([x'_i],[y'_i],[\theta'_i])\}_{i=1}^{n}, \{\hat{s}_i = ([\hat{x}'_i],[\hat{y}'_i],[\hat{w}'_i])\}_{i=1}^{\hat{n}}), f, \gamma, (\alpha_1,\ldots,\alpha_\gamma), \beta, \lambda^2, \lambda_\theta)$

```
1. Set $[S_{LTS}] = [0]$.
2. For $i = 1, \ldots, \hat{n}$, set $\bar{s}_i = \hat{s}_i$.
3. For $\mathsf{ind} = 1, \ldots, \gamma$ do
    (a) Execute $\{([d_i], \tilde{t}_i)\}_{i=1}^{\hat{n}} \leftarrow \mathsf{ClosestPoints}(\{\hat{t}_i\}_{i=1}^{\hat{m}}, \{\hat{s}_i\}_{i=1}^{\hat{n}}, \alpha_{\mathsf{ind}})$.
    (b) Execute $[y] \leftarrow \mathsf{Select}(([d_1], \ldots, [d_{\hat{n}}]), f)$.
    (c) For $i = 1, \ldots, \hat{n}$ do in parallel $[l_i] = \mathsf{LT}([d'_i], [y] + 1)$.
    (d) Execute $(([a_1], b_1, c_1), \ldots, ([a_{\hat{n}}], b_{\hat{n}}, c_{\hat{n}})) \leftarrow \mathsf{Comp}(([l_1], \hat{s}_1 \wedge [l_1], \tilde{t}_1 \wedge [l_1]), \ldots,$
        $([l_{\hat{n}}], \hat{s}_{\hat{n}} \wedge [l_{\hat{n}}], \tilde{t}_{\hat{n}} \wedge [l_{\hat{n}}]))$ using $[l_i]$'s as the keys.
    (e) Compute the optimal motion $(R, v) \leftarrow \mathsf{OptimalMotion}$ $(\{c_i, b_i\}_{i=1}^{f})$.
    (f) For $i = 1, \ldots, \hat{n}$ transform the points in parallel as $[x''_i] = [v_1] + [r_{11}] \cdot [\hat{x}'_i] +$
        $[r_{12}] \cdot [\hat{y}'_i] + [r_{13}] \cdot [\hat{w}'_i]$, $[y''_i] = [v_2] + [r_{21}] \cdot [\hat{x}'_i] + [r_{22}] \cdot [\hat{y}'_i] + [r_{23}] \cdot [\hat{w}'_i]$, and
        $[w''_i] = [v_3] + [r_{31}] \cdot [\hat{x}'_i] + [r_{32}] \cdot [\hat{y}'_i] + [r_{33}] \cdot [\hat{w}'_i]$, then set $\hat{s}_i = ([x''_i], [y''_i], [w''_i])$.
4. Execute $(R, v) \leftarrow \mathsf{OptimalMotion}(\{\bar{s}_i, \hat{s}_i\}_{i=1}^{\hat{n}})$.
```

5. Compute $[c_1] = \mathsf{Div}([\bar{y}_2] - [\bar{y}_1], [\bar{x}_2] - [\bar{x}_1])$, $[c_2] = \mathsf{Div}([\hat{y}'_2] - [\hat{y}'_1], [\hat{x}'_2] - [\hat{x}'_1])$, $[c_3] = \mathsf{Div}([c_1] - [c_2], 1 + [c_1] \cdot [c_2])$, and $[\Delta\theta] = \mathsf{Arctan}([c_3])$.

6. For $i = 1, \ldots, n$ do in parallel $[x''_i] = [v_1] + [r_{11}] \cdot [x'_i] + [r_{12}] \cdot [y'_i]$ and $[y''_i] = [v_2] + [r_{21}] \cdot [x'_i] + [r_{22}] \cdot [y'_i]$, then set $s_i = ([x''_i], [y''_i], [\theta'_i] - [\Delta\theta])$.

7. Compute the number $[C]$ of matched minutiae by executing Match $(\{t_i\}_{i=1}^m, \{s_i\}_{i=1}^n, \lambda^2, \lambda_\theta)$.

8. Return $[C], (R, v)$.

Different from the computation in Algorithm 4, our protocol HighCurvatureFR proceeds using the maximum number of iterations γ, as not to reveal the actual number of iterations needed (which depends on private inputs). The remaining algorithm's structure is maintained, where the computation is replaced with secure and data-oblivious operations. In particular, in each iteration of step 3, we first determine the closest high-curvature point from T to each (possibly transformed) high-curvature point from S and compute f pairs with the smallest distances. Secure computation of the closest points using protocol ClosestPoints is described below, while determining the closest f pairs is performed on steps 3(b)–(d) as follows. After selecting the fth smallest element y among the computed distances (step 3(b)), each distance is compared to that element (step 3(c)). We then proceed with pushing all elements which are less than y to the beginning of the set using compaction (step 3(d)) and consequently use the f smallest distances (located in the beginning of the set) for optimal motion computation.

Once the closest f pairs and the corresponding optimal motion have been determined, the protocol proceeds with applying the optimal motion to the high-curvature points in S. After executing this computation a necessary number of times, the protocol computes the overall motion in step 4 and applies it to the minutia points in the same way as in Algorithm 4. One thing to notice is that HighCurvatureFR has additional (public) inputs compared to Algorithm 4. The parameters $\alpha_1, \ldots, \alpha_\gamma$ specify bounding box sizes for the purposes of closest points computation in step 3(a) (see below). The remaining new parameters λ^2, λ_θ and β are made explicit as inputs to the minutia pairing protocol Match and the scaling factor in distance computation (step 3 of Algorithm 4).

The protocol $\mathsf{ClosestPoints}$ below takes two sets of points t_i's and s_i's (represented using three coordinates each) and for each point s_i returns the closest element among the t_i's and the corresponding distance. As mentioned in Section 6.3.2, the TICP algorithm on which our construction builds uses the bounding box approach to eliminate errors during this computation. In particular, only points within a certain distance from a point are considered, and the maximum allowable distance (i.e., the bounding box size) is denoted by α_i in the ith iteration of $\mathsf{HighCurvatureFR}$ or Algorithm 4. The sizes of the bounding boxes become smaller with each iteration as the two sets of points become closer to each other. When we make a call to $\mathsf{ClosestPoints}$, we pass a single bounding box size for the current iteration and that parameter is denoted as α in the interface of $\mathsf{ClosestPoints}$.

```
$\{([d_i], \tilde{t}_i)\}_{i=1}^{\hat{n}} \leftarrow \mathsf{ClosestPoints}(\{t_i = (x_i, y_i, w_i)\}_{i=1}^{\hat{m}}, \{s_i = (x'_i, y'_i, w'_i)\}_{i=1}^{\hat{n}}, \alpha, \beta)$

1. For $i = 1, \ldots, \hat{n}$ and for $j = 1, \ldots, \hat{m}$ do in parallel
    (a) Set $[d_{(i,j)}] = \mathsf{Sqrt}([\hat{x}_j] - [\hat{x}'_i])^2 + ([\hat{y}_j] - [\hat{y}'_i])^2$.
    (b) If $(\mathsf{LT}([\hat{w}_j], [\hat{w}'_i]))$ then $[a_{(i,j)}] = [\hat{w}'_i] - [\hat{w}_j]$; else $[a_{(i,j)}] = [\hat{w}_j] - [\hat{w}'_i]$.
    (c) Set $[d_{(i,j)}] + \beta \cdot [a_{(i,j)}]$.
2. For $i = 1, \ldots, \hat{m}$ do in parallel $[l_i] = 1$.
3. For $i = 1, \ldots, \hat{n}$ do in parallel $[l'_i] = 1$.
4. For $i = 1, \ldots, \hat{n}$ do
    (a) For $j = 1, \ldots, \hat{m}$ do in parallel if $(\neg[l_j])$ then $[d_{(i,j)}] = \infty$.
```

```
        (b) Execute $([d_i], [j_{min}]) \leftarrow \mathsf{Min}([d_{(i,1)}], \ldots, [d_{(i,\hat{m})}])$.
        (c) $[b] = \mathsf{LT}([d_i], \alpha)$.
        (d) For $j = 1, \ldots, \hat{m}$ do in parallel if $(\mathsf{EQ}([j_{min}], j) \wedge [b])$ then $[l_j] = 0$, $[l'_i] = 0$,
            $\tilde{t}_i = t_j$.
    5. For $i = 1, \ldots, \hat{n}$ do
        (a) For $j = 1, \ldots, \hat{m}$ do in parallel if $(\neg [l_j])$ then $[d_{(i,j)}] = \infty$.
        (b) Execute $([d_i], [j_{min}]) \leftarrow \mathsf{Min}([d_{(i,1)}], \ldots, [d_{(i,\hat{m})}])$.
        (c) For $j = 1, \ldots, \hat{m}$ do in parallel if $(\mathsf{EQ}([j_{min}], j) \wedge [l'_i])$ then $[l_j] = 0$, $[l'_i] = 0$,
            $\tilde{t}_i = t_j$.
    6. Return $\{([d_i], \tilde{t}_i)\}_{i=1}^{\hat{n}}$.
```

Given two sets of points and public parameter α, the ClosestPoints protocol first computes the distances between each pair of points in parallel in step 1 (according to the formula in step 3 of Algorithm 4). Next, we mark each t_i and s_i as available (steps 2 and 3, respectively). Step 4 iterates through all s_i's and determines the closest available point t_j to s_i (the distance to the unavailable points is set to infinity to ensure that they are not chosen). If the closest point is within the bounding box, s_i is paired up with t_j and both are marked as unavailable.

At the end of step 4, some points s_i's will be paired up with one of the t_j's, while others will not be. To ensure that the algorithm produces enough pairs for their consecutive use in HighCurvatureFR, we repeat the pairing process with the s_i's that remain available at this point and without enforcing the constraint that the points of the pair must lie in close proximity of each other. This computation corresponds to step 5. That is, this step pairs each available s_i with the closest available point among the t_j's even if it is far away from s_i (and is likely to be an unrelated point). This is to ensure that enough distances are returned for their use in the parent protocol. In this step, distances of all unavailable points are set to infinity and each s_i which is still marked as available is updated with the closest distance and the corresponding t_j.

What remains is to discuss protocol OptimalMotion that corresponds to secure evaluation of the computation in Algorithm 5 and is given next. The computation in OptimalMotion follows the steps of Algorithm 5 and omit its detailed description here. We re-arrange some operations in this protocol to reduce the number of expensive operations.

$(R, v) \leftarrow \mathsf{OptimalMotion}(\{(t_i = ([x_i], [y_i], [z_i]), s_i = ([x'_i], [y'_i], [z'_i]))\}_{i=1}^n)$

---

1. For $i = 1, \ldots, n$ do in parallel
   (a) Compute $[k'_i] = [x_i] \cdot [x'_i] + [y_i] \cdot [y'_i] + [z_i] \cdot [z'_i]$, $[k''_i] = \mathsf{Sqrt}(([x_i]^2 + [y_i]^2 + [z_i]^2)([x'_i]^2 + [y'_i]^2 + [z'_i]^2))$, and $[k_i] = \mathsf{Div}([k'_i], [k''_i])$.
   (b) Compute $[p_{(i,1)}] = \mathsf{Sqrt}(\frac{1}{2} + \frac{1}{2} \cdot [k_i])$, $[p_{(i,2)}] = \mathsf{Sqrt}(\frac{1}{2} - \frac{1}{2} \cdot [k_i])$.
   (c) Compute $[b_i] = [y_i] \cdot [z'_i] - [z_i] \cdot [y'_i]$, $[b'_i] = [z_i] \cdot [x'_i] - [x_i] \cdot [z'_i]$, and $[b''_i] = [x_i] \cdot [y'_i] - [y_i] \cdot [x'_i]$.
   (d) Compute $[u'_i] = \mathsf{Div}(1, \mathsf{Sqrt}([b_i]^2 + [b'_i]^2 + [b''_i]^2))$ and $u_i = ([u_{(i,1)}], [u_{(i,2)}], [u_{(i,3)}]) = ([b_i] \cdot [u'_i]), ([b'_i] \cdot [u'_i]), ([b''_i] \cdot [u'_i])$.
   (e) Compute and set $q'_i = ([q'_{(i,1)}], [q'_{(i,2)}], [q'_{(i,3)}], [q'_{(i,4)}]) = ([p_{(i,1)}], [p_{(i,2)}] \cdot [u_{(i,1)}], [p_{(i,2)}] \cdot [u_{(i,2)}], [p_{(i,2)}] \cdot [u_{(i,3)}])$.
2. Set $q = ([q_1], [q_2, [q_3], [q_4]) = q'_1$.
3. For $i = 2, \ldots, n$ compute $[q''_1] = [q_1] \cdot [q'_{(i,1)}] - [q_2] \cdot [q'_{(i,2)}] - [q_3] \cdot [q'_{(i,3)}] - [q_4] \cdot [q'_{(i,4)}]$, $[q''_2] = [q_1] \cdot [q'_{(i,2)}] + [q'_{(i,1)}] \cdot [q_2] + [q_3] \cdot [q'_{(i,4)}] - [q_4] \cdot [q'_{(i,3)}]$, $[q''_3] = [q_1] \cdot [q'_{(i,3)}] + [q'_{(i,1)}] \cdot [q_3] + [q_2] \cdot [q'_{(i,4)}] - [q_4] \cdot [q'_{(i,2)}]$, and $[q''_4] = [q_1] \cdot [q'_{(i,4)}] + [q'_{(i,1)}] \cdot [q_4] + [q_2] \cdot [q'_{(i,3)}] - [q_3] \cdot [q'_{(i,2)}]$, then set $[q_1] = [q''_1]$, $[q_2] = [q''_2]$, $[q_3] = [q''_3]$, and $[q_4] = [q''_4]$.
4. Compute $[\hat{q}_{(1,1)}] = [q_1]^2$, $[\hat{q}_{(2,2)}] = [q_2]^2$, $[\hat{q}_{(3,3)}] = [q_3]^2$, $[\hat{q}_{(4,4)}] = [q_4]^2$, $[\hat{q}_{(2,3)}] = [q_2] \cdot [q_3]$, $[\hat{q}_{(1,4)}] = [q_1] \cdot [q_4]$, $[\hat{q}_{(2,4)}] = [q_2] \cdot [q_4]$, $[\hat{q}_{(1,3)}] = [q_1] \cdot [q_3]$, $[\hat{q}_{(3,4)}] = [q_3] \cdot [q_4]$, and $[\hat{q}_{(1,2)}] = [q_1] \cdot [q_2]$.
5. Compute matrix $R = \{[r_{ij}]\}_{i,j=1}^3$, where $[r_{11}] = [\hat{q}_{(1,1)}] + [\hat{q}_{(2,2)}] - [\hat{q}_{(3,3)}] - [\hat{q}_{(4,4)}]$,

$[r_{12}] = 2 \cdot ([\hat{q}_{(2,3)}] - [\hat{q}_{(1,4)}])$, $[r_{13}] = 2 \cdot ([\hat{q}_{(2,4)}] + [\hat{q}_{(1,3)}])$, $[r_{21}] = 2 \cdot ([\hat{q}_{(2,3)}] + [\hat{q}_{(1,4)}])$, $[r_{22}] = [\hat{q}_{(1,1)}] - [\hat{q}_{(2,2)}] + [\hat{q}_{(3,3)}] - [\hat{q}_{(4,4)}]$, $[r_{23}] = 2 \cdot ([\hat{q}_{(3,4)}] - [\hat{q}_{(1,2)}])$, $[r_{31}] = 2 \cdot ([\hat{q}_{(2,4)}] - [\hat{q}_{(1,3)}])$, $[r_{32}] = 2 \cdot ([\hat{q}_{(3,4)}] - [\hat{q}_{(1,2)}])$, and $[r_{33}] = [\hat{q}_{(1,1)}] - [\hat{q}_{(2,2)}] - [\hat{q}_{(3,3)}] + [\hat{q}_{(4,4)}]$;

6. Compute $[t''_1] = \frac{1}{n} \cdot \Sigma_{i=1}^{n} [x_i]$, $[t''_2] = \frac{1}{n} \cdot \Sigma_{i=1}^{n} [y_i]$, $[t''_3] = \frac{1}{n} \cdot \Sigma_{i=1}^{n} [z_i]$;
7. Compute $[s''_1] = \frac{1}{n} \cdot \Sigma_{i=1}^{n} [x'_i]$, $[s''_2] = \frac{1}{n} \cdot \Sigma_{i=1}^{n} [y'_i]$, $[s''_3] = \frac{1}{n} \cdot \Sigma_{i=1}^{n} [z'_i]$;
8. Compute vector $v = \{[v_i]\}_{i=1}^{3}$, where $[v_1] = [t''_1] + [r_{11}] \cdot [s''_1] + [r_{12}] \cdot [s''_2] + [r_{13}] \cdot [s''_3]$, $[v_2] = [t''_2] + [r_{21}] \cdot [s''_1] + [r_{22}] \cdot [s''_2] + [r_{23}] \cdot [s''_3]$, $[v_3] = [t''_3] + [r_{31}] \cdot [s''_1] + [r_{32}] \cdot [s''_2] + [r_{33}] \cdot [s''_3]$.
9. Return (R, v).

Note that here many steps are independent of each other and can be carried out in parallel. For examples steps 1(a)–(b) and 1(c)–(d) correspond to independent branches of computation. Furthermore, some (rather cheap) redundant operations are retained in the protocol for readability, while an implementation would execute them only once. Additional small optimizations are also possible. For example, a number of multiplications in step 1 correspond to multiplication of integer and fixed-point operands, which can be implemented faster than regular fixed-point multiplication.

6.6.3 Secure Fingerprint Recognition based on Spectral Minutia Representation

In this section, we present our third secure fingerprint recognition protocol based on spectral minutia representation called SpectralFR. The construction builds on Algorithm 6 and incorporates both types of feature reduction (CPCA and LDFT) not included in Algorithm 6. Recall that in the second type of feature reduction, LDFT, (or when both types of feature reduction are applied) rotation of fingerprint/matrix S is performed by multiplying each cell of S by value $e^{-i\frac{2\pi}{N}j\alpha}$, where j is the cell's row and α is the amount of rotation. While it is possible to implement rotations by providing $2\lambda + 1$ different copies of S rotated by different amounts as input into the protocol where λ is the maximum amount of rotation, we perform any necessary rotation inside the protocol to avoid the price of significantly increasing the input size. We were also able to maintain performing only $O(\log \lambda)$ score computations instead of all $2\lambda + 1$ scores in our secure and oblivious protocol, which is an important contribution of this work. Combined with avoiding to increase the input size to have a linear dependency on λ, this allows us to achieve low asymptotic complexity and high efficiency. Low asymptotic complexity is important because the size of fingerprint representation is already large in this approach.

In what follows, we treat the case when $\lambda = 15$ (with the total of 31 rotations to be considered) as in the original algorithm [114], but it is not difficult to generalize the computation to any λ. We apply our generalization and optimization of Algorithm 6 described in Section 6.3.3 with $4 \cdot 3^2 = 36$ different rotations to cover at least 31 necessary alignments. Note that this approach computes 8 similarity scores instead of 9 in the original algorithm. We number all 36 rotations as $\alpha = -17, \ldots, 18$. The rotation constants $e^{-i\frac{2\pi}{N}j\alpha} = \cos(-\frac{2\pi}{N}j\alpha) + i\sin(-\frac{2\pi}{N}j\alpha)$ are fixed and can be precomputed for each $j = 1, \ldots, N'$ and $\alpha = -17, \ldots, 18$ by each party. We denote these values by public matrix $Z = \{z_{\alpha,j}\}_{\alpha=-17,j=1}^{18,N'}$ which each party stores locally. Each $z_{\alpha,j}$ is a tuple $(z_{\alpha,j}^{(1)} = \cos(-\frac{2\pi}{N}j\alpha), z_{\alpha,j}^{(2)} = \sin(-\frac{2\pi}{N}j\alpha))$, and Z is specified as part of the input in SpectralFR.

To be able to compute only $O(\log \lambda)$ scores in the protocol, we need to obliviously determine the correct amount of rotation in steps 3 and 4 of Algorithm 6 without revealing any information about k or k' in those steps. We securely realize this functionality by placing the values associated with each possible k or k' in an array and retrieving the right elements at a private index. In more detail, the protocol first computes four scores that correspond to rotations by -13, -4, 3, and 12 positions and computes the best among them (steps 3 and 4 below). Because the location of the best score cannot be revealed, in the next step of the algorithm we put together an array consisting of four vectors and one of them is privately selected using TLookup (steps 4–5 of the protocol). The selected vector consists of $2N'$ values that allow us to compute two new scores and the maximum score for the next iteration of algorithm. We repeat the process of private retrieval of the necessary rotation coefficients, this time using an array consisting of twelve vectors. After computing two more scores and determining the best score, the algorithm outputs the best score together with the amount of rotation that resulted in that score.

The protocol SpectralFR is given next. Because SpectralFR corresponds to the computation with both types of feature reduction, input matrices T and S are composed of complex values. Thus, we represent each cell of T as a pair $(a_{i,j}, b_{i,j})$ with the real and imaginary parts, respectively, and each cell of S as a pair $(a'_{i,j}, b'_{i,j})$. All computations proceed over fixed-point values.

$([C_{max}], [\alpha_{max}]) \leftarrow$ SpectralFR($T = \{[t_{i,j}] = ([a_{i,j}], [b_{i,j}])\}_{i=1,j=1}^{M',N'}$, $S = \{[s_{i,j}] = ([a'_{i,j}], [b'_{i,j}])\}_{i=1,j=1}^{M',N'}$, $Z = \{z_{i,j}\}_{i=-17,j=1}^{18,N'}$, $\lambda = 15$)

---

1. Set $[C_{max}] = [0]$.

2. For $i = 1, \ldots, M'$ and $j = 1, \ldots, N'$ do in parallel if $(j \neq 1)$ then $[x_{i,j}] = 2([a_{i,j}]\cdot[a'_{i,j}]+[b_{i,j}]\cdot[b'_{i,j}])$, $[y_{i,j}] = 2([a'_{i,j}]\cdot[b_{i,j}]-[a_{i,j}]\cdot[b'_{i,j}])$, else $[x_{i,j}] = [a_{i,j}]\cdot[a'_{i,j}]$, $[y_{i,j}] = -[a_{i,j}]\cdot[b'_{i,j}]$.

3. For $k = 0, \ldots, 3$, do in parallel

   (a) $[C_{-13+9k}] = 0$.

   (b) For $i = 1, \ldots, M'$ and $j = 1, \ldots, N'$ do $[C_{-13+9k}] = [C_{-13+9k}] + z^{(1)}_{-13+9k,j} \cdot [x_{i,j}] + z^{(2)}_{-13+9k,j} \cdot [y_{i,j}]$.

4. Compute the maximum as $([C_{max}], [\alpha_{max}]) \leftarrow$ Max($[C_{-13}], [C_{-4}], [C_5], [C_{14}]$).

5. For $i = 0, \ldots, 3$ and $j = 1, \ldots, N'$ do in parallel $z'_{i,j} = z_{-16+9i,j}$ and $z'_{i,N'+j} = z_{-10+9i,j}$.

6. Let $Z'_i = (z'_{i,1}, \ldots, z'_{i,2N'})$ for $i = 0, \ldots, 3$ and execute $[Z'_{max}] \leftarrow \mathsf{TLookup}((Z'_0, \ldots, Z'_3), [\alpha_{max}])$; let $[Z'_{max}] = ([\hat{z}_1], \ldots, [\hat{z}_{2N'}])$.

7. For $i = 1, \ldots, M'$ and $j = 1, \ldots, N'$ do $[C_{\alpha_{max}-3}] = [C_{\alpha_{max}-3}] + [\hat{z}_j^{(1)}] \cdot [x_{i,j}] + [\hat{z}_j^{(2)}] \cdot [y_{i,j}]$, $[C_{\alpha_{max}+3}] = [C_{\alpha_{max}+3}] + [\hat{z}_{N'+j}^{(1)}] \cdot [x_{i,j}] + [\hat{z}_{N'+j}^{(2)}] \cdot [y_{i,j}]$.

8. Compute $([C_{max}], [\alpha_{max}]) \leftarrow \mathsf{Max}([C_{\alpha_{max}-3}], [C_{max}], [C_{\alpha_{max}+3}])$.

9. For $i = 0, \ldots, 11$ and $j = 1, \ldots, N'$ do in parallel $z'_{i,j} = z_{-17+3i,j}$ and $z'_{i,N'+j} = z_{-15+3i,j}$.

10. Let $Z'_i = (z'_{i,1}, \ldots, z'_{i,2N'})$ for $i = 0, \ldots, 11$ and execute $[Z'_{max}] \leftarrow \mathsf{TLookup}((Z'_0, \ldots, Z'_{11}), [\alpha_{max}])$; let $[Z'_{max}] = ([\hat{z}_1], \ldots, [\hat{z}_{2N'}])$.

11. For $i = 1, \ldots, M'$ and $j = 1, \ldots, N'$ do $[C_{\alpha_{max}-1}] = [C_{\alpha_{max}-1}] + [\hat{z}_j^{(1)}] \cdot [x_{i,j}] + [\hat{z}_j^{(2)}] \cdot [y_{i,j}]$, $[C_{\alpha_{max}+1}] = [C_{\alpha_{max}+1}] + [\hat{z}_{N'+j}^{(1)}] \cdot [x_{i,j}] + [\hat{z}_{N'+j}^{(2)}] \cdot [y_{i,j}]$.

12. Compute $([C_{max}], [\alpha_{max}]) \leftarrow \mathsf{Max}([C_{\alpha_{max}-1}], [C_{max}], [C_{\alpha_{max}+1}])$.

13. Return $[C_{max}]$ and $[\alpha_{max}]$.

For efficiency reasons, we perform all multiplications associated with the score computation without any rotation in the beginning of the protocol (step 1). This computation (i.e., multiplications of the real and imaginary components of each $t_{i,j}$ and $s_{i,j}$) is common to all score computations (see Equation 6.1) and is reused later in the protocol. Then to compute a score between T and S rotated by α positions, the coefficients from Z are multiplied to the computed products. In particular, the computation takes form of $\{[z_{\alpha,j}^{(1)}] \cdot [x_{i,j}]\}_{i=1,j=1}^{M',N'}$ and $\{[z_{\alpha,j}^{(2)}] \cdot [y_{i,j}]\}_{i=1,j=1}^{M',N'}$ according to Equation 6.1, which are added together to get the score. The rest of the protocol proceeds as described above by using private table lookups twice. Note that the result of each table lookup is an array as opposed to a single element. Also note that the score computation uses public $z_{i,j}$'s in step 1(b), but the coefficients become private in steps 7 and 11 because they depend of private data. Finally, the factor $\frac{1}{MN^2}$ present in Equation 6.1 is not included in the computation because it is public. The computed score can be scaled down by this factor by an output recipient if necessary.

Recall that SpectralFR uses fixed-point arithmetic, but performance of some operations can be optimized. For example, in the secret sharing setting, we can skip truncation after each multiplication in step 7 or 11 and instead truncate the sum after adding all products. In addition, we can restrict variables to shorter bitlength representations when the range of values they store is known to be small. This is relevant to multiplications in step 2 in the garbled circuit setting, where $t_{i,j}$ and $s_{i,j}$ are known not to exceed $N+1$ and can be represented using a short bitlength for the integer part.

6.7 Performance Evaluation

In this section we evaluate performance of the spectral minutia representation fingerprint recognition protocol SpectralFR to demonstrate that the proposed protocols are practical. We provide experimental results for both garbled circuit and secret sharing settings. In the garbled circuit setting, our implementation uses the JustGarble library [25, 26] for circuit garbling and garbled circuit evaluation which we modified to support a half-gates optimization [116]. In the secret sharing setting, we use PICCO [117] with three computational parties, which utilized linear threshold secret sharing.

In our implementation, we assume that the inputs T and S are represented with 32-bit precision after the radix point. Our garbled circuit implementation maintains 56-bit fixed-point representation (24 and 32 bits before and after the radix point), while in the secret sharing setting we let the numbers grow beyond the 56 bits to avoid the cost of truncation, but perform truncation prior to returning the result. We ran each experiment 10 times and report the mean value.

The results of SpectralFR performance evaluation can be found in Table 6.3 for the three-party case and in Table 6.4 for the garbled circuit case. We report performance for a range of parameters N' and M' that might be used in practice. All numbers correspond to the overall runtime including communication. Because in the garbled circuit setting the overall runtime is composed of multiple components, we provide a breakdown of the total time according to its constituents. We have that garbling uses 26–27% of the overall time, garbled circuit evaluation takes 15–16%, oblivious transfer takes about 1% of the time, and communication time is about 57%. The number of non-XOR gates in the circuits was about 27.6%. These numbers tell us that communication takes most of the time (even with the half-gates optimization that reduces communication). Because circuit garbling can be performed offline, the work associated with circuit garbling (a quarter of the overall time) and garbled circuit transmission can be done in advance saving most of the total time.

TABLE 6.3

EXECUTION TIME OF PROTOCOL SpectralFR IN SECONDS USING THE SECRET SHARING

M'	N'				
	66	70	74	78	82
28	0.250	0.268	0.269	0.279	0.285
30	0.266	0.276	0.276	0.287	0.295
32	0.283	0.283	0.307	0.308	0.309

As one can see, secure fingerprint recognition takes a fraction of a second in the secret sharing setting and tens of seconds in the garbled circuit setting. This shows that the protocol is efficient for practical use specially in secret sharing setting.

The computation used in SpectralFR is a rare example of functionality that can be implemented significantly more efficiently using secret sharing techniques than garbled circuit evaluation.

TABLE 6.4

EXECUTION TIME OF PROTOCOL SpectralFR IN SECONDS AND THE TOTAL NUMBER OF GATES (IN MILLIONS) IN ITS IMPLEMENTATION USING THE GARBLED CIRCUIT

M'	N'									
	66		70		74		78		82	
	Time	Gates	Time	Gates	Time	Gates	Time	Gates	Time	Gates
28	60.0	454.2M	63.6	481.8M	67.3	509.4M	70.9	537.0M	74.5	564.5M
30	64.3	486.7M	68.1	516.2M	72.0	545.8M	75.9	575.3	79.8	604.8M
32	69.1	519.1M	72.7	550.6M	76.8	582.1M	81.0	613.6	85.1	645.1M

CHAPTER 7

CONCLUSIONS AND FUTURE DIRECTIONS

In this chapter, we conclude our main contributions in this thesis in Section 7.1, and then we focus on our future directions in Section 7.2.

7.1 Conclusions

In this thesis, we build general and time-efficient privacy-preserving solutions and protocols that can be used for different applications including voice recognition, DNA computation, and fingerprint recognition where privacy of the data is one of the main concerns. In the following, we conclude our main contributions for each secure biometric modality.

7.1.1 Voice Recognition

We treat the problem of privacy-preserving Hidden Markov Models computation which is commonly used for many applications including speaker recognition. We develop provably secure techniques for HMM's Viterbi and GMM computation using floating-point arithmetic in both two-party setting using homomorphic encryption and multi-party setting using secret sharing. These settings correspond to a variety of real-life situations and the solutions were designed to minimize their overhead. A significant part of this work is dedicated to new secure protocols for floating-point operations in the malicious model in the two-party setting based on homomorphic encryption. To the best of our knowledge, this is the first time such protocols are offered in the literature. We rigorously prove security of our protocols using simulation-based proofs, which constitutes a distinct contribution of this work.

7.1.2 DNA Computation

We study server-aided secure garbled circuit computation in a number of security settings. One of such security settings assumes that users A and B may act arbitrarily and, in addition to requiring security in the presence of malicious users, we also enforce that A and B enter their true inputs based on third party certification. We are not aware of any prior work that combines input certification with general secure multi-party computation based on Yao's garbled circuits. We develop general solutions in our server-aided framework. Despite their generality, they lead to efficient implementations of genetic tests. In particular, we design and implement genetic paternity, compatibility, and common ancestry tests, all of which run in a matter of seconds or less and favorably compare with the state of the art.

7.1.3 Fingerprint Recognition

We design three secure and efficient protocols for fingerprint alignment and matching for garbled circuit and secret sharing settings. They are based on popular algorithms in the computer vision literature and employ new non-trivial techniques. The constructions are presented in the semi-honest setting and known results can be applied to strengthen the security to sustain malicious actors. We believe that this is the first work that treats the problem of secure fingerprint alignment. We also design secure constructions for fundamental numeric and set operations. We present novel secure protocols for sine, cosine, arctangent, and square root for fixed-point numbers, as well as a novel secure and data-oblivious protocol for selection of the fth smallest element of a set (for any type of data). The techniques are applicable to both garbled circuit and secret sharing settings.

7.2 Future Plan

It is our goal to continue expanding upon our existing research as well as to incorporate and design new techniques enhancing the research areas of of security, privacy, and applied cryptography. This plan also includes exploring other research directions within information security and how methods within this current field could benefit biometric challenges. In particular, we are interested in working on the following topics:

7.2.1 Efficient Input Certification Protocols

In the current literature, the strongest security settings are defined in the presence of malicious parties, and usually, the correctness of the entered inputs is not considered in the computations. As a result, a malicious party can modify her inputs to learn more information about the other party's inputs. Therefore, there is a high-priority need for a stronger security model to guarantee that all parties provide authorized information in the computations. For this purpose, we plan to provide efficient certified input solutions for different computational settings that could benefit a variety of applications. In [118], you can find our progress in designing a certified input solution based on garbled circuit evaluation.

7.2.2 Secure Protocols in the Presence of a Covert Adversary

There is another adversarial model known as "covert adversarial model". Covert adversaries may deviate arbitrarily from the protocol specification in an attempt to cheat, but they do not wish to be caught. This unique property of the covert user makes it more practical in real-world situations because it may be the case that the risk of being caught is evaluated against the benefits of cheating. Therefore, we are interested in building efficient and secure protocols in this adversarial model and in applying them to different applications including secure biometric computations.

These protocols will be secure enough, and well-suited to the downstream applications. In addition, these protocols designed under the covert adversarial assumption will computationally outperform those under the malicious adversarial assumption.

7.2.3 Data Mining Computations on Large-Scale Data Sets

Classification, clustering, and frequent pattern mining are common practices for knowledge discovery on large-scale data sets. In many cases the data is not public and it is sensitive; therefore, protection of the data is a major requirement. In data mining on large-scale data sets, utility and privacy of data are two main goals for which accomplishing one may lead to an undesired impact on the other. Nowadays, there is a big challenge and urgent need to achieve both practicality on downstream applications and privacy preservation of sensitive data in data mining and knowledge discovery solutions. As a future direction, we plan to design a set of practical in terms of efficiency and general-form secure solutions to carry out classification, clustering, or frequent pattern mining algorithms to address the research challenge in this area.

BIBLIOGRAPHY

1. 23andMe – Genetic Testing for Ancestry; DNA Test. http://www.23andme.com.

2. Genealogy, Family Trees & Family History Records at Ancestry.com. http://www.ancestry.com.

3. GenePartner.com – DNA matching: Love is no coincidence. http://www.genepartner.com.

4. GMP – The GNU multiple precision arithmetic library. http://www.gmplib.org.

5. NIST special database 4. https://www.nist.gov/srd/nist-special-database-4.

6. OpenSSL: The open source toolkit for SSL/TLS. http://www.openssl.org.

7. Fingerprint minutiae viewer (FpMV). https://www.nist.gov/services-resources/software/fingerprint-minutiae-viewer-fpmv.

8. Secure supply chain management (SecureSCM) project deliverable: D9.2 security analysis, July 2009.

9. M. Aliasgari. *Secure computation and outsourcing of biometric data*. University of Notre Dame, 2013.

10. M. Aliasgari and M. Blanton. Secure computation of hidden markov models. In *International Conference on Security and Cryptography*, 2013.

11. M. Aliasgari, M. Blanton, Y. Zhang, and A. Steele. Secure computation on floating point numbers. In *Network and Distributed Security Symposium*, 2013.

12. M. Aliasgari, M. Blanton, and F. Bayatbabolghani. Secure computation of hidden markov models and secure floating-point arithmetic in the malicious model. *International Journal of Information Security*, to appear.

13. G. Asharov, Y. Lindell, and T. Rabin. Perfectly-secure multiplication for any $t < n/3$. In *Cryptology Conference (CRYPTO)*, pages 240–258, 2011.

14. G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *ACM Conference on Computer and Communications Security*, pages 535–548, 2013.

15. M. Atallah, M. Bykova, J. Li, K. Frikken, and M. Topkara. Private collaborative forecasting and benchmarking. In *ACM Workshop on Privacy in the Electronic Society*, pages 103–114, 2004.

16. E. Ayday, J. L. Raisaro, and J.-P. Hubaux. Privacy-enhancing technology for medical tests using genomic data. Technical Report EPFL-REPORT-182897, EPFL, 2012.

17. E. Ayday, J. L. Raisaro, and J. Hubaux. Personal use of genomic data: Privacy vs. storage cost. In *IEEE Global Communications Conference*, pages 2723–2729, 2013.

18. E. Ayday, J. L. Raisaro, J.-P. Hubaux, and J. Rougemont. Protecting and evaluating genomic privacy in medical tests and personalized medicine. In *ACM Workshop on Privacy in the Electronic Society*, pages 95–106, 2013.

19. E. Ayday, J. L. Raisaro, P. McLaren, J. Fellay, and J.-P. Hubaux. Privacy-preserving computation of disease risk by using genomic, clinical, and environmental data. In *USENIX Workshop on Health Information Technologies*, 2013.

20. P. Baldi, R. Baronio, E. De Cristofaro, P. Gasti, and G. Tsudik. Countering GATTACA: Efficient and secure testing of fully-sequenced human genomes. In *ACM Conference on Computer and Communications Security*, pages 691–702, 2011.

21. P. Bansal, A. Kant, S. Kumar, A. Sharda, and S. Gupta. Improved hybrid model of hmm/gmm for speech recognition. In *Book 5 Intelligent Technologies and Applications*. Institute of Information Theories and Applications FOI ITHEA, 2008.

22. M. Barni, T. Bianchi, D. Catalano, M. Di Raimondo, R. Labati, P. Failla, D. Fiore, R. Lazzeretti, V. Piuri, F. Scotti, and A. Piva. Privacy-preserving FingerCode authentication. In *ACM Workshop on Multimedia and Security*, pages 231–240, 2010.

23. K. Batcher. Sorting networks and their applications. In *AFIPS Spring Joint Computer Conference*, pages 307–314, 1968.

24. F. Bayatbabolghani, M. Blanton, M. Aliasgari, and M. Goodrich. Secure fingerprint alignment and matching protocols. *arXiv Report 1702.03379*, 2017.

25. M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway. The JustGarble library. http://cseweb.ucsd.edu/groups/justgarble/.

26. M. Bellare, V. Hoang, S. Keelveedhi, and P. Rogaway. Efficient garbling from a fixed-key blockcipher. In *IEEE Symposium of Security and Privacy*, pages 478–492, 2013.

27. P. Besl and N. McKay. Method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

28. M. Blanton. Empirical evaluation of secure two-party computation models. Technical Report TR 2005-58, CERIAS, Purdue University, 2005.

29. M. Blanton. Achieving full security in privacy-preserving data mining. In *IEEE International Conference on Information Privacy, Security, Risk and Trust*, pages 925–934, 2011.

30. M. Blanton and E. Aguiar. Private and oblivious set and multiset operations. *International Journal of Information Security*, 15(5):493–518, Oct. 2016.

31. M. Blanton and F. Bayatbabolghani. Efficient server-aided secure two-party function evaluation with applications to genomic computation. *Proceedings on Privacy Enhancing Technologies*, 4:144–164, 2016.

32. M. Blanton and F. Bayatbabolghani. An approach to improving security and efficiency of private genomic computation using server aid. *IEEE Security and Privacy Magazine*, 2017.

33. M. Blanton and P. Gasti. Secure and efficient protocols for iris and fingerprint identification. In *European Symposium on Research in Computer Security*, pages 190–209, 2011.

34. M. Blanton and P. Gasti. Secure and efficient iris and fingerprint identification. In D. Ngo, A. Teoh, and J. Hu, editors, *Biometric Security*, chapter 9. Cambridge Scholars Publishing, 2015.

35. M. Blanton and S. Saraph. Oblivious maximum bipartite matching size algorithm with applications to secure fingerprint identification. In *European Symposium on Research in Computer Security*, pages 384–406, 2015.

36. F. Bruekers, S. Katzenbeisser, K. Kursawe, and P. Tuyls. Privacy-preserving matching of DNA profiles. IACR Cryptology ePrint Archive Report 2008/203, 2008.

37. J. Camenisch and A. Lysyanskaya. A signature scheme with efficient protocols. In *Conference on Security and Cryptography for Networks*, pages 268–289, 2002.

38. J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *Cryptology Conference (CRYPTO)*, pages 56–72, 2004.

39. J. Camenisch and M. Michels. Separability and efficiency for generic group signature schemes. In *Cryptology Conference (CRYPTO)*, 1999.

40. J. Camenisch and M. Stadler. Efficient group signature schemes for large groups. In *Cryptology Conference (CRYPTO)*, 1997.

41. J. Camenisch and M. Stadler. Proof systems for general statements about discrete logarithms. Technical report, Institute for Theoretical Computer Science, ETH Zurich, 1997.

42. J. Camenisch and G. Zaverucha. Private intersection of certified sets. In *Financial Cryptography and Data Security*, pages 108–127, 2009.

43. J. Camenisch, D. Sommer, and R. Zimmermann. A general certification framework with applications to privacy-enhancing certificate infrastructures. In *Security and Privacy in Dynamic Environments*, pages 25–37, 2006.

44. R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

45. O. Catrina and S. de Hoogh. Improved primitives for secure multiparty integer computation. In *Security and Cryptography for Networks*, pages 182–199, 2010.

46. O. Catrina and A. Saxena. Secure computation with fixed-point numbers. In *Financial Cryptography and Data Security*, pages 35–50, 2010.

47. CertiVox. Multiprecision integer and rational arithmetic cryptographic library (MIRACL). http://www.certivox.com/miracl/.

48. D. Chetverikov, D. Svirko, D. Stepanov, and P. Krsek. The trimmed iterative closest point algorithm. In *International Conference on Pattern Recognition*, pages 545–548, 2002.

49. R. Cleve. Limits on the security of coin flips when half the processors are faulty. In *ACM Symposium on Theory of Computing*, pages 573–588, 1986.

50. R. Cramer, I. Damgård, and J. Nielsen. Multiparty computation from threshold homomorphic encryption. In *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 280–289, 2001.

51. I. Damgåard and E. Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, pages 125–142, 2002.

52. I. Damgård and M. Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In *International Workshop on Practice and Theory in Public Key Cryptography*, pages 119–136, 2001.

53. I. Damgård and J. Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In *Cryptology Conference (CRYPTO)*, pages 247–264, 2003.

54. I. Damgård, Y. Ishai, and M. Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 445–465, 2010.

55. E. De Cristofaro and G. Tsudik. Practical private set intersection protocols with linear complexity. In *Financial Cryptography and Data Security*, pages 143–159, 2010.

56. E. De Cristofaro, S. Faber, P. Gasti, and G. Tsudik. GenoDroid: Are privacy-preserving genomic tests ready for prime time? In *ACM Workshop on Privacy in the Electronic Society*, pages 97–107, 2012.

57. E. De Cristofaro, S. Faber, and G. Tsudik. Secure genomic testing with size- and position-hiding private substring matching. In *ACM Workshop on Privacy in the Electronic Society*, pages 107–118, 2013.

58. U. Feige, J. Kilian, and M. Naor. A minimal model for secure computation. In *ACM Symposium on Theory of Computing*, pages 554–563, 1994.

59. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature scheme. In *Cryptology Conference (CRYPTO)*, pages 186–194, 1986.

60. M. Franz. *Secure Computations on Non-Integer Values*. PhD thesis, TU Darmstadt, 2011.

61. M. Franz, B. Deiseroth, K. Hamacher, S. Jha, S. Katzenbeisser, and H. Schröder. Towards secure bioinformatics services (short paper). In *Financial Cryptography and Data Security*, pages 276–283. Springer, 2012.

62. E. Fujisaki and T. Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *Cryptology Conference (CRYPTO)*, pages 16–30, 1997.

63. R. Gamboa and J. Cowles. Formal verification of medina's sequence of polynomials for approximating arctangent. *arXiv preprint arXiv:1406.1561*, 2014.

64. R. Gennaro, M. Rabin, and T. Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In *ACM Symposium on Principles of Distributed Computing*, pages 101–111, 1998.

65. O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, 2004.

66. M. Golestanian, H. Nikoo, and J. Sadri. Intron profile analysis approach for exon detection in dna sequences using spectral analysis. *Journal of Bioinformatics and Intelligent Control*, 3(2):140–142, 2014.

67. M. Goodrich. Data-oblivious external-memory algorithms for the compaction, selection, and sorting of outsourced data. In *ACM Symposium on Parallelism in Algorithms and Architectures*, pages 379–388, 2011.

68. J. Hart, E. Cheney, C. Lawson, H. Maehly, C. Mesztenyi, J. Rice, H. Thacher, and C. Witzgall. *Computer approximations*. John Wiley & Sons, Inc., 1968.

69. D. He, N. Furlotte, F. Hormozdiari, J. Joo, A. Wadia, R. Ostrovsky, A. Sahai, and E. Eskin. Identifying genetic relatives without compromising privacy. *Genome Research*, 24:664–672, 2014.

70. W. Henecka, K. Stefan, A. R. Sadeghi, T. Schneider, and I. Wehrenberg. Tasty: Tool for automating secure two-party computations. In *ACM Conference on Computer and Communications Security*, pages 451–462. ACM, 2010.

71. A. Herzberg and H. Shulman. Oblivious and fair server-aided two-party computation. In *International Conference on Availability, Reliability and Security*, pages 75–84, 2012.

72. F. Hormozdiari, J. Joo, A. Wadia, F. Guan, R. Ostrovsky, A. Sahai, and E. Eskin. Privacy preserving protocol for detecting genetic relatives using rare variants. *Bioinformatics*, pages 204–2011, 2014.

73. B. Horn. Closed-form solution of absute orientation using unit quaternions. *Journal of the Optical Society of America A*, 4(4):629–642, 1987.

74. Y. Huang, L. Malka, D. Evans, and J. Katz. Efficient privacy-preserving biometric identification. In *Network and Distributed System Security Symposium*, 2011.

75. Y. Huang, J. Katz, and D. Evans. Quid-pro-quo-tocols: Strengthening semi-honest protocols with dual execution. In *IEEE Symposium of Security and Privacy*, 2012.

76. Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *Cryptology Conference (CRYPTO)*, pages 145–161, 2003.

77. S. Jarecki and V. Shmatikov. Efficient two-party secure computation on committed inputs. In *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 97–114, 2007.

78. S. Kamara, P. Mohassel, and M. Raykova. Outsourcing multi-party computation. IACR Cryptology ePrint Archive Report 2011/272, 2011.

79. S. Kamara, P. Mohassel, and B. Riva. Salus: A system for server-aided secure function evaluation. In *ACM Conference on Computer and Communications Security*, pages 797–808, 2012.

80. A. Karatsuba and Y. Ofman. Multiplication of many-digital numbers by automatic computers. *Doklady Akademii Nauk SSSR*, 145:293–294, 1962.

81. J. Katz and L. Malka. Secure text processing with applications to private DNA matching. In *ACM Conference on Computer and Communications Security*, pages 485–492, 2010.

82. F. Kerschbaum, M. Atallah, D. M'Raïhi, and J. Rice. Private fingerprint verification without local storage. In *International Conference on Biometric Authentication*, pages 387–394, 2004.

83. M. Kiraz, T. Schoenmakers, and J. Villegas. Efficient committed oblivious transfer of bit strings. In *Information Security Conference*, pages 130–144, 2007.

84. V. Kolesnikov and T. Schneider. Improved garbled circuit: Free XOR gates and applications. In *International Colloquium on Automata, Languages and Programming*, pages 486–498, 2008.

85. V. Kolesnikov, A. R. Sadeghi, and T. Schneider. Improved garbled circuit building blocks and applications to auctions and computing minima. In *Cryptology and Network Security*, pages 1–20. Springer, 2009.

86. V. Kolesnikov, R. Kumaresan, and A. Shikfa. Efficient verification of input consistency in server-assisted secure function evaluation. In *Cryptology and Network Security*, pages 201–217, 2012.

87. B. Kreuter, A. Shelat, and C. Shen. Billion-gate secure computation with malicious adversaries. In *USENIX Security Symposium*, 2012.

88. M. Liedel. Secure distributed computation of the square root and applications. In *Information Security Practice and Experience*, pages 277–288. Springer, 2012.

89. Y. Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In *Cryptology Conference (CRYPTO)*, 2013.

90. Y. Lindell and B. Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, 2009.

91. Y. Lindell and B. Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. *Journal of Cryptology*, 25(4):680–722, 2012.

92. D. Maltoni, D. Maio, A. K. Jain, and S. Prabhakar. *Handbook of Fingerprint Recognition*. New York Springer-Verlag, 2003.

93. T. Matsui and S. Furui. Speaker adaptation of tied-mixture-based phoneme models for text-prompted speaker recognition. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages 125–128, 1994.

94. H. A. Medina. A sequence of polynomials for approximating arctangent. *The American Mathematical Monthly*, 113(2):156–161, 2006.

95. P. Mohassel and M. Franklin. Efficiency tradeoffs for malicious two-party computation. In *Public Key Cryptography*, pages 458–73, 2006.

96. P. Mohassel and B. Riva. Garbled circuits checking garbled circuits: More efficient and secure two-party computation. In *Cryptology Conference (CRYPTO)*, pages 36–53, 2013.

97. P. Mohassel, M. Rosulek, and Y. Zhang. Fast and secure three-party computation: The garbled circuit approach. In *ACM Conference on Computer and Communications Security*, pages 591–602, 2015.

98. K. Nandakumar, A. K. Jain, and S. Pankanti. Fingerprint-based fuzzy vault: Implementation and performance. *IEEE Transactions on Information Forensics and Security*, 2(4):744–757, 2007.

99. M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 448–457, 2001.

100. H. Nguyen and M. Roughan. Multi-observer privacy-preserving hidden markov models. In *Network Operations and Management Symposium*, pages 514–517, 2012.

101. H. Nguyen and M. Roughan. On the identifiability of multi-observer hidden markov models. In *International Conference on Acoustics, Speech and Signal Processing*, pages 1873–1876, 2012.

102. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 223–238, 1999.

103. M. Pathak and B. Raj. Privacy preserving speaker verification using adapted GMMs. In *Interspeech*, pages 2405–2408, 2011.

104. M. Pathak, S. Rane, W. Sun, and B. Raj. Privacy preserving probabilistic inference with hidden Markov models. In *International Conference on Acoustics, Speech and Signal Processing*, pages 5868–5871, 2011.

105. M. Pathak, J. Portelo, B. Raj, and I. Trancoso. Privacy-preserving speaker authentication. *Information Security Conference (ISC)*, pages 1–22, 2012.

106. M. Pathak, B. Raj, S. Rane, and P. Saragdis. Privacy-preserving speech processing: cryptographic and string-matching frameworks show promise. *IEEE Signal Processing Magazine*, 30(2):62–74, 2013.

107. H. Polat, W. Du, S. Renckes, and Y. Oysal. Private predictions on hidden Markov models. *Artificial Intelligence Review*, 34(1):53–72, 2010.

108. S. Shahandashti, R. Safavi-Naini, and P. Ogunbona. Private fingerprint matching. In *Australasian Conference on Information Security and Privacy*, pages 426–433, 2012.

109. A. Shahbazi, F. Bayatbabolghani, and M. Blanton. Private computation with genomic data for genome-wide association and linkage studies. In *International Workshop on Genomic Privacy and Security*, 2016.

110. A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

111. M. Shashanka. A privacy preserving framework for Gaussian mixture models. In *IEEE International Conference on Data Mining Workshops*, pages 499–506. IEEE, 2010.

112. A. Shelat and C. h. Shen. Two-output secure computation with malicious adversaries. In *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2011.

113. P. Smaragdis and M. Shashanka. A framework for secure speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 15(4):1404–1413, 2007.

114. H. Xu, R. Veldhuis, A. Bazen, T. Kevenaar, T. Akkermans, and B. Gokberk. Fingerprint verification using spectral minutiae representations. *IEEE Transactions on Information Forensics and Security*, 4(3):397–409, 2009.

115. H. Xu, R. Veldhuis, T. Kevenaar, and T. Akkermans. A fast minutiae-based fingerprint recognition system. *IEEE Systems Journal*, 3(4):418–427, 2009.

116. S. Zahur, M. Rosulek, and D. Evans. Two halves make a whole: Reducing data transfer in garbled circuits using half gates. In *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 220–250, 2015.

117. Y. Zhang, A. Steele, and M. Blanton. Picco: A general-purpose compiler for private distributed computation. In *ACM Conference on Computer and Communications Security*, pages 813–826, 2013.

118. Y. Zhang, M. Blanton, and F. Bayatbabolghani. Enforcing input correctness via certification in garbled circuit evaluation. Cryptology ePrint Archive Report 2017/569, 2017.

This document was prepared & typeset with pdfLaTeX*, and formatted with* NDdiss2$_\varepsilon$ *classfile (v3.2013[2013/04/16]) provided by Sameer Vijay and updated by Megan Patnott.*