EXPLORING PRIVACY IN LOCATION-BASED SERVICES USING CRYPTOGRAPHIC PROTOCOLS

Roopa Vishwanathan

Dissertation Prepared for the Degree of

DOCTOR OF PHILOSOPHY

UNIVERSITY OF NORTH TEXAS

May 2011

Vishwanathan, Roopa. Exploring Privacy in Location-based Services Using Cryptographic Protocols. Doctor of Philosophy (Computer Science and Engineering), May 2011, 131 pp., 8 tables, 20 figures, references, 87 titles.

Location-based services (LBS) are available on a variety of mobile platforms like cell phones, PDA's, etc. and an increasing number of users subscribe to and use these services. Two of the popular models of information flow in LBS are the client-server model and the peer-to-peer model, in both of which, existing approaches do not always provide privacy for all parties concerned. In this work, I study the feasibility of applying cryptographic protocols to design privacy-preserving solutions for LBS from an experimental and theoretical standpoint. In the client-server model, I construct a two-phase framework for processing nearest neighbor queries using combinations of cryptographic protocols such as oblivious transfer and private information retrieval. In the peer-to-peer model, I present privacy preserving solutions for processing group nearest neighbor queries in the semi-honest and dishonest adversarial models. I apply concepts from secure multi-party computation to realize our constructions and also leverage the capabilities of trusted computing technology, specifically TPM chips. My solution for the dishonest adversarial model is also of independent cryptographic interest.

I prove my constructions secure under standard cryptographic assumptions and design experiments for testing the feasibility or practicability of our constructions and benchmark key operations. My experiments show that the proposed constructions are practical to implement and have reasonable costs, while providing strong privacy assurances.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1. Overview

Location based services (LBS) are becoming increasingly common with location-enabled client user devices like mobile phones, or PDAs making queries to location servers. In such applications, clients may not want to inadvertently reveal their own identity and location while querying the server for nearby places of interest, and the server on its part would not want to reveal more data than what is necessary to the client. Privacy issues aren't limited to the client-server model of LBS and might extend to the peer-to-peer model where a group of mutually untrusting clients jointly compute their group nearest neighbor without the help of a server or third party. Solutions proposed for addressing privacy issues in LBS up until now do not provide complete privacy for all the parties involved in either the client-server model or the peer-to-peer model. One needs to be careful when designing solutions for these problems since ad-hoc or heuristic solutions may lead to one or more parties unintentionally revealing private data. Hence, one needs solutions that are amenable to rigorous analysis of their security properties, and also which are practical and not too expensive to implement in practice. An ideal way to accomplish this goal would be use cryptographic protocols. Another point that may be a matter of concern is that, typically, in applications such as LBS, the software running on the parties' machines (server and users) is assumed to be trusted to perform secure computations. Software can easily be modified and all parties need to be assured that the computations performed on any particular device are trustworthy. Hence trust cannot depend on software alone, trusted hardware must also be part of the process of trust establishment. The need to design privacy-preserving solutions for LBS that meet the twin goals of being efficient and feasible to implement while being amenable to formal analysis in a framework that is based on trusted software and hardware was the main motivating factor that led to the development of this dissertation.

In this dissertation I explore the application of cryptographic protocols to LBS and study this problem from a theoretical and experimental perspective. I consider three situations that have privacy implications in LBS and, firstly, define security properties of functionalities that addresses these privacy concerns, then construct solutions that meet my defined security properties (realize the functionalities), and provide proofs of security for my solutions. I next experimentally evaluate my solutions with respect to computation and communication costs and benchmark key operations.

1.2. Problems Considered in the Dissertation

There are three main problems considered in this dissertation: nearest neighbor queries in the single-user, client-server LBS model and their associated privacy problems, privacy issues in the multi-user, peer-to-peer computation of group nearest neighbor queries in the **semi-honest** model, and addressing privacy problems in the multi-user, peer-to-peer computation of group nearest neighbor queries in the **dishonest** model using trusted computing technology.

1.2.1. Single-User Model

In the single-user, client-server model, client devices make queries to a location-based server about nearby locations of interest to the client such as restaurants, gas-stations, etc.

One way for the client to hide its own location is to query a large region and filter the query results to find its exact location within that region. As a result the server needs to respond to the clients' queries with more data than what is needed; however, the server may not want to reveal the whole LBS database to the client. Hence, I need to construct a solution that balances both parties' interests - protecting the privacy of the client and reducing the amount of data supplied by the server. Most solutions proposed for this problem till now take into account the privacy of the client, but one also needs to ensure that while protecting the privacy and anonymity of the client, one does not inadvertently release more information than what is required about other data stored in the server. Typically a location server might contain thousands of points of interest (POIs) listed in a particular category (e.g., list of restaurants or list of gas-stations in a given area) organized into a table, and if a user queries this table, the server would not want to return every item in its listing. In addition to being redundant, this would also enable a user to engage the server's resources for long periods of time. Hence one needs to ensure that the data returned is fine-grained and precise (very close to the user's location). This would also save the user time, since they do not have to sort through reams of redundant data. That is, if the server maintains a number of POIs with each type organized into a table, it may allow clients to query part of the tables about a limited region around a specific location, but not access all (or large amounts of) the data stored in it. For example, the server may allow queries like: "How many people (or users) are within 1 mile of a given location," or "Where is the nearest gas station from my current location?" But the server may not want the client to ask queries along the lines of "Give me a complete list of restaurants in the state of Texas."

To paraphrase the problem I am trying to solve: In location-based services, I need to find an efficient way to protect the privacy and anonymity of the client, while ensuring that the server answers location-related queries in as precise a manner as possible, keeping the redundant data to a minimum, and has some control over what data is revealed. Specifically, if a user u has a location l and needs to query a LBS to obtain its nearest neighbor from a set of POIs S where $S = \{s_1, s_2, \ldots, s_m\}$, the goal is to develop an efficient protocol that does not allow the LBS to learn l while reducing the subset of S to be sent to u in order to answer the query.

1.2.2. Multi-User, Semi-Honest Model

An alternative communication model is the peer-to-peer model where a group of peers would like to co-operatively compute some location without seeking the help of a centralized location-based server. Apart from the obvious shortcomings of the centralized approach such as the server being untrusted, or the server being a communication bottleneck and a single point of failure, the power of user hand-held devices has been increasing with technologies like Bluetooth or IEEE 802.11 being deployed on a range of user hand-held devices. With the computational power of user devices increasing, users may not even need a location-based server or a base station to answer queries and could process queries among themselves. Although there can be many types of spatial queries that a group of peers can compute, in this dissertation I focus on **group nearest neighbor** queries in which a group of parties, $P_1, \cdots, P_n$ would want to compute their joint group nearest neighbor. In a real-world situation, this could be just a group of users wanting to compute a location that is closest to all of them, e.g., “Which is the restaurant that is closest to all of us where we can meet?” The problem I am focusing on is the situation when the parties want to jointly compute this function but do not want to reveal their individual locations to each other, and work in the absence of a trusted third party. Geometrically, the group nearest neighbor of the group of parties is the minimum of the sum of the distances of each party from each point in a given set of points in the problem space. If there are $P_1, P_2, \cdots, P_n$ parties and $L_1, L_2, \cdots, L_l$ locations, for computing the group nearest neighbor, I firstly compute the sum of the distances of each party from each location: $s_1 = \sum_{i=1}^{n} d(P_i, L_1), s_2 = \sum_{i=1}^{n} d(P_i, L_2), \cdots, s_l = \sum_{i=1}^{n} d(P_i, L_l)$. I then obtain the minimum of all these values: $min(s_1, s_2, \cdots, s_l)$.

1.2.3. Multi-User, Dishonest Model

I next consider the **dishonest** model of peer-to-peer computation where none of the parties trust each other unlike the model in Part 2, where all the parties are considered to be **semi-honest**. There have been many solutions proposed for the dishonest or malicious adversarial model of SFE; some representative work includes [59, 33, 4, 52, 45, 75, 84, 64, 47]. Almost all of the solutions for the multi-party malicious model are based on either Goldreich's compiler [28], or are based on cut-and-choose zero knowledge proofs. The compiler of Goldreich uses generic zero knowledge proofs (ZKPs) which are mainly of theoretical importance and are too inefficient to apply in practice. Cut-and-choose is a popular paradigm used in interactive ZKPs and while it isn't as expensive as generic zero knowledge proofs, is still a highly interactive construct and significantly increases the communication costs of the protocol using it. I explore ways in which one can use modified trusted platform modules (TPMs) in the malicious model to design solutions for the SFE problem which are more efficient than previously known cut-and-choose ZKP-based solutions.

The Trusted Computing Group, an industry consortium of over 100 companies, has developed specifications for a hardware chip called the trusted platform module (TPM) [35] which can help answer basic security questions like “Were the keys truly randomly generated?” or “Are the garbled inputs correctly created?” or more relevant to my problem, “Was the ‘right’ function transformed into a garbled circuit?” TPMs have become common in laptops, business-oriented desktops, and tablet PCs, including those by IBM, Dell, Lenovo, HP, and Toshiba, as well as in some server-class systems such as the Dell R300. TPMs are designed to be very cheap (under $5 each), and are intended to be easily embedded on the motherboard of a standard system.

The most widely used solution paradigm for SFE is **garbled circuits**; other solutions involve **homomorphic encryption**, where the encryption function is an additive or multiplicative homomorphic function. In this dissertation, I investigate ways in which one can use limited pieces of trusted hardware such as a TPM chip to add authentication to garbled circuit-based solutions for SFE in an efficient way, and briefly point out how one can construct homomorphic encryption-based solutions using TPMs, although I leave the implementation of homomorphic encryption-based solutions as part of future work. I then show how the TPM-assisted SFE protocols can be used in LBS.

1.3. Solution Approaches Overview

I consider the three problems outlined in Section 1.2 and explore the application of cryptographic protocols to them. In the single-user client-server model, I use combinations of private information retrieval (PIR) and oblivious transfer (OT), in the multi-user semi-honest model; I model the problem in the framework of secure function evaluation (SFE) and present solutions in two models using garbled circuits: the centralized and distributed models. In the third part, I again model the problem in the SFE framework with the added condition that none of the parties are honest/semi-honest, and design a garbled circuit-based solution using trusted platform modules (TPMs). It should be noted that my TPM-based solutions are of general cryptographic interest and can be used in **any** application that uses SFE, and not just LBS.

The strategy I adopt to tackle the aforementioned problems is to first provide precise definitions of the security properties that any proposed solution is required to have and then construct protocols that meet the definitions. I then examine the feasibility of applying my protocols in terms of cost, i.e., how efficient they are. For the efficiency analysis, I benchmark the time taken for performing key operations by my solutions in each of the three parts of the dissertation. In the first part, I measure the time taken for performing PIR and OT computations on the server and client's side with varying data sizes and varying key sizes, besides measuring the communication cost of PIR and OT. In the second part, I measure the time taken for constructing and evaluating the garbled circuit in the centralized and distributed models. In the third part, I measure the time taken for performing the TPM-assisted garbled circuit operations. Since these operations cannot be implemented on an actual TPM, I use a simulator with performance profiles of real TPMs to benchmark time. Also, I compare the performance of the TPM-assisted garbled circuit construction with an alternative construction - the Flicker-based garbled circuit construction which is based on a trusted computing infrastructure developed by McCune et al. [62].

1.4. Brief Summary of Results

The main contributions in this dissertation are efficient solutions for the problems outlined in Section 1.2. I use cryptographic protocols and TPM chips to achieve this.

In the single-user, client-server model, I present an efficient solution for the exact nearest neighbor query where along with ensuring the client's privacy, the data returned by the server is as precise as possible. My solution is a general-purpose one and can use either a two-level PIR [14] or it can use a combination of PIR and Oblivious Transfer [76]. I describe a general framework in which one can use three different combinations of PIR and OT: PIR+OT, OT+PIR, PIR+PIR and discuss why a two-level OT approach (OT+OT) wouldn't be very efficient to implement. Additionally, I describe a single-level approach where one can use either PIR over a square grid as described in [25] or 1-of-n OT over the entire database organized as single-dimensional list, or one can use a **random** OT technique which offers a lesser degree of privacy. I analyze the cost of my two-level framework and compare it with the single-level approaches. I also formally define and prove the security properties of my protocols.

In the peer-to-peer, semi-honest model, I propose a framework for preserving user privacy in the computation of group nearest neighbor queries in LBS, which is completely peer-to-peer based, does not require the use of a trusted third party and works in the presence of mutually untrusting peers. I propose two models for the group nearest neighbor query computation: the centralized model and distributed model. I experimentally evaluate the performance in both these models and show that the costs are reasonable and applicable in practice. From my experiments, I find that the distributed model has lower computation costs for processing queries (0-2.1 sec) as compared to the centralized model (0-12 sec), while both models incur the same communication costs.

In the peer-to-peer dishonest model, I show how limited trusted hardware such as trusted platform module chips (TPMs) can be extended to support SFE efficiently in the presence of malicious adversaries and can be used as alternate solutions to expensive cut-and-choose zero knowledge proofs which are currently used for this purpose. Most of the protocols that have been proposed for SFE in the malicious model use expensive cut-and-choose zero-knowledge proofs (ZKPs) that are infeasible to implement in most applications that use SFE. I replace the statistical trust gained from ZKPs with a degree of trust in tamper-evident hardware. In my methodology, I require a few extensions to the current TPM version 1.2 specification for supporting garbled circuits: firstly, I require a keyed HMAC-PRF which is a minor extension since TPMs already support HMACs and this would just be a matter of defining a key type to use as a seed for the HMAC; the second extension is a more significant one where I require a set of new TPM commands: TPM_SFEInit, TPM_SFEInput, TPM_SFEGate, TPM_SFEFinish. I believe that these extensions are worth considering since they consist of operations already supported by the TPM, are simple to implement and in a general sense, a lot of applications would potentially benefit from them (secure function evaluation has uses in a variety of applications ranging from medical diagnostics to data mining). Also, as an alternative, I implement my methodologies on a trusted computing infrastructure called **Flicker** [62] and experimentally compare the performance of a TPM vis-a-vis Flicker.

1.5. Organization

This dissertation is organized as follows: In Chapter 2 I review relevant work in the areas of location-based services, the cryptographic protocols I make use of, and trusted computing. In Chapter 3, I discuss preliminaries about and define various theoretical concepts used throughout this dissertation, e.g., secure function evaluation (SFE) and its two models: the **semi-honest** model and the **dishonest** model. Chapter 4 discusses privacy issues in the single-user, client-server model and my solution. Chapter 5 is about privacy issues in the multi-party, peer-to-peer **semi-honest** model and my solution. Chapter 6 presents my

TPM-assisted solution to the multi-party, peer-to-peer **dishonest** model. Chapter 7 presents experimental results and benchmarks for all my solutions discussed in previous chapters. Finally, Chapter 8 wraps up with conclusions drawn from my work and discussions about future work.

CHAPTER 2

LOCATION-BASED SERVICES, CRYPTOGRAPHIC PROTOCOLS AND TRUSTED COMPUTING

In this chapter I review relevant related work in location-based services (Section 2.1), cryptographic protocols (Section 2.2) and trusted computing (Section 2.3).

2.1. Location-Based Services

Previous work in location-based services can be divided into work done in the single-user, client-server model, on which the bulk of prior work has mainly focused and the peer-to-peer model wherein there has been work done in a general sense, but little, if any, work that takes privacy issues into account.

2.1.1. Single-User, Client-Server Model

In this model, one major line of work is based on cloaking techniques using anonymizers [22, 50, 65]. In this approach, service requests are first transmitted to a third-party trusted server called location anonymizing server (AS). The AS strips off a service requester's identifying information, such as user's network address and name. Various perturbation operations can be performed on the AS to obfuscate the original location information. Location based service (LBS) requests are then issued from the AS to the LBS providers. The AS can also help filter the query results and return the exact answers to the original requesters. One major problem with this approach is that the anonymizer becomes a single point of failure and a performance bottleneck, besides the client and server will have to trust the anonymizer.

Another solution is to organize the network as a peer-to-peer network, instead of having a centralized location server [15, 26]. In this approach, a mobile client first finds a peer group that meets the privacy requirement (i.e. find $K-1$ neighbors) through peer searching. Then the client calculates a region that includes the $K-1$ peers (called K-anonymity [36]). It randomly chooses one of its neighbors as the agent to request *LBS* using the cloaking region. The query results are eventually forwarded to the original client through the agent. Since the disclosed location of a requester is expanded to an area that includes at least $K-1$ other mobile users, any of the K people within the disclosed area could have been the user. For private locations that are likely to release a requester's identification, the cloaking area is generally large if one uses K-anonymity due to the small number of users in private areas. This approach requires the construction of a large peer network of mutually trusting and honest users and cannot guarantee service availability when clients are sparsely distributed.

Another approach is perturbation of the client's location among k other locations. One of the papers that adopts this approach is SpaceTwist [86] where a querying client asks the LBS server to return a set of k points closest to its own location or k-nearest neighbors. The client does this by choosing an "anchor" point in the problem space with respect to which the client requests the server to return k points of interest (POIs). In addition, the client also maintains a running list of k-nearest neighbors based on what the server returns. The server does not know the client's real locations and returns the POIs in ascending order of their distance from the anchor point. When the server returns a point that is closest to the client's real location so far, the client adds that point to its running list. The protocol continues till the entire problem space is covered. This approach provides approximate query results (not exact nearest neighbor) and incurs a very high communication cost of incrementally serving locations queries to the client.

The paper by Ghinita et al. [25] presents a solution based on computational private information retrieval [14]. This approach has several advantages: it does not reveal any spatial information, it is resistant against correlation attacks, and it does not require any trusted third party, among others. The paper first presents a PIR-based framework for location based services and then presents solutions to the problems of finding approximate nearest neighbors and exact nearest neighbors. The approach is expensive in terms of communication and computation; some of the improvements and optimizations the authors have proposed include compressing the data sent by the server, having rectangular grids/matrices in addition to square ones, and using off-line data mining to avoid redundant computation. However the server still reveals large amounts of extra information which results in exposing large portions of the server's data assets to the client. Another paper by Khoshgozaran and Shahabi [54] applies hardware-assisted PIR [43], where a trusted hardware module or chip or secure co-processor, **SC** is placed close to (or inside) an untrusted LBS server **LBS_DB**. The secure co-processor SC is a much smaller processor than the server itself, has limited memory and processing power, and is assumed to be a tamper-evident device. The SC first shuffles the location database, LBS_DB using a private shuffling function π such that $LBS\_DB[i] = LBS\_DB_{pi}[\pi[i]]$ and encrypts the shuffled database which can then be safely stored on the untrusted server. A querying client which wants to retrieve the i^{th} bit from LBS_DB then sends its query, $q = LBS\_DB[i]$ encrypted with the public key of the SC to the server which passes it along to the SC. The SC decrypts q to find i and retrieves

$LBS\_DB_{\pi}[\pi[i]]$ from the untrusted server, decrypts it and again re-encrypts it with the client's public key and sends it back to the client. Through the entire process, the server remains oblivious as to what data the client has retrieved from it. This approach (hardware-assisted PIR) as applied to LBS hasn't been implemented and although in principle, this methodology seems simple enough, it isn't clear as to how simple this would be to implement on current commodity trusted hardware such as a TPM [35] chip or a trusted infrastructure such as Flicker [62].

Besides computational and hardware-assisted PIR, there are a couple of other PIR approaches such as multiple-server PIR [5] where the database in distributed among multiple non-communicating servers and symmetric PIR [24], a variation of which is oblivious transfer (OT) which I use in my framework. In this work, I focus mainly on computational PIR, although it would be interesting to explore other forms of PIR for future work.

Another paper that deals with nearest neighbor queries is the one by Khoshgozaran and Shahabi [53] which uses tamper-proof hardware on both the server and client side. The basic idea is to utilize one-way transformations to map the space of all static and dynamic objects to another space and resolve the query blindly in the transformed space. This approach requires that the transformation used preserves relative proximity of POIs. A trusted computing-based approach is taken by Hengartner in [42] in which the execution of the PIR protocol is done by and responses to queries (locations) are given by a trusted module such as a TPM chip. This methodology has not been implemented yet. While this methodology may be realizable using custom-designed hardware tokens [32], it is not clear whether it can be realized using existing present-day TPMs as the paper suggests. TPMs only support the most basic of cryptographic primitives such as encryption, signature, hash functions and pseudo-random number generation, and lack application-specific support such as for LBS.

2.1.2. Peer-To-Peer Model

In the peer-to-peer model of LBS, there would be a group of clients or peers who want to mutually compute some location-related function in the absence of a centralized trusted third party server. One of the common kind of queries in this model are aggregate nearest neighbor or group nearest neighbor queries where the “nearest neighbors” are points of interest in the vicinity such as restaurants, hospitals, gas stations, etc. So a typical group nearest neighbor query for a group of peers would be “Which is the restaurant that is closest to all of us” or “Find a meeting spot that is within 2 miles of all of us?”

Some of the general work in this direction (non-privacy-preserving) includes [86, 87] among others. There has been quite some work in the area of aggregate nearest neighbor queries [87] which show how to do query processing of aggregate nearest neighbor queries in road networks. Papadias et al. [74] propose a suite of algorithms for nearest neighbor queries, range queries, distance joins and storage scheme for objects situated on a static network.

There has been other work too which discusses spatial transformations of the network and shortest path queries with spatial constraints [79]. Jensen [49] et al. present an algorithm for nearest neighbors in a moving object network. More relevant to my work is the paper by Papadias et al. [73] which was one of the first papers in this area. In their paper, they computed the group nearest neighbor of a group of parties by summing up the Euclidean distance between parties and locations and calculating the minimum of the distances. My work is based on the same function as in [73], but to my knowledge, none of the above referenced papers discuss the privacy aspect of group nearest neighbor queries.

Previous solutions for the group nearest neighbor query computation assume that each party would be willing to share its location with all other parties to compute the group nearest neighbor, or there would be a trusted third party who would be willing to compute the group nearest neighbor, to whom all the parties would reveal their individual locations. Although there has been prior work in the area of group nearest neighbor computation, none of the previous papers have suggested how to provide user privacy if the network is modeled as a peer-to-peer network, in the absence of a trusted third party, and if the peers are untrusted. I note that it is not possible to trivially extend or modify other location-based peer-to-peer protocols for the group nearest neighbor query, such as the ones that use k-anonymity [15, 26], since those protocols assume that peers would be willing to share their locations with each other. It can be seen that this problem fits in the secure multi-party function evaluation framework of cryptography discussed in Section 2.2.

2.2. Cryptographic Protocols

In this section, I provide an overview of various cryptographic protocols used throughout this dissertation. In Chapter 4, I use combinations of private information retrieval (PIR) and oblivious transfer (OT) to design privacy-preserving protocols for LBS. Below, I give a brief description of PIR and OT and formally define them in Chapter 3

2.2.1. Private Information Retrieval (PIR)

Private information retrieval (PIR), first introduced by Chor et al. [14], is a technique or protocol to let a user retrieve information from a database server without the database server knowing the element or record that has been retrieved. In the traditional definition of PIR, the database server has an n-bit string $X = X_1, ..., X_n$, and the client wants to know the value of X_I. The client sends a request to the server in the form of an obfuscatedvector, $E(I) = q$, where E denotes the algorithm used for generating the obfuscated vector. The server responds with a value $v(X, q)$. Using this, the client can compute the value of X_I. Typically one requires that the client's request remain private in the presence of a computationally-bounded adversary, which is referred to as "computational PIR" (as opposed to "information theoretic PIR" in which the adversary is not bounded). Kushilevitz and Ostrovsky's well-known solution [57] to the computational PIR problem is based on the computational intractability of deciding whether a given number is a quadratic residue of a composite modulus. Although there are other forms of PIR such as hardware-assisted PIR, multiple-server PIR and symmetric PIR, in this dissertation, I focus on computational PIR and the protocol from Kushilevitz and Ostrovsky.

2.2.2. Oblivious Transfer (OT)

Oblivious transfer is a cryptographic protocol that was introduced by Brassard, Crépeau and Roberts [8] and is used as a fundamental construct in various cryptographic protocols, including those for multi-party secure function evaluation. Consider a setting where Bob has a string of bits $X_1, ..., X_n$ and Alice wants to know one of them. Alice does not want Bob to know which bit she has chosen, and Bob does not want Alice to know any bit other than the one she has chosen. The solution for this is known as the 1-of-n OT protocol. There are two variants of this protocol: the 1-of-2 OT protocol and the k-of-n OT protocol. In 1-of-2 OT, Bob has 2 elements, X_0 and X_1. Alice chooses to know one of them. Alice receives exactly one element without learning anything about the other element, while Bob remains **oblivious** as to which element was sent. In the k-of-n OT protocol, Bob has a string of bits $X_1, ...X_n$, Alice wants to know a subset of k of these bits. Many efficient protocols have been proposed for the oblivious transfer problem, including a protocol in the standard model due to Naor and Pinkas [76], and more recently a hardware-assisted protocol due to Gunupudi and Tate [40].

2.2.3. Secure Function Evaluation (SFE)

Secure function evaluation (SFE) among two or more parties is a well-studied problem in cryptography in which a group of parties want to jointly compute a function $f(x_1, x_2, \cdots, x_n)$ where $x_1, x_2, \cdots, x_n$ represent their individual input bits. The goal for each party is to correctly compute the value of $f(\cdot)$ without divulging its own input bit to any other party and in the absence of a trusted third party. Pioneering work in this area was done by Yao [85], Goldreich, Micali and Wigderson (GMW) [28] and Beaver, Micali and Rogaway (BMR) [4]. There are two variants of the SFE problem: two-party/multi-party computation in the presence of semi-honest adversaries (the semi-honest model), and two-party/multi-party computation in the presence of malicious adversaries (the dishonest model). In the semi-honest model, it is assumed that all the parties involved in the computation will follow the protocol exactly, but will analyze the transcripts of their interactions with other parties to gain extra, un-intended information, or in other words, will function as **passive** adversaries. In the dishonest model, it is assumed that a fraction of the parties that are "dishonest" will arbitrarily deviate from the protocol and will act as **active** adversaries. For the most part, in the semi-honest model, various solutions developed by different authors are based on either Yao's protocol or the GMW paradigm. This dissertation uses Yao's protocol in the semi-honest model (Chapter 5).

A protocol that is secure in the semi-honest model can be transformed into a protocol that is secure in the dishonest model by applying Goldreich et al.'s compiler [28] or by employing cut-and-choose zero knowledge proofs, with varying degrees of increment in the communication cost of the protocol, and indeed, protocols have been obtained this way by Katz et al. [52], Pass et al. [75] and Lindell [59] among others.

Most solutions for the SFE problem, especially in the semi-honest model are based on Yao's protocol [85] which was the first work in this direction. Yao's protocol has inspired most of the subsequent SFE solutions and is possibly the most widely cited work in this area. In Yao's protocol there are two parties, Alice and Bob holding private inputs x and y and want to compute a function of their inputs, $f(x, y)$ without revealing their individual input bits to each other. In order to do this, Alice first converts the function $f(\cdot)$ to a boolean circuit that consists of many 2-input gates. In the next step Alice creates **signals** or keys for each gate's input wires which are random strings. These random strings can be the length of keys used for any standard symmetric encryption algorithm: 80, 128, or 256 bits. In the next step, Alice generates the **semantics** variables which map the signals to plaintext bits, i.e. 0 and 1 and generates the garbled truth tables or **gate labels** which are just the row-wise entries in the truth table of the function. Next, Alice encrypts the plaintext bits with the signals which would give us the garbled inputs. The semantics variables are the bits that associate the plaintext bits with the encrypted values. This is the reason why the semantics variables need to be kept secret and not revealed to the other parties involved, except the circuit creator. The next step is the stage when the circuit is evaluated by Bob. For doing this, Alice reveals the encrypted gate labels and garbled inputs to Bob who then does an oblivious transfer (OT) [76] with Alice for getting each of his input bits. Using the keys that he learns, Bob finds exactly one of the output-wire signals. In this way, Bob can evaluate the entire garbled circuit. At the end of the protocol, Bob computes the last output bit which would be the answer to the function $f(x, y)$, and sends it to Alice or broadcasts it. If Yao's protocol is extended to multiple parties, then one would have just one party - Alice who creates the circuit, and the circuit creation stage with the signals, semantics and gate labels would be the same, just that all the parties would do an OT with Alice to get their input bits and then send their encrypted bits to Bob for evaluation purposes. Bob would then have the input bit of Alice as well as all the other parties which would be sufficient for him to evaluate the entire circuit by himself.

A different kind of technique for constructing garbled circuits is one by Kolesnikov and Schneider [56] which is what I use in my methodology in Chapter 6 of this dissertation. In this construct, there are two kinds of gates: XOR gates and non-XOR, regular gates. In Kolesnikov and Schneider's construction, they show how to evaluate XOR gates for "free" without needing to construct garbled truth tables for them. This yields a construct that is as, if not more efficient than previously known constructs such as Yao's protocol or Pinkas's Fairplay [61] circuit. They then proceed to show that the Free XOR gates can be used in building circuits for many useful applications such as permutation networks and the Universally Composable (UC) framework. I describe their circuit construction and evaluation process below.

Let W_i denote a wire of the circuit and w_i^0, w_i^1 denote its two possible values. The circuit creator, Alice first chooses a random string, $R \in_R \{0,1\}^N$ and for each input wire W_i of the circuit, chooses a garbled value, $w_i^0 = \langle k_i^0, p_i^0 \rangle \in_R \{0,1\}^{N+1}$ and sets the other garbled input value, $w_i^1 = \langle k_i^1, p_i^1 \rangle = \langle k_i^0 \oplus R, p_i^0 \oplus 1 \rangle$. Next, Alice sorts the gates in topological order and depending on whether a gate g_i is an XOR gate or a regular gate, creates garbled truth tables. If g_i is an XOR gate with input wires W_a and W_b and garbled input values

$w_a^0 = \langle k_a^0, p_a^0 \rangle, w_b^0 = \langle k_b^0, p_b^0 \rangle, w_a^1 = \langle k_a^1, p_a^1 \rangle, w_b^1 = \langle k_b^1, p_b^1 \rangle$, then the output wire, $W_c = W_a$ XOR W_b: $w_c^0 = \langle k_a^0 \oplus k_b^0, p_a^0 \oplus p_b^0 \rangle, w_c^1 = \langle k_a^0 \oplus k_b^0 \oplus R, p_a^0 \oplus p_b^0 \oplus 1 \rangle$.

In case g_i is a regular, non-XOR gate, Alice chooses the output wires randomly by setting $w_c = \langle k_c^0, p_c^0 \rangle \in_R \{0,1\}^{N+1}$ and $w_c^1 = \langle k_c^0 \oplus R, p_c^0 \oplus 1 \rangle$. Alice then creates the garbled truth tables for each gate g_i with 4 entries in the table thus: for all possible $v_a, v_b \in \{0,1\}$, there is an entry of the form $e_{v_a,v_b} = H(k_a^{v_a} || k_b^{v_b} || i) \oplus w_c^{g_i(v_a,v_b)}$. Alice then sorts the truth table entries in the correct position by the position variables, $p_a^{v_a}, p_b^{v_b}$. Lastly Alice creates the output-wire garbled values for each output wire W_i with two output garbled values: w_i^0, w_i^1 thus: for $v \in \{0,1\}$, create $e_v = H(k_i^v || \text{“}out\text{”} || j) \oplus v$ and sorts the two entries in the table by the position of their p_c variables.

Bob, the evaluator receives the garbled truth tables and garbled inputs from Alice and can compute W_c for every gate by computing $w_c = \langle k_c, p_c \rangle = \langle k_a \oplus k_b, p_a \oplus p_b \rangle$, if g_i is an XOR gate. Else, if it is a regular gate, Bob computes w_c from the garbled truth table entries: $w_c = \langle k_c, p_c = H(k_a || k_b || i) \oplus e$ where e is in position $\langle p_a, p_b \rangle$ in the truth table. Bob then computes the output wire values thus: for each output wire W_i with garbled values $w_i = \langle k_i, p_i \rangle$, Bob computes $f_i = H(k_i || \text{“}out\text{”} || j) \oplus e)$ where e is chosen from position p_i.

It should be noted that the circuit construction process can be simplified by choosing R such that its least significant bit (lsb) is 1, which helps one to eliminate the p_a, p_b, p_c variables altogether, which is what I do in my protocols. Another point is that, in the case of XOR gates, each gate's garbled input would be constructed from other gates' inputs; as a result the space required for storing the garbled inputs would grow unbounded for a circuit with an arbitrary number of gates. In the case of regular, non-XOR gates, the garbled inputs are randomly generated and do not need to be stored for constructing other gates' inputs. This creates problems if one wishes to use this circuit construction technique with resource-constrained hardware devices such as a TPM. I currently am not aware of a good solution that completely solves this problem and in my TPM-based protocol, skirt this issue by treating all gates as regular, non-XOR gates.

Yao's original protocol was meant only for two-party secure computation, but can be easily extended to the multi-party case by requiring that all parties perform an OT with Alice to get their encrypted input bits and then send their encrypted input bits to the circuit evaluator, Bob who then evaluates the circuit with all parties' garbled inputs. Another way to extend Yao's protocol would be to have all parties jointly create and evaluate the function, instead of just Bob, which is one of the approaches explored in BMR. In this approach, all parties send pieces of their garbled inputs to all other parties and then each party independently evaluates the circuit. Of course, there is no guarantee that each party generates its garbled inputs correctly, and if one wants any such guarantee, each party needs to prove in zero-knowledge that it has generated its inputs correctly.

In general, starting from Yao's work, BMR and GMW, there have been a lot of solutions proposed for the dishonest adversarial model of secure multi-party computation; some representative work includes [59, 33, 4, 83, 68, 52, 45, 75, 84, 64, 47]. Almost all of them are based on either Goldreich's compiler [28] which uses generic zero-knowledge proofs or are based on cut-and-choose zero knowledge proofs which require the prover to prove in zero-knowledge that the circuit was constructed correctly. Lindell and Pinkas [59] present a protocol for secure function evaluation in the dishonest model where each input wire of all gates in the circuit is replaced by a gate with s input wires, where s is the statistical security parameter of the cut-and-choose operation. This increases the communication costs by a factor of s. Ishai et al. [45] study the **feasibility** of doing secure multi-party computation in the semi-honest model and the dishonest model and obtain theoretical results for the same but do not present any protocol for doing so. Mohassel and Franklin [64] and Jarecki and Shmatikov [47] present protocols where the party that is constructing the circuit proves gate-by-gate that the circuit was constructed correctly using cut-and-choose; subsequently Woodruff [84] suggested a way to reduce the communication cost of their cut-and-choose operation using expanders. Beaver, Micali and Rogaway's protocol (BMR) [4] is one of the first works in the dishonest model and most subsequent protocols are based on this. In this protocol, the parties jointly compute the garbled inputs and gate labels by themselves and then independently evaluate the circuit individually. They use cut-and-choose zero knowledge proofs to prove to each other that the gate labels and garbled inputs were computed correctly. Subsequently in independent work, Tate and Xu [83] and Naor, Pinkas and Sumner [68] pointed out a subtle flaw in the BMR protocol (specifically gate labels leaking information about input bits) and each suggested corrections to it.

To see why cut-and-choose techniques are inefficient to use in practice, let us consider the protocol given by Lindell and Pinkas [59] where each input wire of a gate is replaced with a sub-gate with s input wires, where s is the statistical security parameter of the cut-and-choose operation. A minimal value for s could be 100, and if one has a small-size circuit with 512 gates; the two input wires of each gate would be replaced by two sub-gates with 100 input wires each. In doing this, one would be increasing the number of inputs in the circuit by a factor of s - in this case, the number of input wires is increased from 1024 to 102400. The number of rounds of interaction would be linear in s: so if each of the 100 wires of s requires around 80 rounds of interaction to build up trust, the total number of interactions would be

81920. Even for a small-sized circuit of 512 gates the sheer number of rounds of interaction (communication cost) may make it impractical to implement in real-world applications.

Another model of communication is the **covert** adversarial model where an adversary may deviate from the steps of the protocol, but such deviations are mostly guaranteed to be caught by the honest parties. It should be noted that this model is still not the same as the malicious model since the probability of the deviant parties being caught is **not** negligibly close to 1. In the malicious model, one must guarantee that any deviant party **will** be caught with probability of 1 (or negligibly close to 1). The idea of a "covert" adversarial model was first introduced by Canetti and Ostrovsky [10] and later fully expanded and developed by Aumann and Lindell [2]. Recently Goyal et al. [33] present efficient protocols for the two-party and multi-party case for semi-honest and covert adversaries and they remark that their protocols can potentially be extended to the malicious adversary model by using cut-and-choose ZKPs. It should be noted that the covert adversarial model is very realistic and models many real-world situations such as businesses, financial services, political situations, etc. where honest behavior cannot be presumed, but parties involved may not openly cheat or proactively perform malicious actions, since they might have reputations to protect. For example, a bank would presumably never openly leak its customers' information since it would want to retain customers and these sort of activities would lead to loss of business and reputation.

Another line of work that takes the approach of relaxing assumptions is one in which one assumes a **honest majority**. It is assumed that not more than $1/3^{rd}$ or $1/2$ of the parties are malicious and the remaining are honest and guarantee that the correct function output will be delivered to all parties. Efficient protocols have been designed this way by

Damgård and Ishai [44] which makes use of a black-box pseudo-random generator. Damgård and Ishai's scheme can tolerate up to a maximum of $t < n/5$ malicious adversaries with minimal communication cost, and up to a maximum of $t < n/3$ and even $t < n/2$ at the expense of further increasing the communication cost. Both of these models - the covert model and the relaxed, honest-majority model are realistic and such protocols might find practical use in a lot of applications, but strictly from a theoretical standpoint, one would like to have protocols secure in the malicious model since it offers a much more stringent definition of security.

Lastly, staring from Chaum, Crep*é*au and Damgård, [12], there is another line of work in secure multi-party computation which focuses on achieving the notion of **unconditional** or information-theoretic security in case of a honest majority [3, 16, 21, 46]. All known protocols in this area can only be applied to restricted classes of functions such as NC^1 or non-deterministic log-space. Furthermore, since I deal with bounded, polynomial-time machines in this dissertation, I do not discuss this line of research further.

In addition to Yao-style garbled circuits, other solutions for the SFE problem involve homomorphic encryption, where the encryption function used is an additive or multiplicative homomorphic function. One of the popular instantiations of an additive homomorphic scheme is the Paillier cryptosystem [72] which provides a plaintext space that is the size of an RSA modulus; subsequently there have been many generalizations of the Paillier system with larger [18] and smaller [17] plaintext space. Sander et al. [78] have shown that it is possible to construct a secure two-party function evaluation protocol from a secure additive homomorphic encryption scheme. More generic results for constructing SFE protocols from multiplicative as well as additive homomorphic encryption schemes have been obtained by Melchor et al. [63], Armknecht and Sadeghi [1]. Koleshnikov et al. [55] propose a hybrid (garbled circuits + homomorphic encryption) scheme for computing basic integer arithmetic. Fully homomorphic encryption has been proposed [23] although its computational cost makes it un-desirable to implement. One of the interesting lines of work in homomorphic encryption is to investigate whether one can use trusted computing to develop fully homomorphic encryption-based (additive and multiplicative) solutions for SFE which I expand on in Chapter 6.

2.3. Trusted Computing

2.3.1. Trusted Platform Modules (TPMs)

In the last part (Chapter 6) of this dissertation, I consider the use of security hardware as developed by the Trusted Computing Group (TCG) to design SFE protocols in the presence of malicious adversaries. I note that all existing solutions for this problem use cut-and-choose Zero-Knowledge Proofs (ZKPs) which are expensive to implement in terms of communication cost. I replace the statistical trust gained by the use of ZKPs by a degree of trust in the TCG's trusted platform module (TPM). The TCG is an industry consortium of over 150 companies that has developed specifications for various security technologies, including a hardware chip called the trusted platform module (TPM) [35]. TPMs have become common in laptops, business-oriented desktops, and tablet PCs, including those by IBM, Dell, Lenovo, HP, and Toshiba, as well as in some server-class systems such as the Dell R300. TPMs are designed to be very cheap (under $5 each), and are intended to be easily embedded on the motherboard of a standard system. Primarily, the TPM is used for measurement of system parameters (BIOS, hardware elements, operating system, software applications, among others) to provide a measured and protected execution environment, which assures the integrity and trustworthiness of the platform. To support trustworthy reporting of these measurements to outside parties, TPMs sign measurements with “Attestation Identity Keys” (AIKs): keys that cannot exist in usable form outside the TPM, are only used inside the TPM to sign values produced inside the TPM chip itself, and are certified by a designated certification authority known as a PrivacyCA which vouches for the fact that the certified key is indeed a legitimate AIK which can be used only internally to a TPM. The exact process of establishing that a key is an internal-use only AIK relies on a chain of trust from the manufacturer of the TPM, and is beyond the scope of this paper to describe — interested readers are referred to the TPM documentation for details [35]. In addition to operations with measurements and AIKs, TPMs provide a variety of other capabilities, including the ability to generate random numbers (or at least cryptographically secure pseudo-random numbers), support for common cryptographic algorithms including RSA (for encryption and digital signatures), SHA-1 (for generating hashes), and HMAC (hashed message authentication code), as well as more specialized operations such as a privacy-oriented authentication technique known as direct anonymous attestation [9]. In addition to specifying functionality, the Trusted Computing Group has defined a Common Criteria protection profile [34] so that devices can be validated by third parties in order to increase trust in the device. It must be noted that the TPM specification is a work in progress and is evolving as per market requirements and feedback from application designers. Hence the idea of adding new functionalities to the existing TPM v.1.2 specification, which I explore in my methodologies isn't necessarily far-fetched.

Below I outline the salient features of a TPM, its core functionality and various capabilities.

2.3.1.1. TPM Architecture

Figure 2.1 shows the basic architecture and components of a standard v.1.2 TPM as given in the TPM 1.2 specification [35]. The various components are briefly explained below:

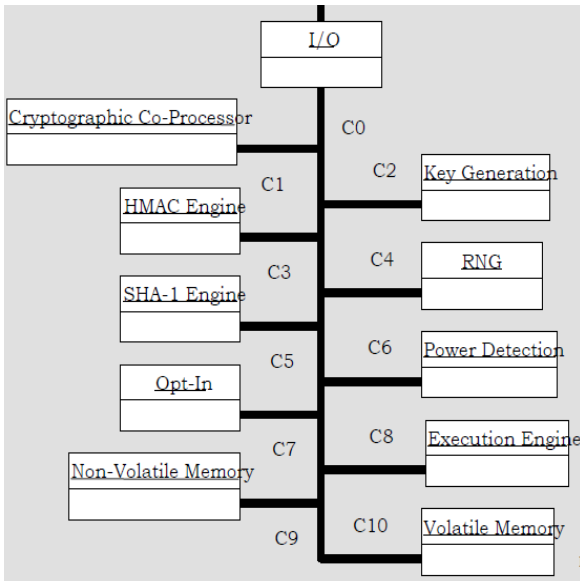


FIGURE 2.1. TPM architecture

(1) Secure Input and Output: This component manages information flow through the communication bus, performs protocol encodings, decoding and performs message routing besides enforcing access control policies and permissions for various TPM commands.

(2) Cryptographic co-processor: This implements cryptographic functionalities inside a TPM. The functionalities it implements are asymmetric key generation (RSA), asymmetric encryption and decryption (RSA), hash generation (SHA-1) and a random number generator (RNG). As of now, current version 1.2 TPMs do not offer

users the capability to use symmetric key operations, although TPMs use symmetric key encryption for internal commands.

(a) RSA: RSA is used for encryption and generating digital signatures. The TPM supports 512, 1024, 2048-bit RSA keys. The RSA exponent size as supported by the TPM is $e = 2^{16} + 1$. The TPM also uses RSA for signatures.

(b) Symmetric key encryption: The TPM currently supports symmetric key encryption only for internal operations such as encrypting authentication information, providing confidentiality in transport sessions and internal encryption of data blobs. The TPM uses a Vernan one-time-pad with XOR for this.

(c) Key generation and usage: The TPM creates RSA and symmetric key pairs and nonces. There is no time limit imposed by the TPM for the key generation process to complete.

(d) HMAC engine: The HMAC engine is used for checking the credentials of incoming authorization data for an object and also to guarantee that the command has not been modified in transit. The creation of the HMAC has to follow RFC 2104 [71]. HMAC, as implemented by the TPM v.1.2 uses SHA-1 as the hash algorithm and has a key length of 20 bytes.

(e) Random number generator (RNG): The RNG generates 32-byte random numbers which can be used in nonces, key generation and signatures. The structure of the RNG consists briefly of a state machine that accepts and mixes unpredictable data and a post-processor that has a one-way hash function such as SHA-1. For more details, readers are referred to the TPM v.1.2 specification [35].

(f) SHA-1 engine: The TPM implements SHA-1 as described in FIPS-180-2 [70]. The TPM generates 20-byte digests when the SHA-1 engine is invoked. The SHA-1 interfaces are exposed to the user to support measurement-taking.

(3) Power Detection: The power detection component synchronizes the power states of the platform with the power states of the TPM and helps the TPM enforce constraints such as restricting command execution subject to physical presence assertions.

(4) Opt-in: The Opt-in component allows the owner of the TPM to turn on/off and activate/deactivate the TPM. It also maintains the state of persistent and volatile flags and enforces the semantics associated with these flags.

(5) Execution engine: The execution engine runs program code to execute the TPM commands received through the I/O port.

(6) Non-volatile storage: Non-volatile memory is used to store persistent state associated with the TPM such as the Endorsement Key (EK) and is also available for use in application by the user of the TPM, although one needs to be careful while using non-volatile memory in applications since overuse of it can prematurely burn out the TPM.

TPMs support various kinds of keys; the main one being the Endorsement Key (EK), which is a 2048-bit RSA key created by the manufacturer of the TPM and is used for attesting the validity of the TPM. An Attestation Identity Key (AIK) is an alias for the EK (which due to security reasons cannot perform signatures). An AIK is also a 2048-bit RSA key and is created by the TPM owner after the TPM ships to them. A TPM can generate a virtually unlimited number of AIKs. A Storage Root Key (SRK) SRK is a keypair that is generated internally by the TPM and acts as a root of trust for all keys generated thenceforth by the TPM as shown in Figure 2.2[1]. Additionally, keys are further classified into migratable and non-migratable keys.

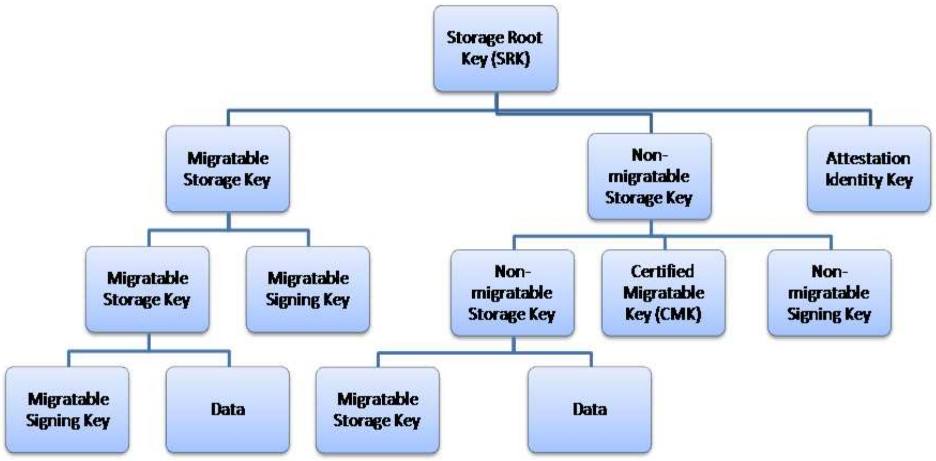


FIGURE 2.2. TPM key hierarchy. Ref: [37]

2.3.1.2. TPM Capabilities

Some of the key capabilities offered by the TPM are:

(1) Authorization protocols: The TPM supports authorization protocols to ensure that the originator of a command has the requisite permissions to access the objects referenced by the command and perform the command. The authorization credentials are passed to the TPM and the command is executed within exclusive (and sometimes logged) transport sessions. The three main authorization protocols supported by the TPM are Object Independent Authorization Protocol (OIAP), Object Specific Authorization protocol (OSAP) and Delegate Specific Authorization Protocol (DSAP).

[1]Reproduced with permission from the author of [37]

(2) Monotonic counter: A TPM provides a set of 4 physical monotonic counters which provide an ever-increasing incremental value. One of these counters is maintained internally by the TPM called as the "internal base" and the remaining three can be used by user applications. The TPM permits only one counter to be used in a particular boot cycle and the TPM must be reset before any other counter is used. Also, a counter can be incremented only once every 5 seconds to prevent burn-out of non-volatile storage and the TPM will throttle any attempts to increment the counter at a higher frequency.

(3) Transport sessions: A TPM allows the creation of an exclusive, logged transport session for executing commands if the user so wishes. Commands are executed in secure sessions and are wrapped in a session key and a session log is maintained. The user can execute any command in a transport session, except commands that would create a nested transport session.

(4) Delegation: A TPM will allow its owner to delegate some of their privileges to perform administrative tasks to other entities which are usually trusted processes.

(5) Direct Anonymous Attestation (DAA): DAA responds to a challenge about the attestation held by a TPM without revealing the attestation itself - something along the lines of a zero knowledge proof. Also, using DAA, groups of TPMs can join together can produce a common group credential. DAA is a very resource-intensive operation and must be used sparingly.

In addition to the main capabilities described above, the TPM supports other minor capabilities such as output redirection, migration of keys and data, revoking trust in a TPM, enforcing physical presence restrictions on data access, auditing of commands and context management.

2.3.2. Realizing Cryptographic Functionalities Using TPMs

One of the emerging areas of work within cryptography that has garnered much attention and investigation recently is the concept of hardware-assisted security. There has been a fair amount of recent research in realizing various cryptographic functionalities using the idea of trusted computing, whether abstract hardware tokens or more concrete instantiations such as TPM chips. In this section I review recent results that use trusted computing in any form for realizing cryptographic goals, either for specific functionalities (such as UC-secure computation, non-interactive oblivious transfer, random oracles, verifiable encryption, fair exchange) or generic secure function evaluation similar to what I investigate in Chapter 6. I then compare my work with others to put things in context with previous or parallel work.

The idea of hardware-assisted security can be traced back to the e-cash scheme due to Chaum and Pederson [13] where hardware tokens facilitate secure transactions between customers and a bank and Goldreich and Ostrovsky's oblivious memory model or ORAM [29] that enables software to run in the presence of untrusted memory. This idea was revisited some time back by Katz [51] who studied the problem of realizing generic UC-secure two-party computation relying on cryptographic assumptions using tamper-proof stateful hardware tokens where each party constructs its own hardware token. In independent, but parallel work, Chandran et al. [11] and Damgard et al. [19] improve upon Katz's work by making the hardware tokens independent of the parties using them besides being stateless and rely on more general assumptions (trapdoor permutations) rather than cryptographic assumptions. Moran and Segev [66] improved upon these results by requiring that only one of the two parties involved needs to have a hardware token, although it needs to be a stateful token. Hazay and Lindell [41] propose efficient protocols using smartcards for oblivious set intersection and various database search operations. More recently Goldwasser et al. [30] coined the term "one-time programs" where they introduced the concept of programs that can run exactly once (or more generally, k times) on any given input and then the program self-destructs and cannot be run again. Recently, Goyal et al. [32] have studied the feasibility of realizing secure computation in a general sense using hardware tokens. The main focus of their work is to study the feasibility of realizing secure computation using hardware tokens under various considerations (UC-secure vs. stand-alone model, stateless vs. stateful tokens, and unconditional vs. computational adversaries). The closest to my work is a recent paper by Järvinen et al. [48] in which they consider the SFE problem in a two-party setting where a server issues a low-cost tamper-proof hardware token to a client. They use garbled circuits with free XOR gates for SFE in the presence of semi-honest, covert and malicious adversaries. Clearly at a fundamental level, this idea is similar to mine with the difference that I use a specific piece of hardware - a TPM chip, and do not require cut-and-choose ZKP, which [48] still requires for the malicious model.

The above papers were about abstract hardware tokens. There has been also been work in the area of instantiating the hardware token as a TPM chip and investigating whether this can help us design cryptographic protocols in more efficient ways than those known before. In particular, Gunupudi and Tate [40] applied **count-limited objects** to create non-interactive protocols for oblivious transfer, designing a particularly efficient technique for a set of concurrent k-of-n oblivious transfers. In addition, Gunupudi and Tate give an application of this protocol to secure mobile agents, removing interaction requirements that were present in all previous secure agent protocols. In other work, Gunupudi and Tate [39], show how a TPM with some slight modifications can be used to act in a way indistinguishable from a random oracle, which can then be used in multi-party protocols. The resulting functionality is called a “TPM-Oracle,” and Gunupudi and Tate proved that any protocol secure in the random oracle model is also secure when using a TPM-Oracle. Finally, Tate and Vishwanathan [82] obtain an efficient non-interactive protocol for verifiable group encryption and the related problem of fair exchange that requires a minor modification to the current TPM specification called “certified randomness” based on standard encryption methods (not Paillier encryption) that eliminated the problem of cut-and-choose ZKPs in earlier protocols for the same applications.

To put my work in context of previous work, I consider general computation, not application-specific [68, 82, 40]. I also consider a specific kind of hardware support - a tamper-evident TPM chip and do not consider generic or custom-designed tamper-proof hardware tokens [32, 48, 51, 11, 19] where the TPM chip provide blackbox access to its owner to its functionality. Some of the previous work [32] focuses on examining **feasibility** of implementing cryptographic functionalities using trusted hardware and my focus is on implementing secure computation in the malicious model in an **efficient** way, reducing costs that are there in existing methodologies.

CHAPTER 3

THEORETICAL MODELS AND DEFINITIONS

This chapter contains definitions for the cryptographic paradigms, tools and models used to conceptualize and define solutions to problems considered in this dissertation. Following a few initial comments about machines and proof models, I provide definitions for secure function evaluation (SFE) in the semi-honest and dishonest adversarial models and define private information retrieval (PIR) and oblivious transfer (OT). I also define the concepts of computational indistinguishability, secure-multi-party computation, computational zero knowledge, and the security properties of Hashed Message Authentication Code (HMAC) when used as a Pseudo-Random Function (PRF).

Adversaries, and all other parties involved in cryptographic protocols that provide computational security are usually considered to be Probabilistic Polynomial-time (PPT) Turing machines with access to oracles or black-box functions. The adversary is assumed to be a randomized algorithm that always halts after a polynomial number of steps independent of the internal outcome of its coin-tosses. The number of coin tosses for a PPT machine M is bounded by a polynomial, T_M in its input length. The machine always makes $T_M(|x|)$ coin tosses. I only consider such machines in this dissertation and do not consider unconditionally secure protocols or non-uniform polynomial time machines.

For reasoning about security and providing meaningful guarantees that are amenable to rigorous analysis, one needs to prove that the designed protocols are secure under a given set of assumptions and against specific kinds of adversaries. Two of the popular proof paradigms in cryptography are the simulation-based proof paradigm and the reduction-based proof paradigm. In simulation-based proofs, I replace the honest parties by a simulator and prove that whatever can be computed by a dishonest party from its view of the protocol in question can also be computed by it from just its own inputs and outputs. Another way to approach a simulator-based proof is to show that whatever can be computed by a dishonest party in its interaction with the honest parties in the "real world" can also be computed by it while interacting with a trusted third party in an "ideal world". This implies that the actual execution of the protocol does not yield any extra information that the adversary does not already have access to. A protocol in the real model is said to be secure if any real-world adversary can be converted into an ideal-world adversary such that their output distributions are computationally indistinguishable.

In reduction-based proofs, I bound the advantage of an adversary with respect to a well-known hard problem. So basically, I prove that if an adversary can break the security of my scheme, then it can break the security of a well-known cryptographically hard problem, such as reverting a one-way function or breaking the CCA-security of a cryptosystem that is known to be CCA-secure, or solving discrete logs in polynomial time, etc. In this dissertation, I primarily use simulation-based proofs. I next present definitions about key concepts that will be referred to later at various points in this dissertation. The definitions presented here are according to Goldreich [27].

3.1. Preliminaries

Computational indistinguishablity: Let there be two distribution ensembles, $X = \{X_w\}_{w \in S}$ and $Y = \{Y_w\}_{w \in S}$ where S is the infinite set, and $w \in S \cap \{0,1\}^k$ for all sufficiently large k, where k is a security parameter. One would say that X and Y are computationally indistinguishable if, for every non-uniform polynomial-time circuit family, $\{C_k\}_{k\in\mathbb{N}}$ (also known as distinguisher) and every positive polynomial $p(\cdot)$, it holds that:

$$|\Pr[C_k(X_w) = 1] - \Pr[C_k(Y_w) = 1]| < \frac{1}{p(k)}$$

Zero-knowledge Proofs: The notion of zero knowledge that that I refer to in this work is computational zero knowledge (CZK) (as opposed to statistical or perfect zero-knowledge) which is defined as follows: Let (P,V) be an interactive proof system for some language L. One would say that (P,V) is computational zero-knowledge if for every probabilistic polynomial-time interactive machine V^* there exists a probabilistic polynomial-time algorithm M^* such that the following two ensembles are computationally indistinguishable:

- $\{\langle P, V^*\rangle\}_{x\in L}$ i.e., the output of the interactive machine V^* after it interacts with the interactive machine P on common input x
- $\{M^*(x)\}_{x\in L}$ i.e., the output of machine M^* on input x

Machine M^* is called a simulator for the interaction of V^* with P.

Next, I formalize what it means for a TPM to be secure; this definition reflects the requirements given in the TPM Protection Profile [34]. I use this definition in Chapter 6, where I present TPM-assisted solutions for secure function evaluation.

3.1. DEFINITION (Trusted Platform Security Assumption). The **Trusted Platform Security Assumption** is the assumption that the system containing a TPM satisfies the following properties:

(1) **Tamper-resistant hardware:** It is infeasible to extract secrets stored in protected locations in the TPM.

(2) **Secure Encryption:** The public-key encryption algorithm used by the TPM is CCA-secure.

(3) **Secure Signatures:** The digital signature algorithm used by the TPM is existentially unforgeable under adaptive chosen message attacks.

(4) **Trustworthy PrivacyCA:** Only valid TPM-bound keys are certified by a trusted PrivacyCA.

3.2. Secure Function Evaluation (SFE) and Its Two Models

In the second and third parts of this dissertation (Chapter 5 and Chapter 6), I use the concept of SFE in the presence of semi-honest and malicious adversaries respectively. In this section, I define the concept of SFE and SFE in the semi-honest and malicious models.

Secure function evaluation: An m-party cryptographic protocol can be cast by specifying a random process that maps m inputs to m outputs where the m inputs are the local inputs of each party and the m outputs are the outputs they receive at the end of the protocol. More formally, an m-ary functionality f can be defined as: $f : (\{0,1\}^*)^m \to (\{0,1\}^*)^m$ which maps sequences of the form $\bar{x} = (x_1, x_2, \cdots, x_m)$ into sequences of random variables $f(\bar{x}) = ((f_1(\bar{x}), \cdots, f_m(\bar{x}))$. So, at the end of the protocol, a party i would wish to obtain $f_i(x_1, x_2, \cdots, x_m)$. In my set-up, I assume that the output of all parties at the end of computation is similar, i.e., for any two parties i and j, $f_i(x_1, x_2, \cdots, x_m) = f_j(x_1, x_2, \cdots, x_m)$.

3.2. Definition (Secure function evaluation in the semi-honest model). Let $f : (\{0,1\}^*)^m \to (\{0,1\}^*)^m$ be an m-ary functionality, where $f_i(x_1, \cdots, x_m)$ denotes the i^{th} element of $f(x_1, \cdots, x_m)$. For $I = \{i_1, \cdots, i_t\} \subseteq [m]\underline{\underline{def}}\{1, \cdots, m\}$, let $f_I(x_1, \cdots, x_m)$ denote the subsequence $f_{i_1}(x_1, \cdots, x_m), \cdots, f_{i_t}(x_1, \cdots, x_m)$. Let Π be an m-party protocol for computing f. The view of the i^{th} party during an execution of Π on $\bar{x} = (x_1, \cdots, x_m)$, is denoted by $\text{view}_i^{\Pi}(\bar{x})$, and for $I = \{i_1, \cdots, i_t\}$, we let $\text{view}_I^{\Pi}(\bar{x}) \underline{\underline{def}} (I, \text{view}_{i_1}^{\Pi}(\bar{x}), \cdots, \text{view}_{i_t}^{\Pi}(\bar{x}))$. Let $\overset{c}{\equiv}$ denote computational indistinguishability by non-uniform families of polynomial-time circuits. We say that Π privately computes f if there exists a probabilistic polynomial-time algorithm denoted S, such that for every $I \subseteq [m]$, it holds that

$$\{S(I, (x_1, cdots, x_{i_t}), f_I(\bar{x})), f(\bar{x})\}_{\bar{x}} \in (\{0,1\}^*)^m \overset{c}{\equiv} \{(\text{view}_I^{\Pi}(\bar{x}), \text{output}^{\Pi}(\bar{x}))\}_{\bar{x}} \in (\{0,1\}^*)^m$$

Informally put, this says that the view of all the parties in I can be efficiently simulated solely based on their inputs and outputs.

3.3. Definition (Secure function evaluation in the malicious model). I first state what it means for a protocol to be secure in the ideal model, and then state what it means for a protocol to be secure in the real-world model. Finally I state that a protocol is secure in the presence of malicious adversaries if the real and ideal world executions are computationally indistinguishable.

- **Ideal Model**: Let m denote the number of parties, let $f = (f_1, f_2, \cdots, f_m)$ be a probabilistic polynomial-time (PPT) m-ary functionality to be computed ($f : (\{0,1\}^*)^m \rightarrow (\{0,1\}^*)^m$) and Π be a multi-party protocol for computing f. Let Real_f and Ideal_f denote the real and ideal model executions of the protocol respectively. Let $\overset{c}{\equiv}$ denote computational indistinguishability by non-uniform families of polynomial-time circuits. Let $I = i_1, \cdots, i_t \subseteq [m] \underline{\underline{\text{def}}} \{1, \cdots, m\}$ denote the set of malicious parties, let $\bar{I} = [m] - I$ denote the set of honest parties, and

$(x_1, \cdots, x_m)_I = (x_{i_1}, \cdots, x_{i_t})$. If B is a PPT algorithm, then the pair (I, B) represents an adversary in the ideal model. Let z be the auxiliary input[1] given to the parties, let r be a uniform random tape for the adversary, and let the ideal model execution of the protocol be defined by $\mathrm{Ideal}^1_{f,I,B(z)}(\bar{x}) \underline{\underline{\mathrm{def}}}\ \gamma(\bar{x}, I, z, r)$. The JOINT EXECUTION of f under (I, B) is defined as follows (assuming the trusted third party returns the output of the computation to the parties in order of the party numbers/IDs):

(1) In case Party 1 is honest ($1 \notin I$):

$$\gamma(\bar{x}, I, z, r) \underline{\underline{\mathrm{def}}} (f_{\bar{I}}(\bar{x'}), B(\bar{x_I}, I, z, r, f_I(\bar{x'}))$$

(2) In case Party 1 is dishonest ($1 \in I$):

$$\gamma(\bar{x}, I, z, r) = \begin{cases} (\perp^{|\bar{I}|}, B(\bar{x_I}, I, z, r, f_I(\bar{x'}), \perp)) & \text{if } B(\bar{x_I}, I, z, r, f_I(\bar{x'})) = \perp \\ (f_{\bar{I}}(\bar{x'}), B(\bar{x}, I, z, r, f_I(\bar{x'}))) & \text{otherwise} \end{cases}$$

where, in both cases $\bar{x'} \underline{\underline{\mathrm{def}}} (x'_1, \cdots, x'_m)$ such that $x'_i = B(\bar{x_I}, I, z, r)_i$ if i is malicious, and $x'_i = x_i$ if i is honest.

- **Real Model**: Let Π denote a real-model protocol for computing f, let z be the auxiliary input provided to all parties (including the adversary), let $I = i_1, \cdots, i_t \subseteq [m] \underline{\underline{\mathrm{def}}} = \{1, \cdots, m\}$ be the set of malicious parties, and let A be the adversary. The JOINT EXECUTION of Π under (I, A) in the real model, on input a sequence $\bar{x} = (x_1, \cdots, x_m)$, denoted $\mathrm{Real}_{\Pi,I,A(z)}(\bar{x})$ is defined as the output sequence resulting

[1]Since we need an upper bound on the complexity of the functionality f, we give this bound explicitly to all parties as an auxiliary input.

from the interaction between the m parties where the messages of parties in I (set of malicious parties) are computed according to $A(\bar{x}_I, I, z)$, and the messages of parties in $\bar{I} = [m] - I$ or the set of honest parties are computed according to Π.

- **Security in the malicious model**: The protocol Π that computes the functionality f is said to be secure if for every real-model PPT adversary A, there exists an ideal-model PPT adversary B, such that for every $I \subseteq [m]$ and $|I| < m/2$, it holds that:

$$\{\text{Ideal}^1_{f,I,B(z)}(\bar{x})\}_{\bar{x},z} \overset{c}{\equiv} \{\text{Real}_{\Pi,I,A(z)}(\bar{x})\}_{\bar{x},z}$$

3.3. Private Information Retrieval (PIR)

Let p and p' be large primes, and $N = p \cdot p'$ be a composite modulus. Let Z_N^* denote the set of numbers that are co-prime or relatively prime with N. The set of **quadratic residues** is defined by $QR_N = \{y \in Z_N^* | \exists x \in Z_N^* : y = x^2 \bmod N\}$, and the set of **non-quadratic residues** is the complement of QR_N, written QR'_N. If one denotes the set $Z_N^{+1} = \{y \in Z_N^* | (y/N) = 1\}$, where (y/N) denotes the Jacobi symbol, exactly half of the numbers in Z_N^{+1} are in the set QR_N and the other half are in QR'_N. Determining whether a given $x \in Z_N^{+1}$ is in QR_N or in QR'_N is easy if the factorization of N is known, but is commonly believed to be computationally intractable if the factorization of N is unknown; this is a widely-used cryptographic assumption known as the "quadratic residuosity assumption." For a detailed theoretical exposition, readers are referred to [14].

To form a PIR query, the client first randomly selects two $f/2$ bit primes, p and p' and computes $N = p \cdot p'$ where N is f bits long. The query is then formed as $E(I) = q =$

$[q_1, \ldots, q_n]$, where each q_i is drawn from Z_N^{+1} such that $q_I \in QR'_N$ and $\forall i \neq I, q_i \in QR_N$. The server computes $v(X, q) = \Pi_{j=1}^{n} w_j$, where $w_j = q_j^2$ if $X_j = 0$, or q_j otherwise. Note that $v(X, q) \in QR_N$ if and only if $X_I = 0$, so the client uses its knowledge of p and p' to test whether $v(X, q) \in QR_N$ and thus determines the value of X_I. As just described, this produces a query of size $\Theta(fn)$ and a response of size $\Theta(f)$; however, in the simplest form of the Kushilevitz/Ostrovsky protocol the server's data is organized into a $\sqrt{n} \times \sqrt{n}$ matrix, and this basic PIR step is run on every row with a single $E(I)$ vector. As a result, the query size is $\Theta(f\sqrt{n})$ and the response size is also $\Theta(f\sqrt{n})$. Further optimizations are possible — see the original paper for details [57].

It should be noted that, like many cryptographic protocols, the traditional PIR definition above is one in which data items are **single bits**; in practice the items would typically be multiple bits

3.4. Oblivious Transfer (OT)

In this dissertation, I use Naor and Pinkas's OT [76] — I describe their technique for doing 1-of-N OT using an existing 1-of-2 OT below. The protocols below handle m-bit messages directly, rather than relying on multiple single-bit OTs.

Let Bob be a server who holds an N-element string $X_1, \cdots, X_N$ where each $X_i \in \{0, 1\}^m$ is an m-bit value. Let Alice be a client who wishes to obtain element X_I. The 1-of-N OT protocol proceeds thus: Bob first prepares $l = \lceil \log N \rceil$ random key-pairs $(K_1^0, K_1^1), (K_2^0, K_2^1), \cdots, (K_l^0, K_l^1)$ where for all $1 \leq j \leq l$ and $b \in \{0, 1\}$, K_j^b is a t-bit key to a pseudo-random function F_K. For all $1 \leq I \leq N$ let $(i_1, i_2, \cdots, i_l)$ be the bits of I. Bob encrypts each element in the string as $Y_I = X_I \oplus \bigoplus_{j=1}^{l} F_{K_j^{i_j}}(I)$ and sends all the encrypted strings $Y_1, \cdots, Y_N$ to Alice. Alice and Bob then engage in a 1-of-2 OT for each pair of keys (K_j^0, K_j^1). If Alice wants element X_I, she picks key $K_j^{i_j}$. Finally Alice reconstructs $X_I = Y_I \oplus \bigoplus_{j=1}^{l} F_{K_j^{i_j}}(I)$.

3.5. Hashed Message Authentication Code (HMAC)

In the last part of this dissertation (Chapter 6), I use HMAC as a PRF [6] and prove the security of the TPM-assisted protocols using HMAC's security properties which are defined below:

HMAC: Let $h : \{0,1\}^c \times \{0,1\}^b \rightarrow \{0,1\}^c$ be a compression function ($b = 512$ bits, $c = 128$ or 160 bits). Let **pad** be the padding function such that $s^* = s \parallel \text{pad}(|s|) \in B^+$ for any string s, where B^+ denotes the set of all strings of length a positive multiple of b bits. The hash function H is defined by $H(M) = h^*(IV, M^*)$ where IV is a c-bit initialization vector and $M \in B^+$ is the message to be hashed. Let $K_{out} = K \oplus opad$ and $K_{in} = k \oplus ipad$ denote the two keys of the HMAC, where K is a random b-bit key and $opad, ipad$ are fixed, distinct constants. Then the HMAC of the message, M can be written as:

$$HMAC(K_{out}||K_{in}, M) = h^*(IV, K_{out} \parallel h^*(IV, K_{in} \parallel M \parallel pad(b + |M|)) \parallel pad(b + c))$$

CHAPTER 4

SINGLE-USER, CLIENT-SERVER MODEL

4.1. Problem Definition

One of the common queries in location services are user queries about their nearest neighbor(s) in which a user might query a location server about finding their approximate or exact nearest neighbors. The "neighbors" could be points of interest (POIs) in the vicinity such as restaurants, convenience stores, shops, etc., e.g., "Where is the nearest gas station?" In such scenarios, one needs to find an efficient way to protect the privacy and anonymity of the user/client, while ensuring that the server answers location-related queries in as precise a manner as possible, keeping the redundant data to a minimum, and has some control over what data is revealed. Also, the location database would be a valuable asset for the server and if the server is operating on a commercial basis, it would be charging the client for each query; revealing extra information may not be in the business interests of the server. Specifically, if a user u has a location l and needs to query a location-based server (LBS) to obtain its nearest neighbor from a set of Points of Interest (POIs) S where $S = \{s_1, s_2, \ldots, s_m\}$, the goal is to develop an efficient protocol that does not allow the LBS to learn l while reducing the subset of S to be sent to u in order to answer the query. In this chapter, I present a solution framework for the exact nearest neighbor query.

4.2. Brief Summary of Related Work

Previous work in privacy in location services in the single-user model focus on cloaking techniques using trusted anonymizers [22, 50, 65] or a network of trusted peers. In the trusted anonymizer approach, service requests are first transmitted to a third-party trusted server called location anonymizing server (AS) which strips off all user-identifying information and sends the query to the location server. One major problem with this approach is that the anonymizer becomes a single point of failure and a performance bottleneck, besides the client and server will have to trust the anonymizer. Another solution is to organize the network as a peer-to-peer network, instead of having a centralized location server [15, 26]. In this approach, a mobile client first finds a peer group that meets the privacy requirement (i.e. find $\mathcal{K} - 1$ neighbors) through peer searching. Then the client calculates a region that includes the $\mathcal{K} - 1$ peers (called K-anonymity [36]). This approach requires the construction of a large peer network of mutually trusting and honest users and cannot guarantee service availability when clients are sparsely distributed. The paper by Ghinita et al. [25] presents a solution based on private information retrieval (PIR) [14]. This approach has several advantages: it does not reveal any spatial information, it is resistant against correlation attacks, does not require any trusted third party among others, and is provably secure. However the server still reveals large amounts of extra information which results in exposing large portions of the server's data assets to the client. There are other approaches to the privacy problem too, which use variants of PIR [54] and trusted computing [42]. For a detailed review, readers are referred to Chapter 2.

4.3. Contributions

I present an efficient solution for the exact nearest neighbor query where along with ensuring the client's privacy, the data returned by the server is as precise as possible. My solution is a general-purpose one and can use either a two-phase PIR or a combination of PIR and oblivious transfer (OT). I use the layout of the server database similar to that in [25] where the server super-imposes a grid on the POIs. A user u in a grid cell wants to know its nearest neighbor but does not want to release its own location or the grid cell that (s)he is in.

Using the single PIR approach proposed in [25] on the grid will result in the server returning an entire column of the grid, thus releasing more information than is necessary to the client. It should be noted that if the data (locations) are not arranged in a grid, but as a one-dimensional list, one can still use PIR to return just one cell, but this approach would be very expensive. Using 1-of-n OT [76] by itself will require the server to transmit the whole grid of encrypted POIs to the client, the cost of which is prohibitive. Thus, by using OT over the entire grid, one would have incurred very high communication costs, using single-phase PIR over the columns of the grid, the volume of data given/revealed by the server would be too high, and using single-phase PIR over cells of the grid would return precise data, but would be too expensive.

One way to solve this problem would be to use a two-phase protocol wherein the server recursively performs a PIR and/or OT on the grid, and the other way is to define a protocol that combines the features of PIR and OT. I explore both of these approaches in my methodology. In Section 4.4, I present a two-phase framework in which one can use three different combinations of PIR and OT: PIR+OT, OT+PIR, PIR+PIR and discuss why a two-phase OT approach (OT+OT), while possible in principle, isn't very efficient in terms of cost. In Section 4.5 I describe a single-phase approach where one can use either (1) PIR over a square grid as described in [25], or (2) 1-of-n OT over the entire database organized as single-dimensional list, or (3) a **random** OT protocol which offers a lesser degree of privacy. In Section 4.6, I formally define the security properties of my protocol and give a proof sketch. Chapter 7 contains experiments where I analyze the cost of my two-phase framework and compare it with the single-phase approaches.

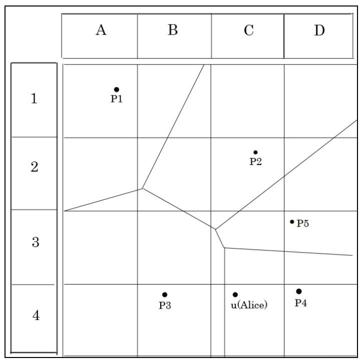
4.4. A Two-Phase Framework

For organizing the location database, I follow the same basic methodology given in [25]. As part of pre-processing, the whole space is divided into Voronoi tessellations using the set of POIs, $S = \{s_1, s_2, \ldots, s_\alpha\}$ and a grid or matrix of size $M = \lceil\sqrt{n}\rceil \times \lceil\sqrt{n}\rceil$ is superimposed on top of it as shown in Figure 4.1. For each grid cell gc (which can also be referred to as $M_{a,b}$ where a is the row number and b is the column number), the server computes the POIs located in all the Voronoi cells that intersect gc and prepares a neighbor list that contains the possible nearest neighbors for any point in the cell. Figure 4.1 shows the neighbor list for each grid cell. The server sets the maximum number of POIs in the lists to a constant fixed number, P_{max} (e.g., in Figure 4.1, $P_{max} = 3$). In case some of the cells do not have enough POIs in the list, the server pads the list with duplicate entries till it reaches the value of P_{max}. This can be seen in Figure 4.1 in the neighbor lists for A_1, C_2, etc. In the figure, I have used duplicate entries for the padding, but one can use a pre-agreed padding value as well.

When a user u with location l requests its nearest neighbor, it simply requests the neighbor list associated with the grid cell that contains l and filters the results to find the nearest neighbor of l. Let $M_{a,b}$ be the matrix cell corresponding to the user's location l. Using PIR, in [25], the user selects a query message $y = [y_1, ..., y_{\lceil\sqrt{n}\rceil}]$ ($y_i \in Z_n^{+1}, i = 1, 2, \ldots, n$) where $y_b \in QR'$ and $\forall_j \neq b, y_j \in QR$ (Here b is the column the user is located in). The server computes for each row, r of the matrix, M, the value $z_r = \Pi_{j=1}^{\lceil\sqrt{n}\rceil}(w_{r,j})$ and returns $z = [z_1, ..., z_{\lceil\sqrt{n}\rceil}]$. Here $w_{r,j} = y_j^2$ if $M_{r,j} = 0$, or y_j otherwise. The client can then use a similar method to calculate the values of $M_{*,b}$ where $*$ represents any row in M.

Since the data (locations) are organized in a grid, this leads to the server returning extra information which the user has not queried about, wherein most of the data returned is not relevant. Specifically, if the user requests one grid or matrix cell $M_{a,b}$, the server, along with $M_{a,b}$ returns the contents of the entire column b. Since the size of the data returned increases with the size of the grid, the user has to sift through a lot of extraneous data, or data that might not be particularly useful to get their nearest neighbors. In the optimization techniques proposed in [25], data compression and using a rectangular matrix instead of a square one are cited as ways to reduce the amount of data returned by the server. By compressing the query results returned by the server, one would just be removing duplicates and reducing the size of the data returned, without reducing the actual content returned. Organizing the data as a rectangular matrix would return as many extra locations as there are rows in the grid. Neither of these approaches ensure that the server always returns just **one** grid cell and its corresponding neighbors list. One could possibly achieve this by organizing the location database as a one-dimensional list, however, this approach would be too expensive; my two-level approaches ensure that the server always returns only one cell to the user and at a reasonable cost. Furthermore, my approaches are orthogonal to techniques proposed in [25] and can be applied on top of their methodology.

In my two-phase framework, the user first asks the server to keep aside its encrypted column, b, using PIR or OT. In the second step, to retrieve its cell in the column, i.e., cell $M_{a,b}$, the user can use either PIR, if they had used PIR or OT in the first step, or 1-of-$\sqrt{n}$ OT, if they had used PIR in the first step. Below I describe my framework for the exact nearest neighbor query, EXACT_NN with three cases: PIR+OT, PIR+PIR, OT+PIR, and the two sub-protocols which are used in them: $OT^1_{\sqrt{n}}$SERVER and $OT^1_{\sqrt{n}}$CLIENT where $OT^1_{\sqrt{n}}$ refers to 1-of-$\sqrt{n}$ OT. I also discuss why the possible fourth case: OT+OT, although possible, isn't as efficient as the other three approaches.



A1	P1,P1,P1
A2	P1,P3,P3
A3, A4, …..	………..
B2	P1,P2,P3
C2	P2,P5,P5
C4	P3,P4,P4

FIGURE 4.1. Sample dataset for protocol illustration

Protocol 1 EXACT_NN WITH PIR+OT

(1) Server – Bob uses the POIs S to create Voronoi tessellations of the space.

(2) Bob divides the space into an $\sqrt{n} \times \sqrt{n}$ grid M, so the grid is represented by $(M_{1,1}, \cdots, M_{\sqrt{n},\sqrt{n}})$. The neighbor list for each grid cell is prepared. Here, the contents of each grid cell is an m-bit string.

(3) User – Alice initiates a query in order to find the neighbor list of the cell she is in. Alice also randomly generates a composite modulus $N = p \cdot p'$, where p, p' are large

primes, and sends N to Bob. Note that only Alice knows the factorization of N, not Bob.

(4) Bob sends the granularity of the grid.

(5) Let Alice be located in $M_{a,b}$. Alice selects the column in which her cell is located, i.e., column b. At this point, Alice can choose to engage in either of two protocols: PIR or $OT^1_{\sqrt{n}}$Client. For this protocol, Alice chooses to engage in PIR.

Let $PIR(b) = [y_1, ..., y_{\lceil\sqrt{n}\rceil}]$ such each $y_i \in Z_N^{+1}$, $y_b \in QR_N$, and $\forall j \neq b, y_j \notin QR_N$. Alice issues a PIR request $PIR(b)$ to Bob.

(6) Bob prepares the PIR response, $z = [z_1, \cdots, z_{\lceil\sqrt{n}\rceil}]$ for each row, r of the matrix thus:

$$z_r = \Pi_{j=1}^{\lceil\sqrt{n}\rceil}(w_{r,j})$$

where $w_{r,j} = y_j^2$ if $M_{r,j} = 0$, or y_j otherwise. In total there will be m responses for each z_i (denoted by $z_i[1], z_i[2], \cdots, z_i[m]$). At this point Bob can choose to engage in either of two protocols: $OT^1_{\sqrt{n}}$Server or PIR. For this protocol, Bob chooses $OT^1_{\sqrt{n}}$Server. Bob runs the protocol $OT^1_{\sqrt{n}}$Server(z) where z is the PIR response for each bit of the m-bit string.

(7) Alice runs the $OT^1_{\sqrt{n}}$Client protocol for obtaining the a^{th} element in z. In the course of the $OT^1_{\sqrt{n}}$Client(a, Y) protocol, Alice gets the encrypted values $Y = [Y_1, \cdots, Y_{\lceil\sqrt{n}\rceil}]$, of which Alice can only decrypt $Y_a = z_a$. After decrypting z_a Alice can check if each bit i ($i = 1, 2, \ldots, m$) of $z_a[i] \in QR_N$ or $z_a[i] \in QR'_N$ using the formula $(z_a[i]^{\frac{p-1}{2}} = 1 \bmod p) \wedge (z_a[i]^{\frac{p'-1}{2}} = 1 \bmod p')$. If this expression evaluates to true, then it implies that $z_a[i]$ is in QR_N. For each bit, if $z_a[i] \in QR_N$, then Alice

concludes that bit is 0, else if $z_a \in QR'_N$, the bit is 1. Alice repeats this process for each bit of the m-bit string. Once she does this, she can get the contents of her cell $M_{a,b}$. From the contents of $M_{a,b}$, she can get the list of neighbors (POIs) associated with $M_{a,b}$. Once Alice knows the set of POI's, she can calculate their individual distances and find her nearest neighbor.

(8) Return the nearest neighbor.

Protocol 2 Exact_NN with OT+PIR

(1) Server – Bob uses the POIs S to create Voronoi tessellations of the space.

(2) Bob divides the space into an $\sqrt{n} \times \sqrt{n}$ grid M, so the grid is represented by $(M_{1,1}, \cdots, M_{\sqrt{n},\sqrt{n}})$; The neighbor list for each grid cell is prepared. Bob converts **each column** (along with its cells' neighbors) into a string: $(X_1 = [X_{1,1}, \cdots, X_{1,\sqrt{n}}], \cdots, X_{\sqrt{n}} = [X_{\sqrt{n},1}, \cdots, X_{\sqrt{n},\sqrt{n}}])$, with each element m bits long. So in all there will be $\sqrt{n}$ strings. Bob then executes the $OT^1_{\sqrt{n}}$Server(X) on each column of the grid and creates $\sqrt{n}$ encrypted columns: $(Y_1 = [Y_{1,1}, \cdots, Y_{1,\sqrt{n}}], \cdots, Y_{\sqrt{n}} = [Y_{\sqrt{n},1}, \cdots, Y_{\sqrt{n},\sqrt{n}}])$. Please note that Bob needs to encrypt each column of the grid with a different set of keys, but the same key is used for encrypting all the cells in a single column. So Bob needs to generate a total of $\log \sqrt{n}$ keys, one for each column. Bob does not engage in the 1-of-2 OT with Alice at this point to give her the keys, but waits until after the PIR step.

(3) User – Alice initiates a query in order to find the neighbor list of the cell she is in (for each bit in the neighbor list). Alice also randomly generates a composite modulus $N = p \cdot p'$ and sends N to Bob (p, p' are large primes). Note that only Alice knows the factorization of N.

(4) Bob sends the granularity of the grid.

(5) Alice chooses the encrypted column containing her cell $M_{a,b}$, e.g., column $Y_b = [Y_1, \cdots, Y_{\lceil\sqrt{n}\rceil}]$, and engages in PIR. Let Z_N^* denote the set of integers that are co-prime with N. Denote the set of quadratic residues (QR's) modulo N using the formula:

$$QR_N = \{y \in Z_N^* | \exists x \in Z_N^* : y = x^2 \bmod N\}$$

Let $PIR(b) = [y_1, ..., y_{\lceil\sqrt{n}\rceil}]$ such that $y_b \in QR_N$ and $\forall j \neq b, y_j \notin QR_N$. Alice issues a PIR request $PIR(b)$ to Bob.

(6) Bob prepares the PIR response: $z = [z_1, \cdots, z_{\lceil\sqrt{n}\rceil}]$ for each row, r of the column b thus:

$$z_r = \Pi_{j=1}^{\lceil\sqrt{n}\rceil}(w_{r,j})$$

where $w_{r,j} = y_j^2$ if $M_{r,j} = 0$, or y_j otherwise. Please note that the PIR request and response are prepared over Bob's **encrypted** database as opposed to the plaintext database as in the EXACT_NN WITH PIR+OT and EXACT_NN WITH PIR+PIR protocols. Bob sends the z vector to Alice.

(7) For each element of the encrypted z vector, Alice checks if each bit i ($i = 1, 2, \ldots, m$) of $z_a[i] \in QR_N$ or $z_a[i] \in QR'_N$ using the formula $(z_a[i]^{\frac{p-1}{2}} = 1 \bmod p) \wedge (z_a[i]^{\frac{p'-1}{2}} = 1 \bmod p')$. If this expression evaluates to true, then it implies that $z_a[i]$ is in QR_N. For each bit, if $z_a[i] \in QR_N$, then Alice concludes that bit is 0, else if $z_a \in QR'_N$, the bit is 1.

At this point, Alice and Bob engage in a 1-of-$\sqrt{n}$ OT over the $\log\sqrt{n}$ keys. Bob converts $\log\sqrt{n}$ keys into a bit string, T and executes $OT^1_{\sqrt{n}}$ Server(T). Alice executes $OT^1_{\sqrt{n}}$ Client(a,E(T)), where $E(T)$ is the encrypted T string. Alice chooses the keys that correspond to the z_a value (more precisely, the keys of the element with index a in z vector). Once she does this, she can get the contents of her cell $M_{a,b}$. From the contents of $M_{a,b}$, she can get the list of neighbors (POIs) associated with $M_{a,b}$. Once Alice knows the set of POI's, she can calculate their individual distances and find her nearest neighbor.

(8) Return the nearest neighbor.

Protocol 3 Exact_NN with PIR+PIR

(1) Server – Bob uses the POIs S to create Voronoi tessellations of the space.

(2) Bob divides the space into an $\sqrt{n} \times \sqrt{n}$ grid M, so the grid is represented by $(M_{1,1}, \cdots, M_{\sqrt{n},\sqrt{n}})$; The neighbor list for each grid cell is prepared. Here, the contents of each grid cell is an m-bit string.

(3) User – Alice initiates a query in order to find the neighbor list of the cell she is in (for each bit in the neighbor list). Alice also randomly generates a composite modulus $N = p \cdot p'$ and sends N to Bob (p, p' are large primes). Note that only Alice knows the factorization of N.

(4) Bob sends the granularity of the grid.

(5) Let Alice be located in $M_{a,b}$. Alice selects the column in which her cell is located, i.e., column b. At this point, Alice can choose to engage in either of two protocols: PIR or $OT^1_{\sqrt{n}}$Client. For this protocol, Alice chooses to engage in PIR. Let Z_N^*

denote the set of integers that are co-prime with N. Denote the set of quadratic residues (QR's) modulo N using the formula:

$$QR_N = \{y \in Z_N^* | \exists x \in Z_N^* : y = x^2 \bmod N\}$$

Let $PIR(b) = [y_1, ..., y_{\lceil\sqrt{n}\rceil}]$ such that $y_b \in QR_N$ and $\forall j \neq b, y_j \notin QR_N$. Alice issues a PIR request $PIR(b)$ to Bob.

(6) Bob computes for each row, r of the matrix, M, the value $z_r = \Pi_{j=1}^{\lceil\sqrt{n}\rceil}(w_{r,j})$ where $w_{r,j} = y_j^2$ if $M_{r,j} = 0$, or y_j, and returns $z = [z_1, ..., z_{\lceil\sqrt{n}\rceil}]$ (please note this process needs to be done for every bit of the contents of the cell).

(7) Alice sends a request $y' = [y'_1, \cdots, y'_{\lceil\sqrt{n}\rceil}]$ with only $y'_a \in QR'_N$. For each bit i of the elements in the vector z denoted by $z_j[i]$

(8) Bob computes the value $z' = \Pi_{j=1}^{\lceil\sqrt{n}\rceil}(w_{j,i})$ where $w_{j,i} = y_j'^2$ if $z_j[i] = 0$, y'_j otherwise. The server returns z' to the client.

(9) Alice recovers z_a from z' (bit by bit) and then $M_{a,b}$ from z_a using a similar approach as described in the EXACT_NN WITH PIR+OT and EXACT_NN WITH OT+PIR. From the contents of $M_{a,b}$, she can get the list of neighbors (POIs) associated with $M_{a,b}$. Once Alice knows the set of POI's, she can calculate their individual distances and find her nearest neighbor.

(10) Return the nearest neighbor.

Sub-protocol 1 $OT^1_{\sqrt{n}}$SERVER(X)

(1) Let $X = [X_1, X_2, \cdots, X_{\sqrt{n}}]$. The client would like to know the contents of one element X_i. Let $l = \log_2 \sqrt{n}$.

(2) The server chooses a symmetric encryption algorithm E_K, which uses key K and generates l random pairs of keys. The pairs of keys are represented by $(K_1^0, K_1^1), (K_2^0, K_2^1), \cdot, (K_l^0, K_l^1)$. So, K_j^b is a key for the encryption algorithm E_K $\forall 1 \leq j \leq l$ and $\forall b \in \{0, 1\}$.

(3) For each $1 \leq i \leq \sqrt{n}$, let $\langle p_1, p_2, ..., p_l \rangle$ represent the bits of i, the server prepares an encryption: $Y_i = X_i \oplus \bigoplus_{j=1}^{l}(E_{K_j^{p_j}}(i))$. Thus, there are a total of $\sqrt{n}$ encryptions.

(4) For all $1 \leq j \leq l$, the server engages in a 1-of-2 OT with the client on the strings $\langle K_j^0, K_j^1 \rangle$.

(5) The server sends the strings $Y = Y_1, \cdots Y_{\sqrt{n}}$ to the client.

Sub-protocol 2 $OT^1_{\sqrt{n}}$CLIENT(i, Y)

(1) The client picks an index i where $\langle p_1, p_2, ..., p_l \rangle$ represents the bits of i to get from the server.

(2) The client engages in a 1-of-2 OT with the server to learn the key $K_j^{p_j}$ for all j $(1 \leq j \leq l)$.

(3) Once the client gets the value of $Y = Y_1, \cdots Y_{\sqrt{n}}$ from the server, it can reconstruct X_i thus: $X_i = Y_i \oplus \bigoplus_{j=1}^{l}(E_{K_j^{p_j}}(i))$.

I have given three protocols using different combinations of PIR and OT: PIR+OT, OT+PIR and PIR+PIR. A possible fourth protocol would be OT+OT. One can implement this option by first having the server encrypt the grid row-wise and then re-encrypting the encrypted grid column-wise. The server and client would then perform two $OT^1_{\sqrt{n}}$'s for the client to get its cell's row and column keys. The communication cost of performing OT+OT is the same as that required for a single-level OT (OT^1_n over the entire database), since, in single-level OT, the server and client need to perform OT^1_2 over $\log n$ key-pairs, while in OT+OT, they need to perform OT^1_2's over two vectors of $\log(\sqrt{n})$ key-pairs (and

$\log n = 2\log(\sqrt{n}))$. The computation cost on the server's side would be a lot more in OT+OT too, since the server needs to encrypt the entire database twice. Hence the efficiency gain (computation and communication-wise) from this would not be as significant as the other 3 two-phase protocols, and the cost of performing OT+OT would be close to, if not more than single-phase OT. Hence I do not pursue the OT+OT approach any further.

I note that the PIR+PIR protocol is quite similar to the recursive PIR protocol given in Kushilevitz and Ostrovsky [57]. The major difference between their protocol and mine is that in the recursive scheme given in [57], if one assumes each element of the $z = [z_1, \cdots, z_{\sqrt{n}}]$ vector to be f-bits long, where f is the length of the PIR modulus N, the client and server engage in f executions of the PIR scheme, once for getting each bit of the element z_a (z_a is the element the client wants). In my scheme, the client gets z_a in one go. Additionally, the PIR+OT and OT+PIR schemes are essentially equivalent to constructing a symmetric PIR scheme where a client gets exactly one element from the server. Kushilevitz and Ostrovsky had suggested a way in which their PIR protocol can be extended to support symmetric PIR, but their solution requires the client and server to execute a zero knowledge proof in which the client proves that it followed the PIR protocol correctly. Such zero knowledge proofs are expensive protocols and should generally be avoided in practice. Using my protocols, one can implement symmetric PIR, but without having to execute a zero knowledge proof. It should be noted that the idea of constructing a symmetric PIR scheme using PIR and 1-of-n OT was also pointed out by Naor and Pinkas [76].

4.4.1. An Example

(1) Alice knows her location in the grid i.e. cell $C4$ which corresponds to $M_{4,3}$, and sends a PIR request to Bob to obtain the contents of her cell: $PIR(C4) = y = [y_1, y_2, y_3, y_4]$. Here $y_1, y_2, y_4 \in QR_N$ and $y_3 \in QR'_N$.

(2) Bob prepares a PIR response, $z = [z_1, z_2, z_3, z_4]$. Here $z_r = \Pi_{j=1}^{[4]}(w_{r,j})$, and $w_{r,j} = y_j^2$ if $M_{r,j} = 0$, or y_j otherwise. So, if $z_4 \in QR_N$, $M_{4,3} = 0$, else $M_{4,3} = 1$.

(3) Alice then initiates a 1-of-4 OT with Bob to obtain z_4 out of z. For doing this, Alice and Bob agree on a symmetric encryption algorithm E which uses key k, and a set of keys, $l = \log_2 \sqrt{n} = 2$. Bob then encrypts each element of z with a different subset of keys. So for each element i in $z = [z_1, z_2, z_3, z_4]$, Bob creates $Y_i = z_i \oplus \bigoplus_{j=1}^{l}(E_{K_j^{p_j}}(i))$ where $\langle p_1, p_2 \rangle$ represents the bits of z_i.

(4) Bob sends the strings $Y_1, \cdots, Y_4$ to Alice.

(5) Alice reconstructs or decrypts the value of z_4 thus: $z_4 = Y_4 \oplus \bigoplus_{j=1}^{l}(E_{K_j^{p_j}}(4))$.

(6) After getting z_4, Alice checks if $z_4 \in QR_N$ or $z_4 \in QR'_N$. Alice can get her location, $M_{4,3}$ thus: If $z_4 \in QR_N$, $M_{4,3} = 0$, else $M_{4,3} = 1$. Here getting $M_{4,3}$ means obtaining the contents of $M_{4,3}$ i.e. the (list of) nearest neighbors of $M_{4,3}$.

(7) Once Alice gets her list of neighbors $\{P3, P4, P4\}$, she can calculate which one of them is her nearest neighbor based on their distance.

4.5. Single-Phase Approaches

In this section, I describe three single-phase approaches for the purpose of comparing performance with the two-phase approaches. The single-phase approaches include a single-phase PIR protocol, a single-phase 1-of-n OT protocol and a "random" OT protocol. In the first case, single-phase PIR, the server database is organized as a grid in exactly the same way and the protocol is the same as given in [25]. In the second case, single-phase OT, the database is organized as a one-dimensional list and the protocol is a simple invocation of OT_n^1Server and OT_n^1Client. These are exactly the same as $OT_{\sqrt{n}}^1$Server and $OT_{\sqrt{n}}^1$Client with all instances of $\sqrt{n}$ replaced by n. In the random OT case, I organize the database as a grid and do a 1-of-k OT over k randomly chosen cells (chosen by the client). The last approach is less expensive than the two-phase approaches or other single-phase approaches but with the caveat that it isn't completely privacy-preserving — the server has a $1/k$ probability of guessing the client's location.

Protocol 4 Exact_NN with PIR

(1) Server - Bob uses the POIs S to create Voronoi tessellations of the space.

(2) Bob arranges the database as a single-dimensional grid with n cells, $X_1, \cdots, X_n$. Here, the contents of each grid cell is an m-bit string.

(3) User - Alice initiates a query in order to find the neighbor list of the cell she is in (for each bit in the neighbor list). Alice also randomly generates a composite modulus $N = p \cdot p'$ and sends N to Bob (p, p' are large primes). Note that only Alice knows the factorization of N.

(4) Bob sends the value of n.

(5) Let Alice be located in X_i. Alice issues a PIR request $[y_1, ..., y_{\lceil n \rceil}]$ to Bob such that $y_i \notin QR_N$ and for all $j \neq i$, and $y_j \in QR_N$.

(6) Bob prepares the PIR response, $z = [z_1, \cdots, z_{\lceil n \rceil}]$ for each bit of the m-bit string. So there will be m responses for each z_i (denoted by $z_i[1], z_i[2], \cdots, z_i[m]$).

(7) From the previous step, Alice gets $z_i[1], z_i[2], \cdots, z_i[m]$, of which Alice can only decrypt z_a. After decrypting z_a Alice checks if each bit i ($i = 1, 2, \ldots, m$) of $z_a[i] \in QR_N$ or $z_a[i] \in QR'_N$ using the formula $(z_a[i]^{\frac{p-1}{2}} = 1 \bmod p) \wedge (z_a[i]^{\frac{p'-1}{2}} = 1 \bmod p')$. For each bit, if $z_a[i] \in QR_N$, then Alice concludes that bit is 0, else if $z_a \in QR'_N$, the bit is 1. Alice repeats this process for each bit of the m-bit string. Once she does this, she can get the contents of her cell X_i. From the contents of X_i, she gets the list of neighbors (POIs) associated with X_i. Once Alice knows the set of POI's, she calculates their individual distances and find her nearest neighbor.

(8) Return the nearest neighbor.

Protocol 5 EXACT_NN WITH OT

(1) Server - Bob uses the POIs S to create Voronoi tessellations of the space.

(2) Bob arranges the database as a single-dimensional grid with n cells, $X_1, \cdots, X_n$. Here, the contents of each grid cell is an m-bit string.

(3) Alice calls OT_n^1CLIENT(i, Y), where i is the index of the cell Alice wants.

(4) Bob executes $OT_n^1 Server(X)$ and prepares the response $Y = [Y_1, \cdots, Y_{\lceil n \rceil}]$ to return to Alice.

(5) Alice computes X_i from $Y = [Y_1, \cdots, Y_{\lceil n \rceil}]$, and from the contents of X_i, gets her list of neighbors and computes the nearest neighbor based on minimum individual distance.

Protocol 6 EXACT_NN WITH RANDOM OT

(1) Server - Bob uses the POIs S to create Voronoi tessellations of the space.

(2) Bob divides the space into an $\sqrt{n} \times \sqrt{n}$ grid M, so the grid is represented by $(M_{1,1}, \cdots, M_{\sqrt{n},\sqrt{n}})$; The neighbor list for each grid cell is prepared. Here, the contents of each grid cell is an m-bit string. Also, Bob converts the $M = \sqrt{n} \times \sqrt{n}$ matrix into a n-element string: $X = X_1, \cdots, X_n$.

(3) User - Alice initiates a query in order to find the neighbor list of the cell she is in (for each bit in the neighbor list).

(4) Bob sends the granularity of the grid.

(5) Let Alice be located in X_i. Alice chooses $k-1$ other random cells in the grid so she has a total of k indices, $i_1, i_2, \ldots, i_k$, where i (Alice's location) is one of the indices.

(6) Alice then runs the $OT_n^k\textsc{Client}(i_1, \cdots, i_k, Y)$ algorithm as given in Naor and Pinkas [76].

(7) Bob runs the $OT_n^k\textsc{Server}(k)$ algorithm as in [76] and returns X_i and its associated nearest neighbors to Alice

(8) Alice computes nearest neighbor from the list of neighbors based on minimum individual distance.

4.6. Definitions, Proof and Complexity

In this section, I define the security properties that are required for the client and server in the two-phase framework and then show that my protocols satisfy them. PIR and OT have been individually proven secure [57, 76], and I do not reproduce the proofs here. I need to prove that my combination of PIR and OT retains the relevant security properties. Additionally, I assume that the OT protocol in [76] uses a secure 1-of-2 OT protocol.

4.6.1. Definition of Privacy Properties

Definition 4.1 gives the formal definition of the desired privacy properties, but these properties also have simple intuitive descriptions. The **Correctness** property states that if the client and server are honest and the client requests the neighbors of a grid cell i, then the client will learn the neighbors of cell i. The **Client's Privacy** property states that the server's view of the transaction when the client requests the neighbors of cell i is computationally indistinguishable from the case where the client requests the neighbors of a different cell i'. The **Server's Privacy** property states that the client's view of the transaction can be completely simulated by a polynomial-time simulator that is given access to just the client's inputs and output, such that the simulated and real executions are computationally indistinguishable. This implies that a corrupted client cannot gain any information that it is not meant to learn.

4.6.1.1. Formal Definition

4.1. Definition (Exact_NN Security Properties). As before, I let $M_{i,j}$ for $1 \leq i, j \leq \sqrt{n}$ be the cells in the matrix M, where each cell contains the list of Voronoi regions that intersect the cell and is exactly m bits long (with padding added if necessary). Let k be a security parameter that determines, among other things, the length of the keys in the cryptographic constructs that are used in my protocols — the size of the input (nm) should also be bounded by a polynomial in k. Let $I = (a, b)$ denote the client's location, so the full input to the Exact_NN computation is $X = (I, M)$, with I being known to the client and M being known to the server. For an Exact_NN protocol Π, let $\text{view}_s^{\Pi}(I, M)$ and $\text{view}_c^{\Pi}(I, M)$ denote the server's and client's views of the protocol, respectively (note that since Π can be randomized, these are actually random variables, or ensembles indexed by (I, M)). Similarly, let $\text{output}_c^{\Pi}(I, M)$ denote the output of the client when running protocol Π (note that while the server interacts with the client, it does not have an "output" or result of its own). Given these definitions, an Exact_NN protocol Π is secure if the following properties hold:

(1) Correctness: For honest client c and honest server s interacting as defined by Π,

$$Pr\left[\text{output}_c^{\Pi}(I, M) \neq M_I\right] \leq \frac{1}{p(k)}$$

for any polynomial $p(k)$ and sufficiently large k.

(2) Client's Privacy: For any PPT server s (not necessarily the honest server specified by protocol Π), there exists a polynomial time simulator S_s such that

$$S_s(M) \overset{c}{\equiv} \text{view}_s^{\Pi}(I, M)\,.$$

Note that since the simulation $S_s(M)$ operates independently of I, it follows that for every $I, I' \in \{1, \ldots, \sqrt{n}\} \times \{1, \ldots, \sqrt{n}\}$,

$$\text{view}_s^{\Pi}(I, M) \overset{c}{\equiv} \text{view}_s^{\Pi}(I', M)\,.$$

(3) Server's Privacy: For any PPT client c (not necessarily the honest client specified by protocol Π), there exists a polynomial time simulator S_c such that

$$S_c(I, M_I) \overset{c}{\equiv} \text{view}_c^{\Pi}(I, M)\,.$$

Note that S is given only c's input I, and the correct output M_I, so there is no information leaked about the other cells of M.

4.6.2. Theorem

As a starting point for my constructions, I assume that I am given secure protocols for PIR and OT — in my protocols I use the PIR protocol due to Kushilevitz and Ostrovsky (KO) [57] and the OT protocol due to Naor and Pinkas (NP) [76]. I need to prove that my combination of PIR and OT preserves the security properties of the original protocols. The two primary security proof models for cryptographic protocols are the simulation-based model and the reduction model. Sequentially composing two proofs in the simulation model is typically straightforward; however, the KO PIR protocol was previously proved secure in the reduction model. Hence, I first convert the PIR reduction proof of KO to a simulation-based proof, and then sequentially compose it with the simulation-based OT proof of Naor and Pinkas. I note here that the security of the PIR+PIR scheme follows directly from the security of the recursive PIR scheme presented in [57] and is not discussed further.

Readers can either refer to Section 3 of the KO paper for the description of the PIR scheme or refer to Chapter 3 of this dissertation. I re-write the KO proof in the simulation model below. Let Bob be the server with inputs $(M_{1,1}, \cdots, M_{\sqrt{n},\sqrt{n}})$ arranged in a $\sqrt{n} \times \sqrt{n}$ matrix, and let Alice be a querying client who wants element $M_{a,b}$.

4.2. Theorem. *The PIR protocol as described in the KO paper preserves the privacy of Alice against a corrupt static, semi-honest PPT server, Bob.*

Proof. The correctness property follows directly from the description of the PIR scheme. For protecting against a corrupt Bob (or for preserving Alice's privacy), I need to construct a simulator for Bob, $B(M_{1,1}, \cdots, M_{\sqrt{n},\sqrt{n}})$ who can simulate Bob's view of the protocol. B first picks two $f/2$-bit primes, multiplies them and gets a f-bit modulus N. Since B knows

the factorization of N, B can easily generate a vector of fake values: $y'_1, \cdots, y'_{\sqrt{n}} \in \mathbb{Z}_N^{+1}$. This is not the same as Alice's input $y_1, \cdots, y_{\sqrt{n}}$ in the real, non-simulated protocol, but is nevertheless computationally indistinguishable from the real y values since N belongs to a **hard set** for which quadratic residuosity predicates are hard to determine. It is easy to see that B can generate the $z_1, \cdots, z_{\sqrt{n}}$ vector which would be indistinguishable from the real Bob's z vector. Hence B can completely simulate the view of Bob.
Since this is PIR, not OT, I do not have to account for a corrupted Alice, since PIR only preserves Alice's privacy, not Bob's. Hence the proof. □

Next follows my main theorem.

4.3. THEOREM. *Using the KO PIR protocol and the NP OT protocol, the PIR+OT and OT+PIR protocols given in Section 4.4 are secure* EXACT_NN *protocols as defined in Definition 4.1.*

PROOF. I consider the individual properties from Definition 4.1 below.

(1) Correctness: Follows directly from the correctness of PIR and OT.
(2) Client's privacy: I construct a simulator for the server (Bob) $S_s(M)$ by combining Bob's simulators for PIR and OT: B_{PIR} from Theorem 4.2 and B_{OT}. If one combines the simulators in the order: (B_{PIR}, B_{OT}), i.e., B_{PIR} goes first, since Alice's contribution to the OT protocol is independent of the result of the PIR, one can simply concatenate B_{PIR} and B_{OT} as the simulation of the PIR+OT protocol (Bob computes the $z_1, \cdots, z_{\sqrt{n}}$ vector, so it can be given as input to the B_{OT} simulator). If one combines the simulators in the order: (B_{OT}, B_{PIR}), i.e., B_{OT} goes first, B_{OT}'s output is $\sqrt{n}$ encrypted vectors, each corresponding to one column of the

grid: $E(y_1), \cdots, E(y_{\sqrt{n}})$. The encrypted vectors are then given to the B_{PIR} simulator which then simulates the rest of the PIR protocol over the encrypted matrix, instead of the plaintext matrix.

Since the outputs of the PIR and OT simulators are individually indistinguishable from the views of the real protocols, and since Alice's contributions to the two protocols are independent, the concatenated simulated view is indistinguishable from the view of the sequentially composed real protocols.

In both cases (PIR+OT and OT+PIR) I have created a simulator for the server's view, and therefore Alice's privacy is preserved in the presence of a corrupted Bob.

(3) Server's privacy: For a client c (i.e., Alice), I need to construct a simulator $S_c(I, M_I)$ that completely simulates Alice's view of the transaction. I assume that I have a simulator A_{OT} that simulates Alice's view of the OT protocol. I do not have a A_{PIR} simulator since PIR, by itself is not secure against a malicious Alice. For (PIR+OT) where PIR is performed first, Bob performs the first half and does not return anything to the real Alice, so S_c does not need to simulate anything. In the second half, S_c can easily simulate Alice's part by running the A_{OT} simulator. If the OT keys are generated using a secure PRF and if a secure OT_2^1 exists, the output of S_c will be computationally indistinguishable from the real Alice's output.

In (OT+PIR), where OT goes first, S_c does not have to do anything for OT, since Bob does not return any result in the first half to the real Alice. S_c needs to simulate Alice's view in the second half - PIR. For this, S_c randomly generates a vector: $y'_1, \cdots, y'_{\sqrt{n}}$ and the corresponding $z'_1, \cdots, z'_{\sqrt{n}}$ vector, which will be fake strings, but indistinguishable from the real y and z vectors because of the quadratic residuosity assumption.

Hence one can construct a simulator for Alice, and Bob's privacy is preserved in the presence of a corrupted Alice.

Hence any combination of PIR and OT are sequentially composable. $\square$

4.6.3. Complexity Discussion

The computation and communication complexity of the single-phase and two-phase protocols are shown in Table 4.1, Table 4.2, Table 4.3, and Table 4.4. In the analysis, I will assume that the location database is organized as a $\sqrt{n} \times \sqrt{n}$ grid with the contents of each grid cell being m-bits long. The length of the PIR modulus is f bits and the length of the keys used in OT are g bits. In the following, I use polynomials $p(f)$ and $p(g)$ to denote the cost of basic computations on f-bit and g-bit values — at worst these are modular powering operations, and so $p(x) = O(x^3)$. The costs of basic operations used in the analysis are given below:

(1) The computation cost of a user preparing a PIR request is $O(p(f)\sqrt{n})$. This is the cost of a user preparing the $y = [y_1, \cdots, y_{\sqrt{n}}]$ vector.

(2) The server computation cost for PIR is $O(p(f)mn)$. This is the cost of the server computing the $z = [z_1, \cdots, z_{\sqrt{n}m}]$ vector.

(3) The computation cost of a user decrypting the z array received from server as the PIR response is $O(p(f)m\sqrt{n})$.

(4) The communication cost of PIR is $O(fm\sqrt{n})$.

(5) The user computation cost for OT is $O(p(g)\log n)$ — in the last step of OT, the user needs to decrypt the server's response $\log n$ times, and g is the length of the keys used.

(6) The server computation cost for OT is $O(p(g)n)$ — the server needs to encrypt the entire database, where g is the length of keys being used.

(7) The communication cost of OT is $O(gn)$, since the server sends the entire encrypted database over to user. Note that there is also the cost of $\log n$ 1-of-2 OT's, but this cost of $O(g\log n)$ is dominated by the database communication cost.

Analysis: The single-phase PIR and single-phase OT analyses, given in Table 4.1 and Table 4.2 are just direct applications of the above costs. For analyzing the computation cost of **random** OT, I replace n cells by k cells. For the communication cost of **random** OT, the main cost is due to sending k encrypted cells from the server, with other communication (initial communication of k cells and the cost of doing 1-of-2 OTs) being insignificant in comparison. For the two-phase protocols, the computation and communications analysis is briefly explained below.

TABLE 4.1. Computational cost for single-phase protocols

Party	PIR	OT	Random OT
User	$O(p(f)m\sqrt{n})$	O$(p(g)\log n)$	$O(p(g)\log k)$
Server	$O(p(f)mn)$	$O(p(g)n)$	$O(p(g)k)$
Total time	$O(p(f)mn)$	$O(p(g)n)$	$O(p(g)k)$

Computation analysis: In the two-phase PIR computation analysis, I consider the server's computation cost for performing PIR twice and user's side computation for preparing and decrypting the server's PIR response. In the PIR+OT computation analysis, I consider

TABLE 4.2. Communication cost for single-phase protocols

PIR	OT	Random OT
$O(fm\sqrt{n})$	$O(gn)$	$O(gk)$

the user's computation cost of preparing the PIR request, server's cost of performing the PIR computation and encrypting the result of the first phase (PIR) using OT, besides generating the key pairs for OT. Also, I consider the user's computation cost for decrypting the final result of the OT that the server sends to the user. For the OT+PIR computation analysis, I consider the server's cost of encrypting the database and generating keys in the first step for OT, and the user's cost for decrypting the final PIR response that the server sends. The computation cost of the two-phase protocols are given in Table 4.3.

TABLE 4.3. Computational cost for two-phase protocols

Party	PIR+PIR	PIR+OT	OT+PIR
User	$O(p(f)m\sqrt{n})$	$O(p(f)m\sqrt{n} + p(g)m\log\sqrt{n})$	$O(p(f)g\sqrt{n})$
Server	$O(p(f)mn)$	$O(p(f)mn + p(g)m\sqrt{n})$	$O(p(g)n + p(f)g\sqrt{n})$
Total time	$O(p(f)mn)$	$O(p(f)mn + p(g)m\sqrt{n})$	$O(p(g)n + p(f)g\sqrt{n})$

Communication analysis: For the communication cost, in the PIR+PIR protocol, I consider the cost of user sending the initial PIR request vector and the server sending back the final PIR response. For the PIR+OT, I consider the user sending the initial PIR vector and the server and user doing the 1-of-2 OT's for the user to get their keys and the cost of the server sending the encrypted OT data to the user. For the OT+PIR analysis, I consider the cost of the user and server performing 1-of-2 OT's for the user to pick their keys and the cost of the server sending the final PIR response to the user. The communication cost of the two-phase protocols are given in Table 4.4

TABLE 4.4. Communication cost for two-phase protocols

PIR+PIR	PIR+OT	OT+PIR
$O(fm\sqrt{n})$	$O(gm\sqrt{n})$	$O(fg\sqrt{n})$

The main advantage of my two-phase scheme over single-phase PIR is that the amount of data revealed by the server is very low. Hence if one has a server that allows the user to query some part of the database, but does not want to reveal the entire database, it can do so more effectively with my protocol than with single-phase PIR (as described in [25] for square grids). It is possible to achieve this with PIR over a single-dimensional list or array of data (not a grid), but that would be too expensive. Additionally, this also reduces the burden on the client/user to sort through redundancies in the data the server sends to find the data element or item that they (client) are looking for. Also, one can use OT over the entire grid and achieve the same level of security, but the cost of doing this is significantly higher than the two-phase approach ($\Theta(gn)$ in single-phase OT vs. $\Theta(g\sqrt{n})$ in two-phase). Random OT has a lower cost than the single and two-phase approaches, but provides a lower degree of privacy: the server has a $1/k$ probability of guessing the client's location.

CHAPTER 5

MULTI-USER, SEMI-HONEST MODEL

This chapter presents the second contribution of this dissertation: a privacy-preserving solution for the multi-user, semi-honest model of LBS. In the following paragraphs, I define the problem and briefly paraphrase prior work before presenting my contributions and methodology.

5.1. Problem Definition

An alternative communication model in LBS other than the client-server model (which was studied in Chapter 4) is the peer-to-peer model where a group of peers would like to co-operatively compute some location without seeking the help of a centralized location-based server. Apart from the obvious shortcomings of the centralized approach such as the server being untrusted, or the server being a communication bottleneck and a single point of failure, the power of user hand-held devices has been increasing with technologies like Bluetooth or IEEE 802.11 being deployed on a range of user hand-held devices. With increased computational power, users may not even need a location-based server or a base station to answer queries and could process queries among themselves. Although there can be many types of spatial queries that a group of peers can compute, in this chapter I focus on **group nearest neighbor** queries in which a group of parties, $P_1, \cdots, P_n$ want to compute their joint group nearest neighbor. In a real-world situation, this could be just a group of users wanting to compute a location that is closest to all of them, e.g., "Which is the restaurant that is closest to all of us where we can meet?" The problem I am focusing on is the situation when the parties want to jointly compute this function but do not want to reveal their individual locations to each other, and work in the absence of a trusted third party. Geometrically, the group nearest neighbor of the group of parties is the minimum of the sum of the distances of each party from each point in a given set of points in the problem space. If there are $P_1, P_2, \cdots, P_n$ parties and $L_1, L_2, \cdots, L_l$ locations, for computing the group nearest neighbor, I firstly compute the sum of the distances of each party from each location: $s_1 = \sum_{i=1}^{n} d(P_i, L_1), s_2 = \sum_{i=1}^{n} d(P_i, L_2), \cdots, s_l = \sum_{i=1}^{n} d(P_i, L_l)$. I then obtain the minimum of all these values: $min(s_1, s_2, \cdots, s_l)$. This value represents the location that is the "group nearest neighbor" of the group of users. In this part, I propose two models for preserving privacy of all peers and experimentally evaluate the performance of them.

5.2. Brief Summary of Related Work

Previous solutions for the group nearest neighbor query computation assume that each party would be willing to share its location with all other parties to compute the group nearest neighbor, or there would be a trusted third party who would be willing to compute the group nearest neighbor, to whom all the parties would reveal their individual locations. Although there has been prior work in the area of group nearest neighbor computation, none of the previous papers have suggested how to provide user privacy if the network is modeled as a peer-to-peer network, in the absence of a trusted third party, and if the peers are untrusted. It should be noted that it is not possible to trivially extend or modify other location-based peer-to-peer protocols for the group nearest neighbor query, such as the ones that use k-anonymity [15, 26], since those protocols assume that peers would be willing to share their locations with each other.

Some of the general work in this direction (non-privacy-preserving) in LBS includes [74, 87] among others. In [87], the authors show how to do query processing of aggregate nearest neighbor queries in road networks. Papadias et al. [74] propose a suite of algorithms for nearest neighbor queries, range queries, distance joins and storage scheme for objects situated on a static network. For a detailed review, readers are referred to Chapter 2.

Secure function evaluation (SFE) among two or more parties is a well-studied problem in cryptography in which a group of parties want to jointly compute a function $f(x_1, x_2, \cdots, x_n)$ where $x_1, x_2, \cdots, x_n$ represent their individual input bits. The goal for each party is to correctly compute the value of $f(\cdot)$ without divulging its own input bit to any other party and in the absence of a trusted third party. Pioneering work in this area was done by Yao [85], Goldreich, Micali and Wigderson (GMW) [28] and Beaver, Micali and Rogaway (BMR) [4]. There are two variants of the SFE problem: two-party/multi-party computation in the presence of semi-honest adversaries (the "semi-honest" model), and two-party/multi-party computation in the presence of malicious adversaries (the "dishonest" model). In the semi-honest model, it is assumed that all the parties involved in the computation will follow the protocol exactly, but will analyze the transcripts of their interactions with other parties to gain extra, un-intended information or in other words, will function as **passive** adversaries. In the dishonest model, it is assumed that a fraction of the parties that are "dishonest" will arbitrarily deviate from the protocol and will act as **active** adversaries. For the most part, in the semi-honest model, the various solutions/protocols developed are based on either Yao's protocol or the GMW paradigm [47]. I model the group nearest neighbor computation problem in the semi-honest model using Yao-style circuits.

5.3. Contributions

My contributions in this chapter can be summarized as follows:

(1) I propose a framework for preserving user privacy in the computation of group nearest neighbor queries in LBS, which is completely peer-to-peer based, does not require the use of a trusted third party and works in the presence of mutually untrusting peers.

(2) I propose two models for the group nearest neighbor query computation: the centralized model and distributed model

(3) I experimentally evaluate the performance in both these models and show that the costs are reasonable and applicable in practice. From my experiments, I find that the distributed model has lower computation costs for processing queries (0-2.1 sec) as compared to the centralized model (0-12 sec), while both models incur the same communication costs.

5.4. Group Nearest Neighbor Queries in the Semi-Honest Model

My methodology is based on the secure function evaluation (SFE) framework and "garbled" circuits are central to solutions proposed for SFE, where the function to be computed is converted into a boolean circuit, encrypted to create a garbled circuit, and finally the garbled circuit is evaluated. Garbled circuits are basically encrypted boolean circuits where the regular boolean values are replaced by encrypted signals. In any garbled circuit, one would have two main parties: a circuit creator and a circuit evaluator - the creator encrypts the circuit gate-by-gate and the evaluator obliviously computes the entire circuit by themselves. At the end of the evaluation process, the evaluator makes publicly known the output of the last gate of the circuit which would be the answer to the function I wish to evaluate. Below I define some notations based on [83, 4, 56] regarding garbled circuits.

5.4.1. Yao-Style Garbled Circuits

5.1. Definition (Garbled Circuit). : Let there be n parties each having an $m-$bit input. I assume that the function which is represented by the boolean circuit outputs a single $m-$bit value. Let the function be represented by a garbled boolean circuit, C consisting of Γ 2-input gates. Let W be the total number of wires where wires $0, \cdots, nm-1$ represent the inputs and $W-m, \cdots, W-1$ represent the outputs; in this description, I use Greek letters to refer to individual wires. Let the plaintext input bit of wire α be denoted by $b_\alpha \in \{0,1\}$. Each wire has a pair of **signals** associated with it: signals are random strings of the size of standard symmetric encryption keys (80, 128 or 256 bits in length) which are numbered in the same order as the wires. So, if α is a wire, $s_{2\alpha}$ and $s_{2\alpha+1}$ are the two signals associated with it. Each signal has a **semantics** variable associated with it which corresponds to the signal's plaintext value; the semantics variable is randomly chosen and is kept secret. For a wire α, a semantics variable $\lambda_\alpha \in \{0,1\}$ is randomly chosen and indicates that $s_{2\alpha}$ has semantics λ_α and $s_{2\alpha+1}$ has semantics $\bar{\lambda_\alpha}$.

The garbled inputs are signals of the input wires that correspond to the actual input bits. The garbled inputs for a wire α would be of the form $\sigma_\alpha = s_{2\alpha+(b_\alpha \oplus \lambda_\alpha)}$ and $\sigma'_\alpha = s_{2\alpha+1+(b_\alpha \oplus \bar{\lambda_\alpha})}$. A set of four **gate labels** are computed by the creator of the circuit. Loosely speaking, the four gate labels for each gate are generated by feeding the signals associated with each wire of the gate to a pseudorandom generator and computing the XOR of the strings generated. The gate labels can be thought of as a truth table (with four values) for gates of 2-input wires. Let $\mathcal{G}$ be a pseudorandom generator that takes as input a k-bit seed and outputs a

$(k + 2nk)$-bit string. I define F, G, and H to be the first k, next nk and last nk bits of $\mathcal{G}$'s output respectively. Let $f_j = F(s_j)$, $g_j = G(s_j)$, $h_j = H(s_j)$, for $0 \leq j \leq W - 1$ (i.e., j ranges over the wire indices). Let $\oplus$ represent the logic function XOR, and let $\odot$ represent the logic function that the gate computes (AND, OR, etc.). If the left and right incoming wires and outgoing wire of a gate i are α, β, and γ respectively, then writing $a = 2\alpha$, $b = 2\beta$, and $c = 2\gamma$, one can compute the gate labels as follows:

$$A_i = g_a \oplus g_b \oplus \begin{cases} s_c & \text{if} \lambda_\alpha \odot \lambda_\beta = \lambda_\gamma \\ s_{c+1} & \text{otherwise} \end{cases}$$

$$B_i = h_a \oplus g_{b+1} \oplus \begin{cases} s_c & \text{if} \lambda_\alpha \odot \bar{\lambda}_\beta = \lambda_\gamma \\ s_{c+1} & \text{otherwise} \end{cases}$$

$$C_i = g_{a+1} \oplus h_b \oplus \begin{cases} s_c & \text{if} \bar{\lambda}_\alpha \odot \lambda_\beta = \lambda_\gamma \\ s_{c+1} & \text{otherwise} \end{cases}$$

$$D_i = h_{a+1} \oplus h_{b+1} \oplus \begin{cases} s_c & \text{if} \bar{\lambda}_\alpha \odot \bar{\lambda}_\beta = \lambda_\gamma \\ s_{c+1} & \text{otherwise} \end{cases}$$

The evaluator of the circuit learns all the above gate labels and the garbled inputs from all players. If the evaluator holds two signal inputs: s_{a+p} for the left incoming wire and s_{b+q} for the right wire, where $p, q \in \{0, 1\}$, the evaluator can easily compute:

$$g_{a+p} \oplus g_{b+q} \oplus A_i \quad \text{if } p = 0 \text{ and } q = 0$$

$$h_{a+p} \oplus g_{b+q} \oplus B_i \quad \text{if } p = 0 \text{ and } q = 1$$

$$g_{a+p} \oplus h_{b+q} \oplus C_i \quad \text{if } p = 1 \text{ and } q = 0$$

$$h_{a+p} \oplus h_{b+q} \oplus D_i \quad \text{if } p = 1 \text{ and } q = 1$$

5.4.2. Centralized and Distributed Models

My goal is to study the application of garbled circuits for answering group nearest neighbor queries in LBS in the semi-honest user model in two settings: centralized and distributed. Let us consider a group of parties $P_1, \cdots, P_n$ who want to jointly compute their group nearest neighbor; I describe two ways to realize this:

(1) Centralized setting: In the centralized setting, a single user, Alice creates the boolean garbled circuit that represents the group nearest neighbor function. For evaluating the circuit one needs another user who can be called an evaluator, Bob. All the other users then just get their encrypted input bits from Alice via oblivious transfer (OT) and send their encrypted input bits to the circuit evaluator Bob, who after getting all the parties' encrypted input bits (including Alice's) evaluates the circuit by himself. At the end of the circuit evaluation process, Bob will make known publicly the output of the last gate of the circuit, which would be the group nearest neighbor of the group of parties. Note that since all the users get their input bits from Alice via OT and then send their encrypted input bits to Bob, neither Alice nor Bob would get to know any user's input bits other than their own. Figure 5.1 represents the centralized model with n users.

(2) Distributed setting: In the distributed setting, the entire region is divided into groups of users and locations, with a pair of users being responsible for creating and evaluating a boolean circuit for a group of locations. If there are n users and l locations, the users can be divided into groups of pairs and one can have up to $n/2$

such groups. The locations can be divided into groups and one can have up to $n/2$ such location groups. In a fully distributed scenario, where all the users actively participate in circuit creation and evaluation, one would have $n/2$ groups of users and $n/2$ groups of locations with $2l/n$ locations in each group. The actual number of groups would be determined by the number of users and the number of locations in each individual case. If the number of locations is greater than or equal to the number of users, one would have $n/2$ location groups and user groups with each user actively participating in circuit creation and evaluation.

On the other hand, if the number of locations is less than the number of users, then there would be a small subset of users who would do the circuit creation and evaluation and most of the other users would just interact with the subset members to get their own input bits. In both cases each group of locations would be managed by 2 users - Alice who acts as a creator and Bob who acts as an evaluator. Alice creates an encrypted circuit for computing the group nearest neighbor of all users among the locations in her group and Bob acts as the evaluator of the circuit. All users have to perform OT with Alice to get their encrypted inputs bits and then transmit their encrypted input bits to Bob. In addition Alice herself transmits her encrypted input bits to Bob. This procedure has to be repeated for all groups of locations which can be done in parallel. By doing this, I divide the time taken for creating and evaluating the circuit among the users besides introducing more parallelism than what was originally there in the protocol. Figure 5.2 represents the distributed model with $n/2$ groups of users.

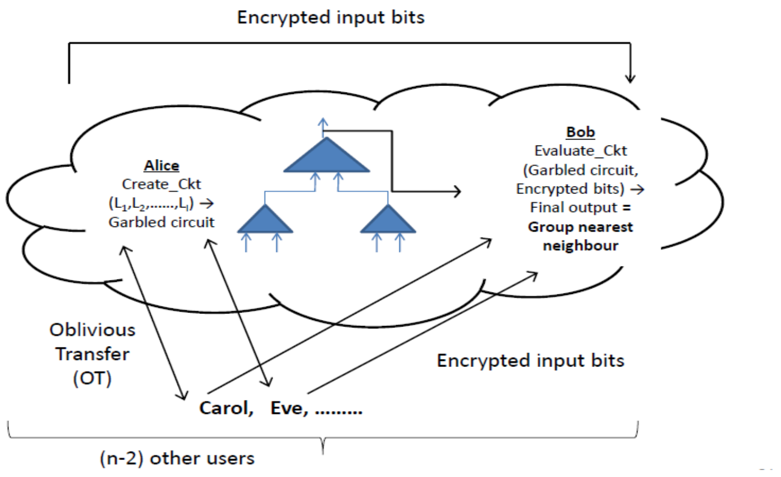


FIGURE 5.1. Illustration of centralized model with $n/2$ users

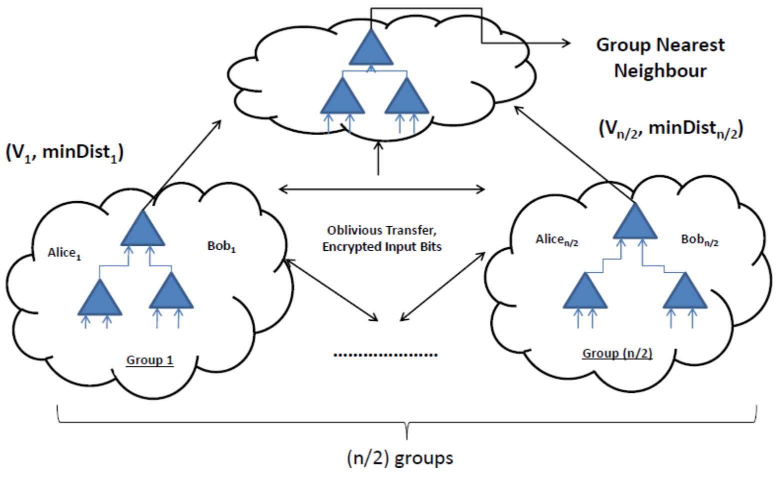


FIGURE 5.2. Illustration of distributed model with $n/2$ users

In the centralized version, the steps of Alice creating a single circuit, users getting their encrypted input bits from Alice via OT, users transmitting their encrypted input bits to Bob and finally Bob evaluating the entire circuit all need to be performed sequentially. In the distributed setting, this can be done in parallel with many small pieces of the circuit which saves time and also reduces the workload on each user.

5.4.3. Group Nearest Neighbor Algorithm

Below I give a high-level overview of an algorithm for my secure group nearest neighbor function evaluation protocol (GNN) in a fully distributed setting. The centralized version is a generalized, simpler version of this. The notation used is the same as given in the garbled circuit definition.

Algorithm GNN_DISTRIBUTED

(1) Initialization: In this step, the n users and l locations are divided into groups and a pair of users are assigned to be responsible for circuit creation (Alice) and evaluation (Bob) for a single group of locations. If one has a set of users/parties $P_1, \cdots, P_n$, arranged in the ascending order of their network ID's, then one can randomly choose one user, P_i and create pairs or groups of users from that user on in a round-robin fashion. For example, after randomly picking user P_i, the groups of users would be created thus: $(P_i, P_{i+1}), (P_{i+2}, P_{i+3}), \cdots, (P_{i-2}, P_{i-1})$. At the end of this step, each group of locations will have 2 users: Alice and Bob who are responsible for creating and evaluating the circuit for computing the group nearest neighbor of all users among this group of locations. Regarding formation of groups, in the case where $l \geq n$, the users are divided into groups of up to $n/2$ users or $(n-1)/2$ users if n is odd. The locations too are divided into $n/2$ or $(n-1)/2$ groups depending on whether n is even or odd with $2l/n$ locations in each group. If $l < n$, the locations are divided into groups with $l \bmod n$ locations each. In this case, one would start forming user pairs/groups starting from a random user P_i and assign a location group to each successive pair until one have exhausted all locations.

(2) Circuit creation: Each group's circuit creator - Alice creates a boolean circuit C, with W wires. Next, Alice creates the signals and semantics for each wire, and finally the garbled inputs and the gate labels for each gate of the circuit. This process is the same as outlined in the garbled circuit description in the above paragraph titled "Garbled Circuit". Note that the circuits of all groups will be created and evaluated in parallel as the circuit creation and evaluation process of each group is independent of the others.

(3) Circuit evaluation: After each group's Alice creates the circuit for her group, Bob - the evaluator of each group evaluates the final output bit of his circuit. Once each sub-circuit is evaluated, the evaluators of all groups will together construct a circuit that computes the location that has the minimum of sum of distances from all their individual output locations, which will be the group nearest neighbor of the entire group.

The proof of security for the centralized and distributed algorithms follows directly from the security of Yao's protocol which was proven secure by Lindell and Pinkas [60]. In [60], Yao's protocol was proven secure with a single circuit creator and evaluator, while in the centralized and distributed algorithms, there are multiple parties participating. But since all the parties, except Alice and Bob are passive participants, not active participants, the same proof holds true in my case under the assumption that a secure 1-of-2 OT exists and the encryption algorithm I am using is semantically secure. It should be noted that in my algorithms, all the passive participants only need to perform a 1-of-2 OT with Alice over their input bits and then send their input bits to Bob who will then evaluate the circuit using encrypted input bits.

5.4.4. An Example

Below I illustrate this process with an example. Let us consider a group of 10 users who wish to compute their group nearest neighbor out of 2000 locations. In a centralized setting, one would just have one user - Alice who converts the group nearest neighbor function into a boolean circuit and another user Bob evaluate the circuit. All the other users would just need to perform an OT with Alice and then send their encrypted input bits to Bob who evaluates the circuit and broadcasts the final answer. In a distributed setting, one would need to form groups of users and locations and the process is a bit more involved than the centralized setting and the various steps of the process are shown below:

(1) In the initialization process, I divide the users into 5 pairs/groups and the locations into 5 groups of 400 locations each. Of the two pairs of users in each user group, I randomly pick one as a circuit creator- Alice and the other as the circuit evaluator - Bob and assign a group of locations to them. Although the first user is picked randomly, for ease of illustration, let us assume that the answer of the random number generator was "User 1" and I start at User number 1. So the first group would be (User 1, User 2). I then take the first group (User 1 and User 2) and assign it to the first group of 400 locations (1-400), the next group (User 3 and User 4) will get assigned the next group of 400 locations (400-800), (User 5 and User 6) would get assigned the next group (800-1200), (User 7 and User 8) would get assigned the next location group (1200-1600) and (User 9, User 10) will get assigned the last group of 400 locations (1600-2000).

(2) Each group's circuit creator, Alice will create a garbled circuit that computes the group nearest neighbor among her assigned group of locations and each group's

circuit evaluator, Bob will evaluate the circuit. Let us call the output values (output values are just a single location) of all the groups $V_1, V_2, \cdots, V_5$ and the minimum of the sum of distances of all parties from the output locations of each group i can be denoted by Dist_i.

(3) The evaluator of all groups, (Bobs') will get together and construct a second-level circuit that computes the minimum of the distances associated with the outputs in Step 2, $\langle (V_1, \mathrm{Dist}_1), (V_2, \mathrm{Dist}_2), \cdots, (V_5, \mathrm{Dist}_5) \rangle$. The circuit outputs the value with the minimum Dist_i, or the group nearest neighbor of the group. For creating the second-level circuit, I randomly choose a creator and evaluator from the group, and all the evaluators of the first-level circuit, i.e. the "Bobs"' will be the participants.

CHAPTER 6

MULTI-USER, DISHONEST MODEL USING TRUSTED COMPUTING TECHNOLOGY

This chapter presents the third contribution of this dissertation: a privacy-preserving solution for the multi-user, dishonest model of LBS. In the following paragraphs, I define the problem and briefly paraphrase prior work before presenting my contributions and methodology.

6.1. Problem Definition

Trusted computing technology, whether in the form of abstract hardware tokens or more concrete components such as the trusted platform module (TPM) is being increasingly used in the construction of various cryptographic protocols, either for implementing functionality that has previously been impossible to realize or for designing protocols with improved efficiency than those known before. The idea of using hardware components in some form in the design of cryptographic protocols can be traced back to Goldreich's Oblivious RAM model and Chaum and Pedersen's e-cash scheme, among others. Of late, there has been renewed interest in this area and there have been a number of recent works that have investigated the usage of hardware components to realize various functionalities such as UC-secure computation, oblivious transfer, and more recently secure computation in a general sense. In this chapter, I consider the problem of designing a secure function evaluation (SFE) protocol using garbled circuits in an efficient way using trusted platform modules (TPMs) which are inexpensive, commonly available chips being deployed in many current systems. In doing so, I consider one of the stronger adversarial models which has been the subject of various impossibility results - the malicious or dishonest adversarial model where any party which is an adversary (or is corrupted by an adversary) can arbitrarily deviate from a given SFE protocol.

It should be noted that it is possible to construct solutions for the dishonest model of SFE without using TPMs, or any other form of trusted computing technology and most or all existing solutions do not use trusted computing. Almost all of the solutions for the multi-party malicious model are based on either Goldreich's compiler [28], or are based on cut-and-choose zero knowledge proofs. The compiler of Goldreich uses generic zero knowledge proofs (ZKPs) which are mainly of theoretical importance and are quite inefficient to apply in practice. Cut-and-choose is a popular paradigm used in interactive ZKPs and while it isn't as expensive as generic zero knowledge proofs, is still a highly interactive construct and significantly increases the communication costs of the protocol using it. In this chapter, I investigate ways in which one can use limited pieces of trusted hardware such as a trusted platform module (TPM) chip to add authentication to existing garbled circuit-based solutions for SFE in an efficient way thus avoiding the use of cut-and-choose zero knowledge proofs.

I then apply my hardware-assisted SFE protocol to LBS with two active parties: one party creating and one party evaluating the circuit, with all other parties passively participating in the SFE protocol. My methodology has the restriction that any one party cannot be a circuit creator and evaluator both.

6.2. Brief Summary of Related Work

One of the emerging areas of work within cryptography that has garnered much attention and investigation recently is the concept of hardware-assisted security. There has been a fair amount of recent research in realizing various cryptographic functionalities using the idea of trusted computing, whether abstract hardware tokens or more concrete instantiations such as TPM chips. The idea of hardware-assisted security can be traced back to the e-cash scheme due to Chaum and Pederson [13] where hardware tokens facilitate secure transactions between customers and a bank and Goldreich and Ostrovsky's oblivious memory model or ORAM [29] that enables software to run in the presence of untrusted memory. This idea was revisited some time back by Katz and Damgård et al. [51, 19] who studied the problem of realizing generic UC-secure two-party computation relying on cryptographic assumptions using tamper-proof stateful hardware tokens where each party constructs its own hardware token. Other recent work in the area of using abstract tokens for realizing cryptographic functionalities include [41, 30, 32]. The closest to my work is a recent paper by Järvinen et al. [48] in which they consider the SFE problem in a two-party setting where a server issues a low-cost tamper-proof hardware token to a client. Clearly at a fundamental level, this idea is similar to mine with the difference that I use a specific piece of hardware - a TPM chip, and do not require cut-and-choose ZKP, which [48] still requires for the malicious model. There has been also been work in the area of instantiating the hardware token as a TPM chip and designing efficient oblivious transfer [40] and verifiable encryption [82] protocols. In the area of location-based services, there have been a few papers that suggest using a TPM chip to design privacy-preserving solutions for the client-server, single-user model [54, 42], but none of these solutions have been implemented. Besides it isn't clear how one would go about implementing them since current TPM chips do not support any LBS-specific functionality. It should be noted that there have been no TPM-assisted solutions proposed for the peer-to-peer model. For a more detailed review of related work, readers are referred to Chapter 2.

6.3. Contributions

I show how limited trusted hardware such as trusted platform module chips (TPMs) can be extended to support SFE efficiently in the presence of malicious adversaries and can be used as an alternate solution to expensive cut-and-choose zero knowledge proofs which might be infeasible to implement in practical applications that use SFE by replacing the statistical trust gained from ZKPs with a degree of trust in tamper-evident hardware. In my methodology, for supporting garbled circuits, I require a few extensions to the current TPM v.1.2 specification: firstly, one requires a keyed HMAC-PRF which is a minor extension since TPMs already support HMACs and this would just be a matter of defining a key type to use as a seed for the HMAC; the second extension is a more significant one where one requires a set of new TPM commands: TPM_SFEInit, TPM_SFEInputs, TPM_SFEGate, and TPM_SFEFinish. I believe that these extensions are worth considering since they consist of operations already supported by the TPM, are simple to implement and in a general sense, quite a few applications would potentially benefit from them (secure function evaluation has uses in a variety of applications ranging from auctions to medical diagnostics to data mining). Finally, I apply my TPM-assisted SFE methodology to LBS with one circuit creator, one circuit evaluator and multiple passive participants. It should be noted that my methodology is of independent cryptographic interest and can be used in any application that uses SFE and not just LBS.

6.4. Hardware-Assisted Secure Function Evaluation Using Garbled Circuits

I consider a scenario where there are n participants who want to compute a function $f(x_1, x_2, \cdots, x_n)$ over their individual inputs $x_1, x_2, \cdots, x_n$ without any party revealing its input to other parties and in the absence of a trusted third party; for simplifying the discussion, I set $n = 2$. Since I consider the malicious adversarial model, one cannot trust Alice to create the circuit correctly. In solutions that have been proposed for the malicious model up until now, Alice would have had to provide a zero-knowledge proof at every stage of the circuit construction to other parties proving that she has constructed the circuit correctly which is the part I replace with trust in secure hardware, or more specifically TPM chips. I use a slightly modified version of Kolesnikov and Schneider's circuit construction technique [56] for my circuit construction. Details of the Kolesnikov-Schneider construction are given in Chapter 2.

Let there be $n = 2$ parties each having m-bit inputs; let the output of the circuit be m-bits. Let the plaintext truth table that defines a gate g_i implementing functionality $\otimes$, with input wire indices, a, b and output wire index c be denoted by TT_{g_i}, which consists of $2^{\text{Num. of input wires}}$ entries - in this case 4 entries. $TT_{g_i} = \langle(0, 0, 0 \otimes 0), (1, 0, 1 \otimes 0), (0, 1, 0 \otimes 1), (1, 1, 1 \otimes 1)\rangle$. Let the circuit have W wires with $0, \cdots, nm - 1$ being input wires and $W - m, \cdots, W - 1$ being output wires. At one time, the TPM outputs one set of garbled truth tables. For doing this, one needs a few simple extensions to the existing TPM specification to support the garbled circuit creation, something along the lines of an "SFE suite" of commands. The following are the proposed TPM extensions for performing SFE using garbled circuits:

`TPM_SFEInit`$(gateNum, inputNum) \rightarrow (cktHandle)$**:** : Alice's TPM takes in the number of gates and input wires in the unencrypted boolean circuit, internally stores them and defines a handle to the internal protected storage kept aside for this particular circuit. The TPM creates two internal variables: **count** and **inputCount** for keeping track of the number of gates and input wires, both of which it sets to 0 initially; count will be incremented with each gate until $count = gateNum$ and inputCount will be incremented with each input wire until $inputCount = inputNum$. The TPM then generates a secret key which will only be known to the TPM and uses this key as a seed, called SFESeed, to an HMAC which can be used as a Pseudo Random Function (PRF) (it is known that HMAC is a good candidate for instantiating a PRF under the assumption that its compression function is a PRF [6]). Additionally, the TPM creates a random string, $R \in \{0,1\}^N$ which will be used in the TPM_SFEGate command. R needs to be generated in a way such that its least significant bit (lsb) is 1 - I'll explain how this is useful in the TPM_SFEGate command. Finally, the TPM generates two digests, one for the plaintext truth tables, *hashPlain* and one for the encrypted truth tables, *hashEncrypted* which are both initially set to be null hashes. All of the above created variables are stored in an internal TPM structure called TPM_SFE_DATA which the TPM references in future SFE commands for this particular circuit:

```
TPM_SFE_DATA{
hashPlain,hashEncrypted,
count,gateNum,inputNum,inputCount,
SFESeed,R
}
```

As the output of this command, the TPM returns the handle pointing to the internally stored TPM_SFE_DATA structure.

`TPM_SFEInputs`$(a, cktHandle) \rightarrow (k_a^0, k_a^1)$: :

In this command, Alice's TPM takes in the input wires of the circuit one at a time and outputs the garbled values of each input wire. Alice can store these garbled inputs in an array, e.g., $inputValues : [k_0^0, k_0^1, \cdots, k_{nm-1}^0, k_{nm-1}^1]$. For keeping track of the number of input wires, the TPM uses the internal variable called $inputCount$ which was created and initialized to 0 in the TPM_SFEInit command. The TPM increments $inputCount$ internally every time the TPM_SFEInputs command is invoked until $inputCount = inputNum$. The TPM next generates and outputs the garbled inputs thus: $k_a^0 \leftarrow HMAC(SFESeed, a)$, set $k_a^1 \leftarrow k_a^0 \oplus R$ for input wire a. Since Alice can infer the value of R from any set of garbled inputs, one requires that Alice cannot be a circuit creator and evaluator both. This command needs to be called nm times by Alice; in the end, $inputValues$ will have nm entries in it. At this point, there are two ways for Alice to handle the input values and send them to Bob:

(1) Option 1: Alice performs a 1-of-2 OT for each of Bob's garbled values in the $inputValues$ array. In total, Alice needs to do m 1-of-2 OT's with Bob for Bob to get his m input values. Note that if one has $n \geq 2$ parties, where the remaining parties other than Alice and Bob are passive participants, then they have to get their garbled input bits from Alice via 1-of-2 OT's each and send

their garbled inputs to Bob for evaluation. Bob, after receiving inputs from all parties evaluates the circuit.

(2) Option 2: This option can be chosen if one has multiple circuit evaluators (anyone can be an evaluator except Alice). Bob can either create m one-time use **count-limited objects** or **clobs** each of the form $C = (countLimit, PK, SK)$ and send the m **clobs** to Alice's TPM, or Alice and Bob can engage in a non-interactive oblivious transfer as in [40] using just one **clob**. If Bob is using m-one-time use **clobs**, each **clob** will be used to decrypt one of his input bits. If the TPM has the infrastructure for supporting multiple **clobs**, Bob and Alice can do a non-interactive OT over m inputs with a single **clob**. Again, if there are $n \geq 2$ parties where all parties other than Alice and Bob are passive participants, then they need to either create m one-time use **clobs** each or do a non-interactive OT over m inputs with a single **clob** each.

`TPM_SFEGate`$(a, b, c, cktHandle, g_i, TT_{g_i}, flag) \rightarrow (ET_{g_i}, (e_0, e_1))$**:** :

(1) This command takes in the circuit handle, a string representing a gate ID g_i, the input wire indices, a and b of g_i, the plaintext truth table, TT_{g_i} and a $flag$ for the output wires and returns as output the encrypted truth table and the output wire encodings in case the output of this gate is an output wire of the circuit.

Firstly, the TPM increments the value of $count$ and then generates the keys to be used in the construction of g_i's garbled values thus: $k_a^0 \leftarrow HMAC(SFESeed, a)$, set $k_a^1 \leftarrow k_a^0 \oplus R$ and $k_b^0 \leftarrow HMAC(SFESeed, b)$, set $k_b^1 \leftarrow k_b^0 \oplus R$ for the 2 input wires a and b. The $k_a^0, k_a^1, k_b^0, k_b^1$ values are stored internally and are used in construction of the truth tables. Since the lsb of R is 1, the values of k_i^0 and k_i^1 for any wire i would always be different.

(2) Next, the TPM needs to create truth tables for the gate. First it generates: $k_c^0 \leftarrow HMAC(SFESeed, c_0)$ and $k_c^1 \leftarrow \langle k_c^0 \oplus R \rangle$. the TPM then creates the garbled truth table with 4 entries as follows:

$$e_{0,0} = H(k_a^0||k_b^0||i) \oplus k_c^{0\otimes 0} \quad \text{in position } (p_a, p_b)$$

$$e_{1,0} = H(k_a^1||k_b^0||i) \oplus k_c^{1\otimes 0} \quad \text{in position } (\bar{p_a}, p_b)$$

$$e_{0,1} = H(k_a^0||k_b^1||i) \oplus k_c^{0\otimes 1} \quad \text{in position } (p_a, \bar{p_b})$$

$$e_{1,1} = H(k_a^1||k_b^1||i) \oplus k_c^{1\otimes 1} \quad \text{in position } (\bar{p_a}, \bar{p_b})$$

where p_a = lsb of k_a^0, p_b = lsb of k_b^0 and p_c = lsb of k_c^0 (they aren't separate variables as in the original construction [56]). The TPM then stores all the four table entries along with the gate ID in internal storage. These values will be needed for computing the hash, after which these values can be erased from internal storage. I'll refer to the encrypted truth table of gate g_i which consists of the four truth table entries collectively as ET_{g_i}.

(3) The TPM checks if $(flag == true)$. The $flag$ is used for keeping track of the output wires. If $(flag == true)$, it means that this gate is an output gate and c being the output wire of this gate, the TPM needs to output the values of the output wire encodings. The TPM computes:

$$e_0 = H(k_c^0||\text{“}out\text{”}||g_i) \oplus 0 \text{ in position } p_c$$

$$e_1 = H((k_c^0 \oplus R)||\text{“}out\text{”}||g_i) \oplus 1 \text{ in position } \bar{p_c}$$

where p_c is the lsb of k_c^0 (not a separate variable). In case $(flag == false)$, the TPM just returns $\perp$ in place of the e_0, e_1 values.

(4) Lastly, the TPM updates the plain and encrypted truth table digests in TPM_SFE_DATA:

$$hashEncrypted = H(hashEncrypted \parallel g_i \parallel H(ET_{g_i}))$$

$$hashPlain = H(hashPlain \parallel g_i \parallel H(TT_{g_i}))$$

and stores them in TPM_SFE_DATA. These are running hashes of all the gates of the circuit as the TPM processes them one-by-one (this operation is similar to the TPM_PCREXTEND operation present in current v.1.2 TPMs).

(5) The TPM outputs all 4 entries in the truth table and the output wire garbled values.

`TPM_SFEFinish`$(cktHandle) \rightarrow (Sign_{AIK}(hashPlain, hashEncrypted))$: : This command is called after the TPM_SFEGATE command has been executed for each gate.

Firstly, the TPM takes in the circuit handle and checks if $(count == gateNum)$ ($count$ is incremented for each gate in the TPM_SFEGATE command). If no, the TPM returns an error code, if yes, the TPM knows that it has processed the last gate of the circuit and there are no more gates and the TPM outputs the signed plain and encrypted hashes, $Sign_{AIK}(hashPlain, hashEncrypted)$.

Below is an algorithm that describes the actions of the untrusted circuit creator whose TPM creates and authenticates the circuit. Let GC_List be a linked list used to store the gates of the circuit in topological order. Let n be the number of parties with m-bit inputs each (m is also the number of output wires), TT_{g_i} and ET_{g_i} are the plaintext and encrypted truth table of a gate g_i respectively, $(W-m), \cdots, (W-1)$ are the output wires and $C_1, \cdots, C_m$ are Bob's m one-time use **clobs** for getting his input values.

Algorithm GCCREATOR $(gateNum)$

```
    TPM_SFEInit$(gateNum, inputNum) \rightarrow (cktHandle)$
    for$(i = 0; i < m - 1; i++)$
        TPM_SFEInputs$(i) \rightarrow \langle (k_i^0, k_i^1, E_{PK_i}(k_i^0), E_{PK_i}(k_i^1), \mathrm{Sign}_{AIK}(C_i)) \rangle$
        Store the garbled input values of $i$ in $inputValues[\cdots]$ array
    for$(i = 0; i < gateNum; i++)$
        Let $a, b$ be wire numbers for the inputs of gate $i$ and $c$ be the output wire
        if $i$ is an output gate, set $flag = true$
        TPM_SFEGate $(a, b, c, cktHandle, g_i, TT_{g_i}, flag \rightarrow \langle ET_{g_i}, (e_0, e_1) \rangle$
    TPM_SFEFinish$(cktHandle) \rightarrow \langle Sign_{AIK}(hashPlain, hashEncrypted) \rangle$
    for$(i = 0; i < gateNum; i++)$
        GC_List.append$\langle ET_{g_i}, g_i, TT_{g_i} \rangle$
    return $\mathcal{S} = \langle Sign_{AIK}(hashPlain, hashEncrypted), GC\_List, (e_{W-m}^0, \cdots, e_{W-1}^1)$
    $Sign_{AIK}(C_1, \cdots, C_m), inputValues \rangle$
```

In the GCEvaluator procedure, the evaluator first computes the plain and encrypted truth table hashes by itself over the entire circuit with the information the GCCreator has provided and then compares its hashes with the hashes the TPM has signed and returned - $\mathrm{Sign}_{AIK}(hashPlain, hashEncrypted)$. If they do not match, the evaluator aborts the protocol.

```
Algorithm GCEvaluator $(\mathcal{S})$
    if $hashPlain, hashEncrypted$ have not been signed with a valid AIK, return $\perp$
    if $C_1, \cdots, C_m$ has not been signed with a valid AIK, return $\perp$
    for$(i = 0; i < gateNum; i++)$
```

```
        Get plain and garbled truth tables from GC_List and compute $hashPlain, hashEncrypted$
        **if** computed hashes do not match with TPM-signed hashes, **return** $\perp$
    Get correct input wire garblings from the clobs thus:
    **for**$(i = 0; i < m - 1; i++)$
        $Decrypt_{SK}(inputValues[i])$
    **for**$(i = 0; i < gateNum; i++)$
        Decrypt the correct truth table entry from position $(p_a, p_b) : e_{a,b} = k_c = H(k_a, k_b, g_i) \oplus e$
    **for**$(i = W - m; i < W - 1; i++)$
        Decrypt output value from output table entry from position $p_c$ thus:
        $f_i = H(k_i || \text{"}out\text{"} || g_i) \oplus e$
    **return** $f_{W-m}, \cdots, f_{W-1}$
```

Protocol HASFE_GC: Let Alice's inputs be $x = (x_1, \cdots, x_m) \in \{0,1\}^m$ and Bob's inputs be $y = (y_1, \cdots, y_m) \in \{0,1\}^m$. Let their common inputs be a plaintext circuit, $C(x, y) = f(x, y)$ where $f : \{0,1\}^m \times \{0,1\}^m \rightarrow \{0,1\}^m$ is the function that they wish to evaluate. The following is a hardware-assisted SFE protocol for securely computing f.

(1) Alice constructs garbled circuit $C'(x, y) = f(x, y)$ using **Algorithm** GCCreator.

(2) Alice sends her garbled input wires: $(k_{x_1}, \cdots, k_{x_m})$ to Bob and also sends the encrypted truth tables to Bob.

(3) Alice sends Bob his garbled inputs in one of two ways:

   (a) Alice and Bob engage in a 1-of-2 OT over $(k_{y_1} \cdots, k_{y_m})$

(b) Bob either generates m one-time use **clobs**, or generates a single m-time use **clob** of the form: $\alpha = (PK, SK, countLimit)$, as the case might be, and sends them to Alice who sends Bob his encrypted inputs.

(4) Bob runs **Algorithm** GCEvaluator and outputs $f(x, y)$ - the output wires of the circuit.

6.5. Theorem

I first prove that the HASFE_GC protocol is secure in the presence of a semi-honest Alice and semi-honest Bob and then prove that if Alice and Bob are malicious, the proof still holds. I assume, according to Bellare's paper [6] that HMAC is a PRF if its compression function is a PRF.

6.1. Theorem. *If one uses a TPM that satisfies the Trusted Platform Security Assumption, and assuming that HMAC can be used to instantiate a PRF, the HASFE_GC protocol is secure in the presence of malicious PPT adversaries.*

Proof. There are two parts to the proof: One is the case where the circuit creator is malicious and the other is the case where the circuit evaluator is malicious. I assume that I use a secure OT^1_2 (1-of-2 OT) protocol and there exists a OT^1_2 simulator for Alice and Bob. I first prove the protocol secure in the semi-honest model and then show how to extend the proof to the malicious model.

Case 1 - Alice is corrupted: Let there be a simulator A that simulates Alice's view of the protocol. A needs to produce Alice's output given the input $A(x, f(x, y))$. A simulates the garbled circuit in the same way as Alice and outputs the garbled input values and encrypted truth tables. A then runs the OT^1_2 simulator with fake garbled inputs corresponding to Bob and outputs the transcript of the OT_2^1. It is easy to see that the output of Alice and A are computationally indistinguishable from each other.

Case 2 - Bob is corrupted: In Kolesnikov and Schneider's paper [56], the garbled circuit evaluation was proven secure with respect to a semi-honest Bob using **truly random strings**. In this proof, I need to do the same using a pseudo-random function (PRF) generated using a HMAC instead of truly random strings. So, I need to prove that if a corrupted Bob cannot gain any advantage against an Alice who uses truly random strings, he cannot gain any advantage against an Alice who uses PRF-strings generated by her TPM either.
Let there be two simulators for Bob: B and B' such that B works with HMAC-PRFs and B' is given access to a true random oracle. I show that one cannot construct a polynomial-time distinguisher D that can consistently distinguish with non-negligible probability between the outputs of B and B'. I describe B and B' below:
Simulator B: Let B be a simulator that simulates Bob's view of the protocol, B needs to produce Bob's output given the input $B(y, f(x, y), \mathcal{HPRF})$ where $\mathcal{HPRF}$ is a HMAC-PRF that generates pseudo-random strings. B has access to Bob's input, but the real Bob gets the garbled circuit from Alice as well, which B does not receive as input. So, B needs to simulate the entire garbled circuit with garbled inputs and garbled truth tables as received by Bob. For simulating the circuit, B generates a random string $R' \in \{0,1\}^N$ and generates the garbled inputs thus: $(k_1^0 \leftarrow \mathcal{HPRF}(1), k_1^1 \leftarrow k_1^0 \oplus R', \cdots, k_{nm}^0 \leftarrow \mathcal{HPRF}(nm), k_{nm}^1 \leftarrow k_{nm}^0 \oplus R')$. It also generates fake truth tables thus: $e'_{00} = H(k_a^0 \parallel k_b^0 \parallel i) \oplus k_c^{0 \otimes 0}$, and so one for the other three table entries.

The table entries are easily placed in position based on the lsb's of the input wires. B then creates fake output tables using the same procedure as in Kolesnikov and Schneider's proof [56], i.e., for each circuit output wire, W_i, create a fake garbled output table with both entries corresponding to the same output garbled value:

$$e'_0 = H(k'^0_c || \text{“}out\text{”} || g_i) \oplus f_i(x,y)$$
$$e'_1 = H((k'^0_c \oplus R') || \text{“}out\text{”} || g_i) \oplus f_i(x,y)$$

Finally B outputs $Sign_{AIK}(hashPlain, hashEncrypted)$ - the digests of the plaintext and encrypted circuits. Here I require that B should be able to simulate both, the TPM's AIK and the PrivacyCA certifying the AIK, since Definition 3.1 stipulates that TPM signatures are existentially unforgeable.

For simulating the transfer of garbled input values, B either runs the OT^1_2 simulator with its inputs $(y_1, \cdots, y_m)$ and outputs the transcript of the OT^1_2, or uses a **clob** as the case might be. If B needs to use a **clob**, and simulate the TPM's aignature on it, B needs to simulate both, the TPM's AIK and the PrivacyCA that certifies the AIK. In the end, B returns the signed hashes, the fake tables, inputs, and fake output tables.

Hence B, with access to a HMAC-PRF, can correctly simulate the view of Bob such that the output of B is computationally indistinguishable from that of Bob.

Simulator B': Let B' be a simulator that simulates Bob's view of the protocol, B' needs to produce Bob's output given the input $B'(y, f(x,y), \mathcal{RO})$ where $\mathcal{RO}$ is a random oracle that generates true random strings. B' needs to simulate the entire garbled circuit as received by Bob. For simulating the circuit, B' generates a random string R'' and generates the garbled inputs thus: $(k^0_1 \leftarrow \mathcal{RO}(1), k^1_1 \leftarrow k^0_1 \oplus R'', \cdots, k^0_{nm} \leftarrow \mathcal{RO}(nm), k^1_{nm} \leftarrow k^0_{nm} \oplus R'')$ and the garbled tables too, just like B does. For simulating the transfer of garbled input values, B' either runs the OT^1_2 simulator with the inputs it is given $(y_1, \cdots, y_m)$ and outputs the transcript of the OT_2^1, or if it needs to use a **clob**, B' generates the **clob(s)** and encrypts the garbled input values in the same way as B did.

Hence B', with access to a random oracle, can correctly simulate the view of Bob such that the output of B' is computationally indistinguishable from that of Bob.

Assume that one has a polynomial-time distinguisher D who can distinguish between the output of B and the output of B'. The following is the standard PRF-game played by D:

Game PRF

(1) D is given access to a PRF-oracle: $\mathcal{PO} \in \{\mathcal{PRF}, \mathcal{RO}\}$ where the $\mathcal{PRF}$ is generated from a HMAC. D asks $\mathcal{PO}$ to output a set of garbled inputs.

(2) If $\mathcal{PO} = \mathcal{PRF}$, it constructs the garbled inputs in a way similar to B, if $\mathcal{PO} = \mathcal{RO}$, it constructs the garbled inputs in a way similar to B'.

(3) D examines the garbled inputs and outputs a guess g

D wins the game if it can distinguish between the case when $\mathcal{PO} = \mathcal{PRF}$ and the case when $\mathcal{PO} = \mathcal{RO}$. The advantage of D winning the PRF game can be bounded as:

$$Adv_{PRF}(D) \leq Adv_{HMAC}(D)$$

Since I assume, according to Bellare's paper [6] that a HMAC is a PRF it its compression function is a PRF, for every positive polynomial $p(\cdot)$ and all sufficiently large n's, the advantage of D winning the PRF game can be bounded as:

$$Adv_{PRF}(D) = Pr_{g \leftarrow \mathcal{PRF}}[D^g = 1] - Pr_{g \leftarrow \mathcal{RO}}[D^g = 1] \ < \frac{1}{p(n)}$$

where the probabilities are over the choices of g and the coin tosses of D. If such a D exists, it can distinguish between B and B' in the circuit construction process, since B uses $\mathcal{PRF}$ to create the garbled circuit and B' uses $\mathcal{RO}$ to create the garbled circuit. The advantage of D when given a random member of the set $\{B, B'\}$ is:

$$Adv_D(B, B') \leq Adv_{PRF}(D)$$

Since the $Adv_{PRF}(D)$ is negligible in the PRF-game, $Adv_D(B, B')$ is also negligible. Hence B and B' are computationally indistinguishable from each other. From Kolesnikov and Schneider's construction [56], B' which uses true random strings in the circuit construction process is secure in the semi-honest model. Since B and B' are computationally indistinguishable, B, which uses a HMAC-PRF in place of a true random oracle is also secure in the semi-honest model.

Extending the result to the malicious model: In the malicious model, the main adversary to protect against is a malicious Alice. Traditionally, Alice does a cut-and-choose zero knowledge proof to prove that the circuit inputs were suitably randomized and they were used in generation of the truth tables, etc. Here, Alice's circuit is completely constructed by her TPM. I had shown in Case 1 that a semi-honest but corrupted Alice can be protected against (there exists a simulator A, which on inputs $A(x, f(x, y))$ can produce Alice's view of the protocol). The only thing that a malicious Alice can do that a semi-honest Alice cannot do is to present a circuit to Bob that **appears** to have been certified by her TPM, but in reality the TPM does not certify it. The only way for a malicious Alice to do this is by forging the TPM's AIK signature over her encrypted circuit:

$$Adv_{HASFE}(Alice) \leq Adv_{AIK}(Alice)$$

From Definition 3.1, the signature algorithm used by the TPM is existentially unforgeable, so $Adv_{AIK}(Alice)$ is negligible and hence $Adv_{HASFE}(Alice)$ is also negligible.
Bob can be malicious, but since I am considering the model where there is a single circuit creator and no circuit creator can also be an evaluator, a malicious Bob cannot do anything that he could not have done in the semi-honest model. Hence the proof. □

6.6. Homomorphic Encryption

Another approach to doing SFE is homomorphic encryption; homomorphic encryption-based SFE solutions have been used in secure auctions [17] and secure mobile agents [78]. The two kinds of homomorphic encryption schemes are additively homomorphic encryption schemes and multiplicatively homomorphic encryption schemes. If one has two plaintext messages, m_1, m_2 and an encryption algorithm E, a homomorphic function is represented by: $E(m_1(+\,\mathrm{or}\,*)m_2) = f(E(m_1), E(m_2))$ where $f(\cdot)$ is an efficiently computable function that combines ciphertexts. In previous papers, $f(\cdot)$ was computed in standard models, in this paper I consider the situation where one is allowed to use a TPM to compute $f(\cdot)$. Some of the well-known asymmetric ciphers such as un-padded RSA [77], Paillier's cryptosystem, etc. have inherent homomorphic properties, each with its own set of drawbacks with respect to the hardware-assisted security model (summarized in Table 6.1). Craig Gentry [23] recently proposed a scheme that is additively and multiplicatively homomorphic, i.e., fully or algebraically homomorphic, but uses lattice-based cryptography and is computationally intensive. There have been variants of Gentry's scheme since then [80, 20] which do not use lattice-based cryptography and are based on simpler constructs, but are still computationally intensive and require large or unbounded amounts of storage. Although current TPM chips do not support ElGamal, ElGamal (which has plaintext size $\mathbb{Z}_n$ and ciphertext size $\mathbb{Z}_n^*$) seems to be the best option for implementing a fully homomorphic cryptosystem on TPM chips efficiently. I have briefly explored this idea by creating TPM commands and algorithms for the circuit creator and evaluator, but have not implemented them as yet. Hence I leave this as part of future work.

TABLE 6.1. Homomorphic cryptosystems

Homomorphic Cryptosystem	Unsuitable features for TPMs	P	C
Paillier [72]	Works only over $\mathbb{Z}_{n^2}^*$	$\mathbb{Z}_n$	$\mathbb{Z}_{n^2}^*$
DJ [18]	Works only over $\mathbb{Z}_{n^2}^*$	$\mathbb{Z}_{n^s}$	$\mathbb{Z}_{n^{s+1}}^*$
DGK [17]	Decryption needs unbounded space	$\mathbb{Z}_x$	$\mathbb{Z}_n^*$
Benaloh [7]	Decryption needs unbounded space	$\mathbb{Z}_m$	$\mathbb{Z}_n^*$
RSA [77]	Can't be CPA-secure and homomorphic, both	$\mathbb{Z}_n$	$\mathbb{Z}_n^*$
GM [31]	Expensive, plaintext gets encrypted bit-by-bit	$\mathbb{Z}_2$	$\mathbb{Z}_n^*$

6.7. Application to LBS

I apply the garbled circuit-based, TPM-assisted SFE protocol to LBS in the model where there is one circuit creator, one circuit evaluator and multiple passive participants. The circuit remains the same as in Chapter 5 (the semi-honest model), with the difference that the TPM creates the entire circuit. As mentioned in Section 6.1, my methodology has the restriction that any party cannot be a circuit creator and circuit evaluator at the same time.

CHAPTER 7

EXPERIMENTS

In this chapter I describe experimental results for the solutions presented for the three main problems considered in this dissertation. I first describe the results of implementing the two-phase framework and single-phase framework for the single-user, client-server model. Next, I present the results of implementing the centralized and distributed models using Yao's protocol [85]. Finally, I present the results of implementing the TPM command extensions on a TPM simulator and the results of implementing the SFE protocol with the TPM and Flicker [62]. It should be noted that the TPM v.1.2 does not support the commands I have proposed in the dissertation and hence one needs to implement them on the simulator.

7.1. Single-User, Client-Server Model

I used Java to implement the two-phase and single-phase frameworks and used the **Stopwatch** library to instrument timing measurements in the code; the numbers given in the experiments are all actual execution times. For measuring computation time at the server and user, I averaged the time taken over 100 queries. I needed two datasets to use in my experiments: a user location dataset and a Points of Interest (POIs) dataset for which I used a set of spatial datasets provided by Li et al. [58] which is a real dataset of a road network and POIs in California. The number of user locations in my experiments range from 0-100K. For generating user locations, I partially used the California road network dataset [58] which contains 21,048 locations and the rest were randomly generated by us. Each user location corresponds to a grid cell. The POIs in my experiments range from 0-100K for which I used the California POI dataset provided in [58] which contains 104,770 POIs. The dataset I used contains POIs from 62 categories such as school, church, airport, beach, etc. In the experiments, for computing the user's nearest neighbor, I measured the Euclidean distance from the user locations to the POIs and considered all POIs in a single category. In particular, in the set of experiments where one keeps the number of POIs constant and vary the number of grid cells, I set an upper limit on the number of POIs (1000). The 1000 POIs can be taken from any category as long as they are closest to the user's location. This can easily be extended to a more restricted case where the user's neighbors are from a specific category of POIs by reducing the limit on the POIs returned by the server and choosing POIs only from a specific category. For the restricted case, I estimate the costs will be less than the ones reported in this work.

Since PIR requires large integers, I used the Java **BigInteger** data type in the PIR implementation. I measured the performance of the three protocols in my two-phase framework and the performance of the three protocols in the single-phase framework in terms of computational and communication cost. In my experiments, I varied the length of the modulus for PIR, N from 576-1536 bits (the size of a public key) and varied the number of POIs and grid cells from 10K to 100K. In my implementation, I used Kushilevitz and Ostrovsky's version of PIR [57] and Naor and Pinkas's version of 1-of-n OT [76].

7.1.1. Computation Cost at Server's Side

Firstly I compare the server's computation time for the two-phase protocols: PIR+PIR, PIR+OT, OT+PIR and single-phase protocols: PIR, OT, Random OT, for varying grid sizes from 10K to 100K, the results of which are shown in Figure 7.1 and Figure 7.2. For this set of experiments, I kept the number of POIs per cell constant at 1000 (since I was varying the grid size). In the random OT protocol, I varied the number of random cells chosen by the user from 2K for 10K grid to 20K for 100K grid. One can see that PIR+PIR and PIR+OT have similar times while OT+PIR requires significantly more time. This is due to the fact that the server needs to encrypt the entire database column-by-column in OT+PIR, whereas in PIR+OT, the server just has to encrypt one column. The two-level protocols are expectedly more expensive than the single-level PIR and random OT but also offer more privacy. In particular, the client gets only their cell and its neighbors in the two-level protocols, but using single-level PIR, it would get the entire grid column, and using random OT, the server would have a $1/k$ probability of guessing the client's location, where k is the number of cells the client chooses. The two-phase protocols provide a higher level of privacy at the expense of increasing the computation time on the server's side. The cost of single-phase OT is much higher than the two-phase protocols though, which confirms my hypothesis that a 1-of-n OT over the entire grid would be too expensive, although it provides the same level of privacy.

I next compared the computation time required at the server for two-phase protocols with the single-phase protocols with varying modulus sizes. The modulus N as used in PIR is the size of a public key (between 576-1536 bits). As the modulus grows it becomes harder to factor, and hence provides higher security, but incurs more computation cost. Figure 7.3 shows that even for a modulus of size 1536 bits, which is reasonably large, the computation time taken is a little over a minute in the two-phase protocols. I estimate that in applications such as LBS, the typical size of the modulus would be around 768 bits. Figure 7.4 shows the computation time for single-phase PIR which is less than the time for the two-phase

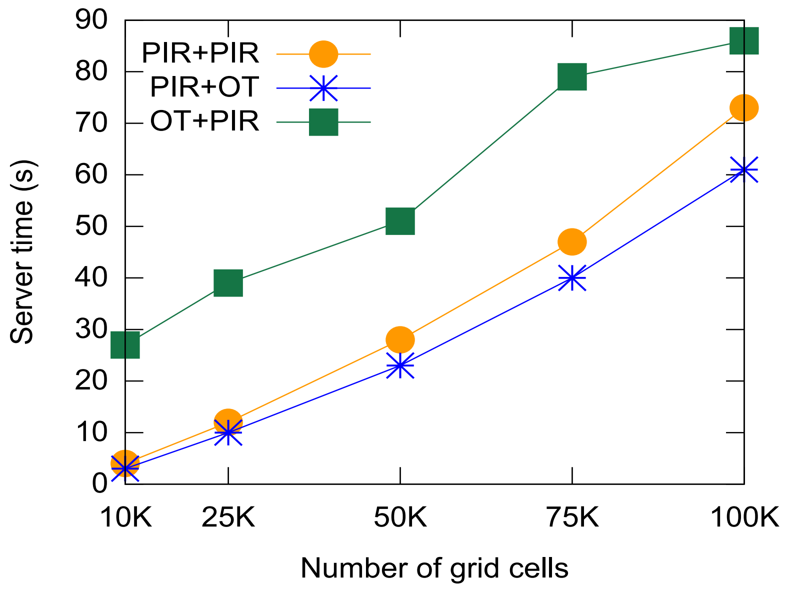


FIGURE 7.1. Server computation time with varying grid size in two-phase protocols

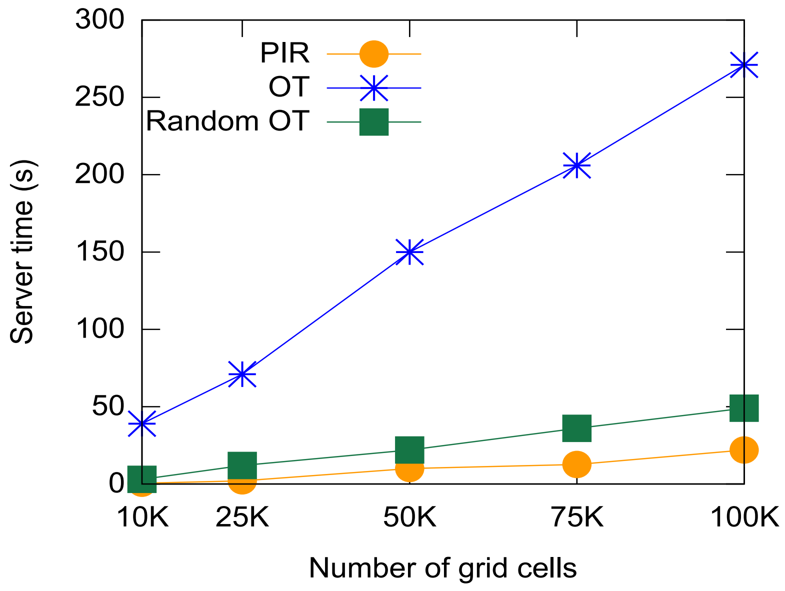


FIGURE 7.2. Server computation time with varying grid size in single-phase protocols

protocols, since I perform computations on the modulus just once. Also, the modulus is used only in PIR, hence I haven't shown graphs for single-phase OT and random OT. In the two-phase protocols, this is reflected in the fact that the time taken for PIR+OT and OT+PIR do not increase as much as PIR+PIR.

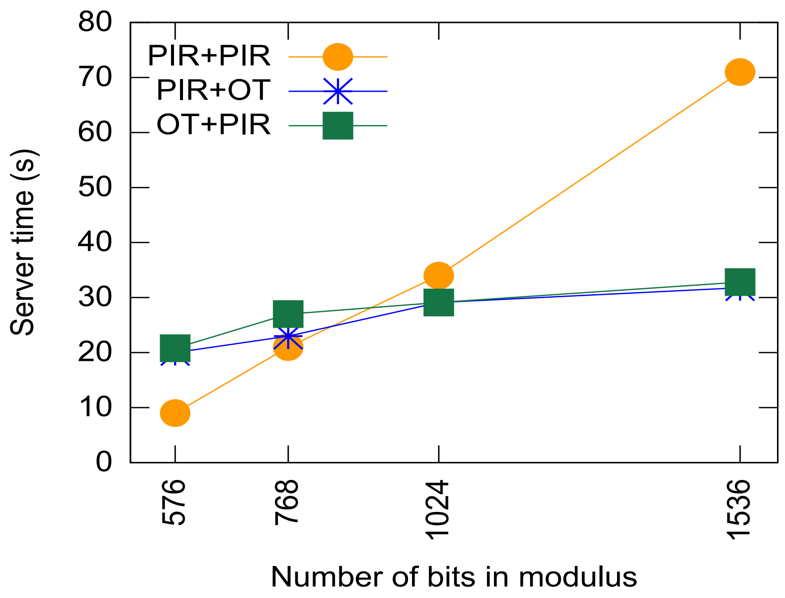


FIGURE 7.3. Server computation time with varying modulus size in two-phase protocols

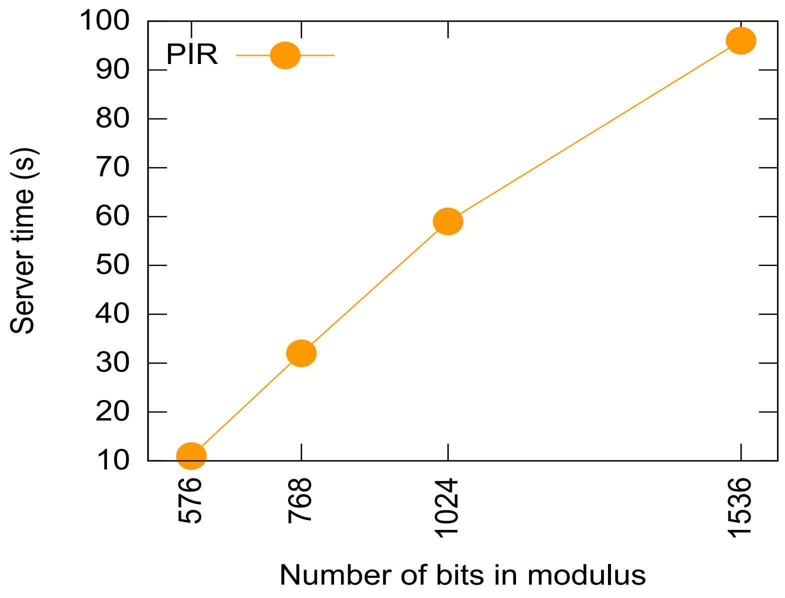


FIGURE 7.4. Server computation time with varying modulus size in single-phase protocol

7.1.2. Communication Cost

Figure 7.5 shows the communication cost in Kb for the two-phase protocols with the number of POIs returned increasing linearly from 10K-100K. The communication cost here denotes the amount of data returned from server to client. The communication cost for PIR+PIR is the least since the server just has to send the PIR response vector (the z vector) back to the client. For PIR+OT and OT+PIR, it is slightly higher since the server needs to send an encrypted column back to the user and also perform a 1-of-2 OT for exchanging keys. Figure 7.6 shows the communication cost in MB for POIs varying from 10K to 100K for the single-phase protocols. I note that the communication cost for the single-phase protocols is in terms of MB rather than Kb as in the two-phase protocols and is one of the major points of difference between the two-phase and single-phase protocols. In single-phase PIR, the communication cost is obviously higher since the server has to send extraneous data to the client. In single-phase OT, I do a 1-of-n OT over the entire database, hence the server and client have to perform $\log n$ 1-of-2 OT's for the client to get its keys, besides the server having to send the entire encrypted database to the client. In the two-phase protocols where I use 1-of-n OT, the OT is performed over a single column and the amount of encrypted data and keys the server has to send the client is much less than in OT over the entire database. In random OT, the server has to send k encrypted cells of the grid to the client and keys for decrypting a single cell out of k cells.

7.1.3. Computation Cost at User's Side

I next compare the user computation time with increasing number of POIs (10K-100K) in the two-phase and single-phase protocols. I can see from Figure 7.7 that the user computation

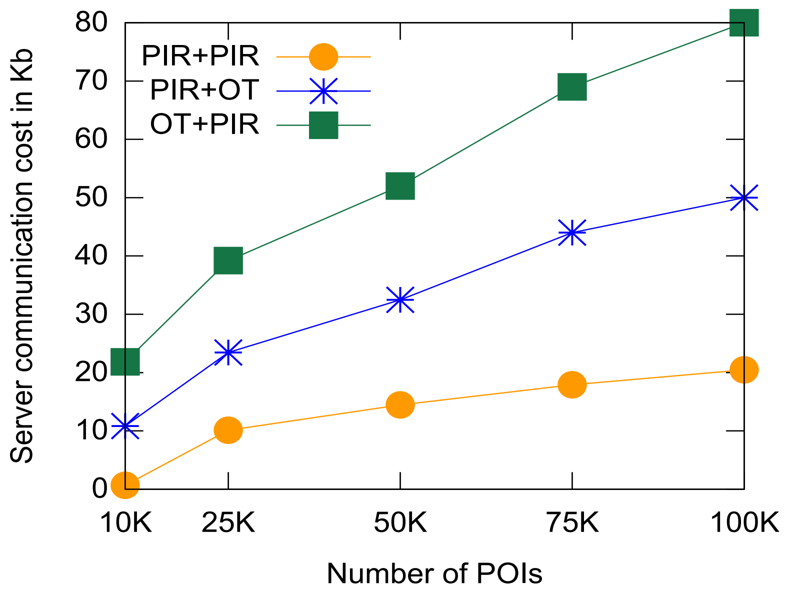


FIGURE 7.5. Server communication cost in two-phase protocols

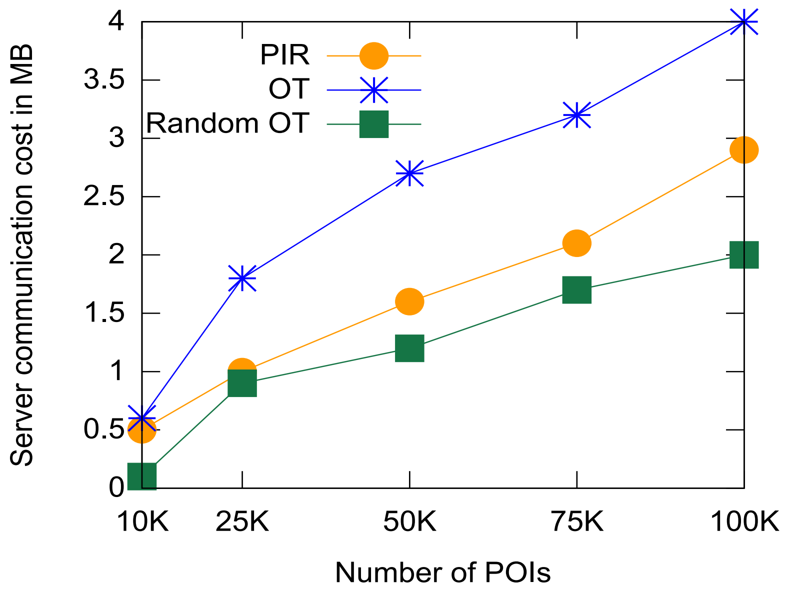


FIGURE 7.6. Server communication cost in single-phase protocols

time for PIR+OT and OT+PIR is slightly higher than PIR+PIR since the user needs to decrypt its grid cell and the cell's nearest neighbors from the data items returned by the server using the keys obtained from the server through 1-of-2 OT. In case of PIR+PIR, the user does not have to perform any decryptions: it just needs to check whether the bits of the string returned by the server are quadratic residues or not. In the single-phase protocols in Figure 7.8, the user computation time is a bit higher for single-phase PIR than for two-phase PIR since the user has to perform the quadratic residue/non-residue check for a few more cells than in the two-level case. The time taken for single-phase OT is significantly higher than the two-phase protocols since the user needs to perform decryptions over $\log n$ elements as opposed to $\log \sqrt{n}$ in the two-phase protocols that involve OT. The user computation time in random OT is less than the single-phase OT, but this really depends on the choice of k in the random OT. If k is too small, the cost will be less, but the privacy offered will also be lower. The largest value of k is n; if one sets $k = n$, the cost will be the same as single-phase OT. This confirms that a single-level OT, while offering the same level of privacy as the two-level protocols is much more expensive than the two-level approaches.

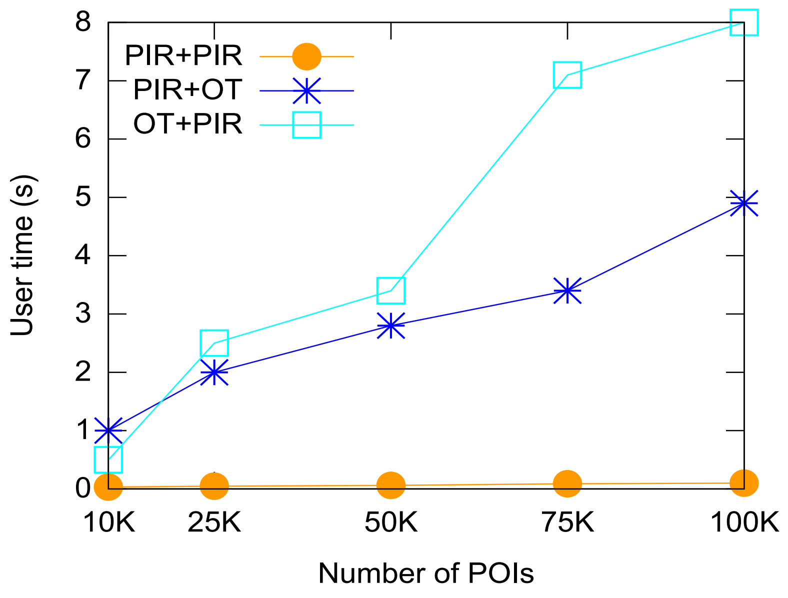


FIGURE 7.7. User computation time with varying number of POIs in two-phase protocols

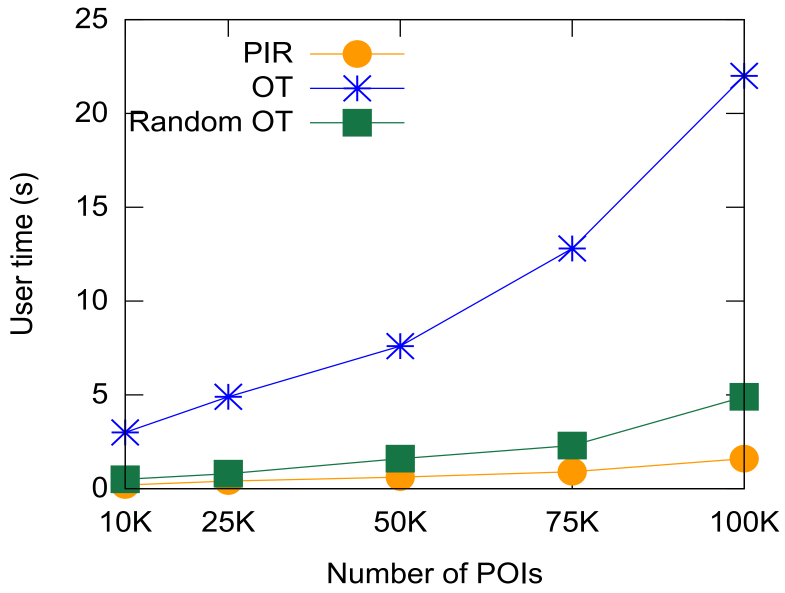


FIGURE 7.8. User computation time with varying number of POIs in single-phase protocols

Figure 7.9 shows the user computation time with a varying modulus, N, from 576 bits to 1536 bits. This is relevant only in the case of PIR+PIR since OT does not use a modulus. In the two-phase protocols, PIR+PIR shows significant variation while the PIR+OT and OT+PIR times are almost constant. Figure 7.10 shows the user computation time in the single-phase protocols with varying modulus bits from 576-1536 bits; for two-phase PIR, this is 2-3 seconds less than the single-phase PIR.

Figure 7.11 and Figure 7.12 show the user computation time for the two-phase protocols and single-phase protocols respectively with varying size of OT keys. The keys used in OT are symmetric keys with size ranging from 80-256 bits. One can see that in the two-phase protocols, the time for PIR+OT and OT+PIR is almost same ranging from 1-13 seconds. For single-phase OT, the user computation time goes from 1-25 seconds. Needless to say, this is due to the extra decryptions performed on the user's side ($\log n$ decryptions in single-phase as compared to $\log \sqrt{n}$ in the two-phase approaches). I measured only the user's computation

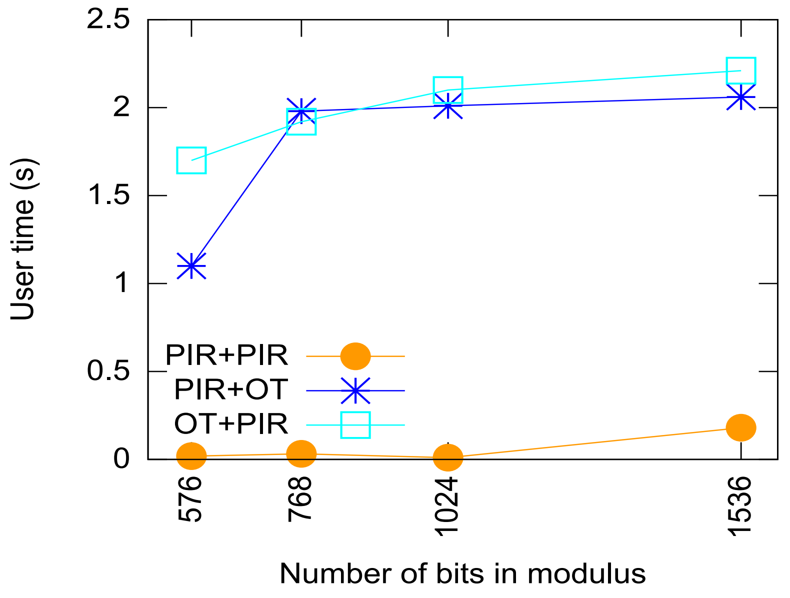


FIGURE 7.9. User computation time with varying modulus size in two-phase protocols

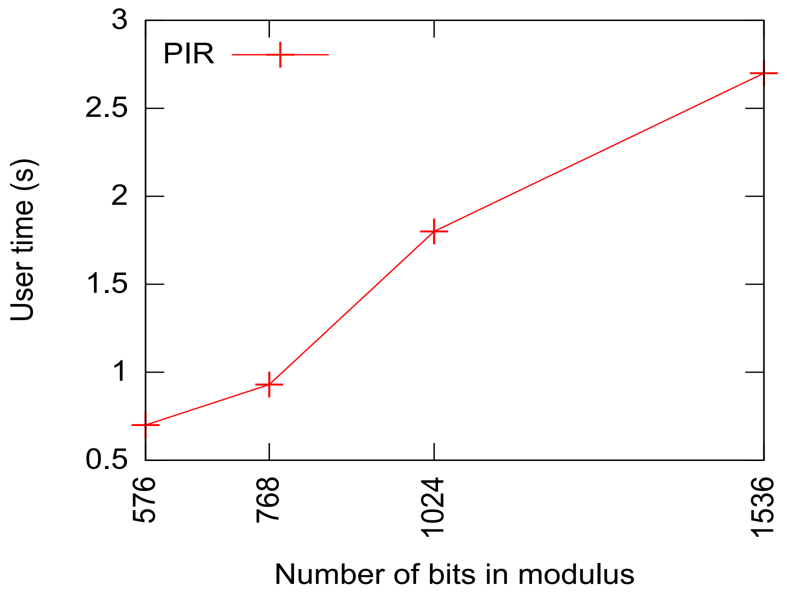


FIGURE 7.10. User computation time with varying modulus size in single-phase protocol

time on varying the key size, since in typical LBS scenarios; the user's device would be a mobile phone, PDA or any resource-constrained device while the server would be a more powerful machine. Hence one would be more interested in minimizing the computation cost on the user's side rather than server's side.

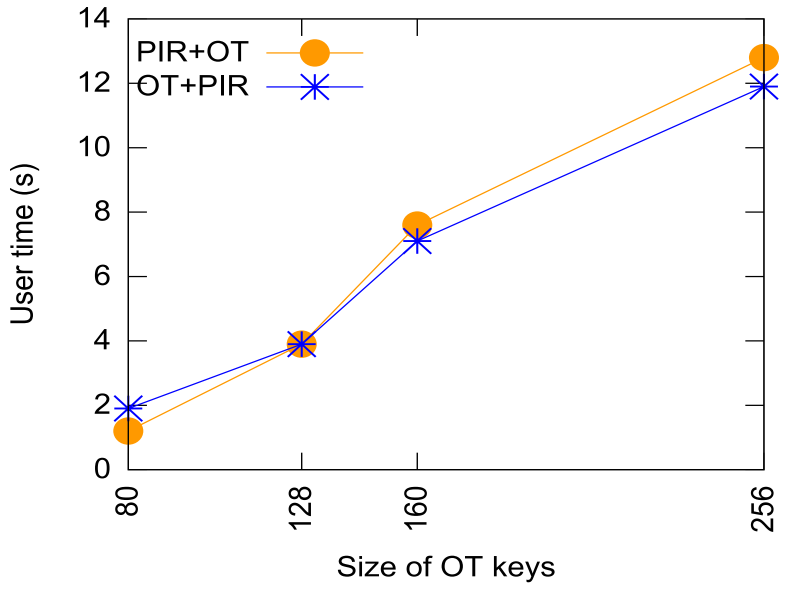


FIGURE 7.11. User computation time with varying OT key size in two-phase protocols

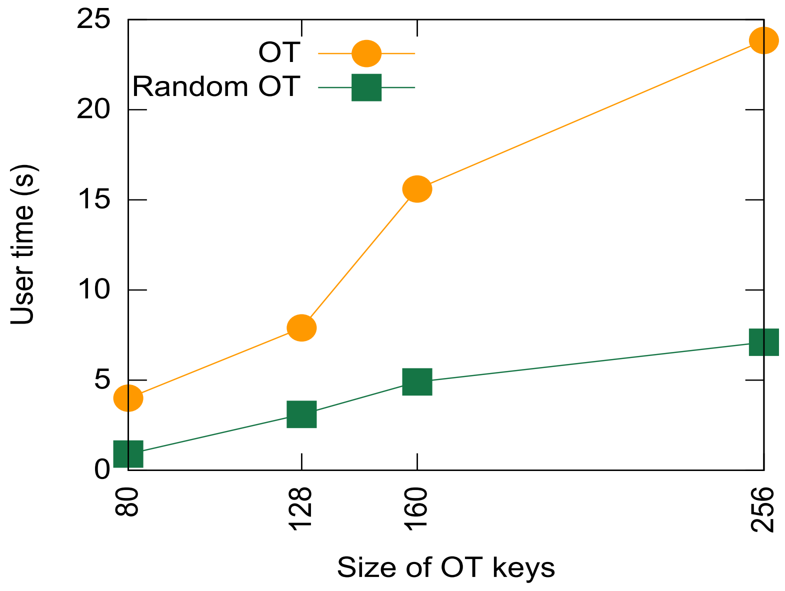


FIGURE 7.12. User computation time with varying OT key size in single-phase protocol

The three two-phase protocols: PIR+PIR, PIR+OT and OT+PIR have a slightly higher (30-50 seconds) server computation cost than single-phase PIR but offer an extra layer of security. The same level of security can be obtained by using a 1-of-n OT, the cost of which is too high. One can use Random OT as a middle ground between the conflicting goals of providing privacy and minimizing cost, but this does not provide 100% privacy which the other protocols do. Also, it should be noted that the communication cost of the two-phase protocols are far lower than the single-phase protocols.

7.2. Peer-To-Peer Semi-Honest Model

I developed a prototype of my framework in Java and implemented the centralized and distributed models of garbled circuit creation and evaluation. I varied the number of locations between 200-2000 and considered the number of users to be 10. In the experiments, I considered a fully distributed model i.e., where all users in the system participate in circuit creation and evaluation. One of the main goals of my experiments was to show how scalable my models were and whether or not they are applicable in real-world situations. Hence I had to consider as large a number of locations as realistically possible. The garbled circuit I built for the group nearest neighbor function consists of two kinds of gates: a set of adder gates and a comparator gate. I assumed the input bit length of each party to be 64 bits ($m = 64$). For computing the sum of the distances of each party $P_1, P_2, \cdots, P_n$ from each location $L_1, L_2, \cdots, L_l$ — $s_1 = \sum_{i=1}^{n} d(P_i, L_1), s_2 = \sum_{i=1}^{n} d(P_i, L_2), \cdots, s_l = \sum_{i=1}^{n} d(P_i, L_l)$, I created an adder gate for each location; in total one would have $lm-$bit adder gates, one for each location. Each adder was implemented as a ripple-carry adder [69], e.g., a 64-bit adder was constructed by connecting 64 1-bit adders in series. After the sums of distances have been computed, one needs to compare them to find the minimum sum of distance, which I modeled using a single $m-$ bit comparator gate. The total number of gates used in my approach is $(l+1)m-$bit gates.

7.2.1. Centralized and Distributed Models Cost

I measured the performance of the centralized model and distributed models with the number of locations ranging from 200-2000. Fig 7.13 shows the time taken in seconds in the centralized and distributed models for a single user to create the circuit. Fig 7.14 shows the time taken in seconds for a single user to evaluate the circuit in the centralized and distributed models. The time taken for creating a circuit is more than that for evaluating a circuit since creating a circuit involves more expensive steps like generating random keys, encrypting the plaintext values with them, and generating the truth tables. In the centralized model, a single user creates and a single user evaluates the circuit; in the distributed model, the locations and users are divided into groups, with each user group being responsible for creating and evaluating the circuit for their assigned group of locations, hence the workload on each user in distributed setting is far lower than that in the centralized setting.

The communication costs in both the centralized and distributed models are dominated by the oblivious transfer (OT) operation between the circuit creator and all other users and the number of OT's would be the same in both the centralized and distributed models. This is because in the centralized model, all users would perform an OT with a single circuit creator over all locations. In the distributed model, each user will perform OT's with many other circuit creators, but the number of OT's performed with each circuit creator would be the same as the number of locations that creator is responsible for. For example, if there are 200 locations and 10 users, in the centralized model, each user would perform 200 OT's with a single circuit creator. In the distributed model, each user will perform 40 OT's with

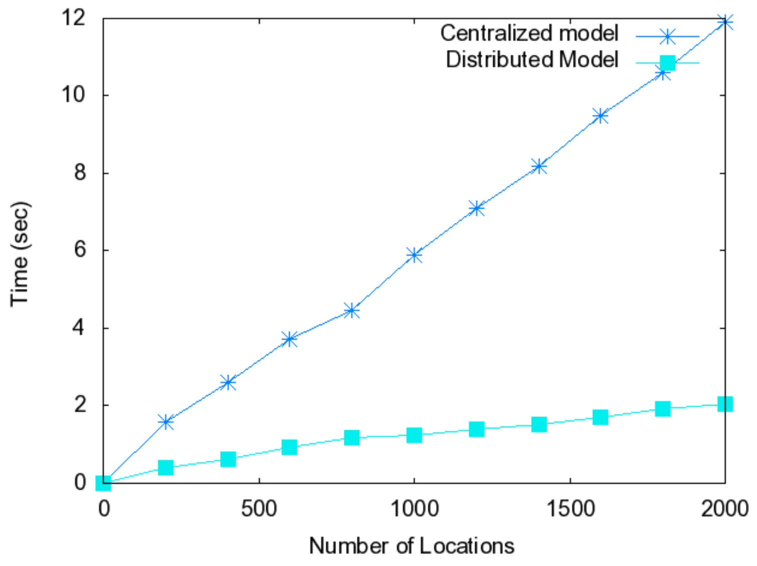


FIGURE 7.13. Time taken for creating the circuit in the centralized and distributed models

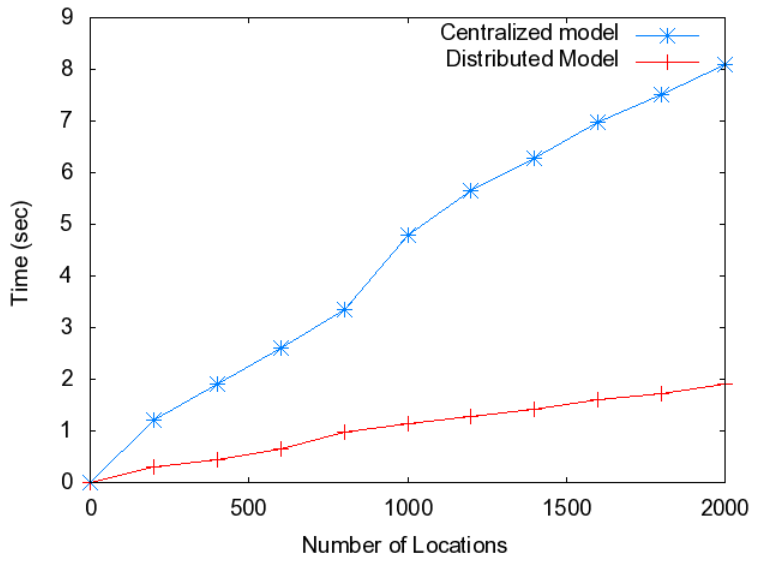


FIGURE 7.14. Time taken for evaluating the circuit in the centralized and distributed models

5 circuit creators; in both cases, the total number of OT's remain the same. Hence the communication costs of the centralized and distributed models are the same.

7.3. Peer-To-Peer Dishonest Model

The TPM commands I propose in Chapter 6 are not supported by the current TPM v.1.2 specification, hence I needed to implement the TPM commands (extensions) on a TPM simulator.

I used TPM/J developed by Sarmenta et al. [67], a Java-based front-end interface, for communicating with the TPM simulator which interfaces with the simulator's device driver. I have used v.0.5 of the TPM simulator by Mario Strasser [81] in my experiments and added the required SFE commands: TPM_SFEInit, TPM_SFEInputs, TPM_SFEGate and TPM_SFEFinish to it. TPMs are currently manufactured by 6 companies: Atmel, BroadComm, Infineon, STMicroelectronics, Winbond and Intel. I have run the TPM simulator with the performance profile of an Infineon TPM, since this is among the faster TPMs. The performance profiles were part of earlier work due to Gunupudi and Tate [38] in which they had benchmarked profiles of four real TPMs: Atmel, STMicro, Infineon and Winbond on the TPM simulator. The performance profiles are simply time delays introduced in the simulator (which runs on the host CPU) in order for it to better mimic the speed of a real TPM, since real TPMs' processing speed is far slower than that of a standard desktop CPU.

7.3.1. Individual Command Times

I first report the time taken for executing individual commands on the TPM simulator and Flicker. The SFE_Init and SFE_Finish commands are one-time costs while the SFE_Inputs and SFE_Gate need to be executed for each input set of input wires and input gates. Note that while the TPM processes the gates one-by-one, Flicker can take in up to 60 gates at a time and encrypt them. Considering that the overhead incurred by

Flicker per command is very high - around 3-4 seconds per command; by passing in 60 gates at a time, one would be incurring $\frac{1}{60}$ the penalty. Table 7.1 shows the time taken for the TPM and Flicker to execute individual SFE commands. The SFE_Init, SFE_Inputs and SFE_Gate all involve hash operations using a cryptographic hash function (SHA-1 in the TPM simulator) which takes around 0.003 seconds on an average on a Infineon TPM and hence all the three commands measure in the order of milliseconds. The SFE_Finish command involves a signature which takes around 0.2 seconds on an average on an Infineon TPM, hence the time taken by this command is a bit higher than the others.

Table 7.1. SFE command times

Command	TPM (seconds)	Flicker (seconds)
SFE_Init	0.001 s	3 s
SFE_Inputs	0.003 s	4 s
SFE_Gate	0.05 s	5 s
SFE_Finish	0.2 s	4 s

7.3.2. SFE Timings

The TPM commands described above were used in the construction of a garbled circuit-based SFE protocol as described in Chapter 6. The same circuit was also constructed using a trusted computing-based infrastructure called Flicker [62]. The main purpose of creating the circuit in two ways was to compare the performance of the TPM and Flicker to decide which option was more feasible: extending the TPM to support SFE commands or use existing trusted computing infrastructure to support SFE. If existing infrastructure can support SFE efficiently, one does not have much reason or justification for extending the TPM specification. The time taken for creating and evaluating the TPM-assisted garbled circuit and Flicker-assisted garbled circuit are as shown in Table 7.2. Since the overhead incurred by Flicker is around 3-4 seconds per command, and the times per command as executed by the TPM are around 0.004 seconds, the TPM outperforms Flicker by a significant margin. The evaluation times, which are shown in Table 7.3, are more or less similar to the semi-honest model, as shown in Figure 7.13 and Figure 7.14, since on the evaluator's side I do not use any kind of hardware support. The only thing that is different from an evaluator in the semi-honest model is the fact that the evaluator needs to validate the TPM's signature on the encrypted and plaintext circuits as described in the **Algorithm GCEvaluator** in Chapter 6.

Table 7.2. Hardware-assisted circuit creation time

Number of Gates	TPM (seconds)	Flicker (seconds)
501	25.3 s	56 s
1001	51.81 s	98 s
1501	74.11 s	137 s
2001	102.84 s	186 s

Table 7.3. Circuit evaluation time

Number of Gates	Evaluation time (seconds)
501	3 s
1001	5.6 s
1501	7.0 s
2001	8.9 s

Finally, I compare the performance of circuit creation in the semi-honest model and in the dishonest model (TPM and Flicker assisted circuit creation) in Figure 7.15. I find that

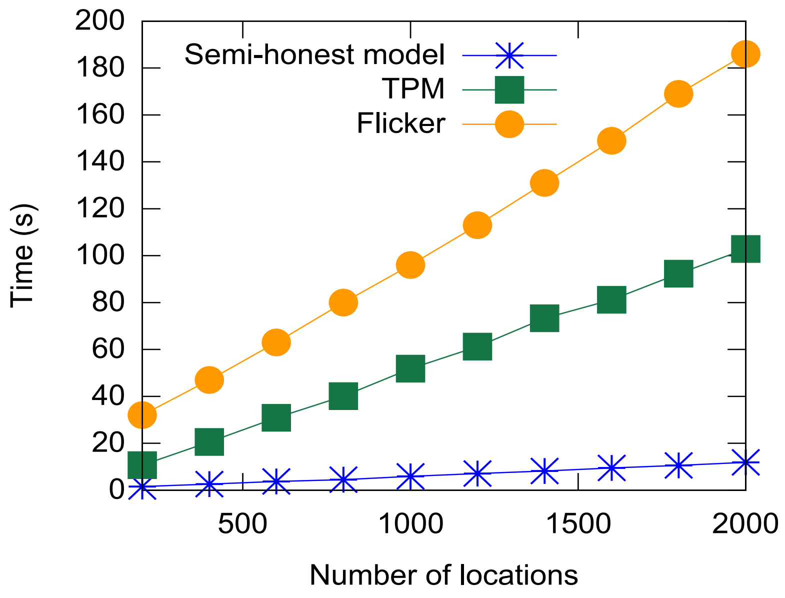


Figure 7.15. Comparison of circuit creation costs

one can obtain a higher level of security in the dishonest model at the cost of around 2-3 minutes extra computation time.

CHAPTER 8

CONCLUSION AND FUTURE WORK

8.1. Conclusion

In this dissertation, I have considered three situations in location-based services that have privacy implications and proposed solutions for them. Each of the solutions I have developed have a theoretical component as well as experiments that test the feasibility of applying them in location-based services. In the single-user, client-server model, I have proposed a two-phase privacy-preserving framework that guarantees user privacy, while ensuring the server disseminates as precise data as possible. I have also described a single-phase framework with which I compare my two-phase protocols. I have found that the protocols in the two-phase framework provide strong privacy guarantees for both, the client and server, and are efficient to implement in terms of computation and communication cost. I have also defined the privacy properties desired of my protocols and provided a proof sketch for them. My experiments show that the two-phase protocols using PIR and OT have reasonable costs wherein most operations take around a minute and are thus feasible to use in real-world LBS applications.

In the peer-to-peer semi-honest model, I have explored the application of secure multi-party computation techniques to address privacy issues in location-based queries. I have considered the group nearest neighbor query and shown how one can guarantee privacy of all the users involved without requiring a trusted third party and when none of the users trust each other. Compared to previous work in this area which has mainly considered the client-server model and proposed ways to achieve client privacy, I consider the peer-to-peer model and show how to preserve privacy of all peers involved. My experimental results show that my approach is practical and has reasonable costs.

In the peer-to-peer dishonest model, I have considered the application of trusted computing technology, specifically TPMs to construct secure function evaluation (SFE) protocols and have implemented them in location-based services. The protocols I propose are proven secure under standard computational assumptions and under the assumption that TPMs are secure. My construction does require extra functionality added to the TPM v.1.2 specification in the form of TPM extensions. Another approach that I have briefly explored - homomorphic encryption, requires fewer extensions to the existing TPM specification than my garbled circuit-based protocols. For TPM-assisted garbled circuits, I require 4 TPM extensions vis-a-vis one extension for TPM-assisted homomorphic encryption. But in the garbled circuit extensions, the only main thing one needs is a HMAC (which can be used as a PRF) which seems a more realistic extension than the homomorphic encryption extension which requires the TPM to support an entirely new asymmetric encryption scheme - ElGamal. Since TPMs already support HMAC, my garbled circuit construction inspite of having 4 extensions, really requires no new cryptographic capabilities at all. It might be worth investigating whether by combining the two TPM-assisted techniques, one can make the hybrid protocol secure in the malicious model with a minimal number of TPM extensions. I have also compared the performance of the TPM-assisted circuit construction with a Flicker-assisted circuit construction (which uses a trusted computing-based infrastructure called Flicker [62]) and I find that the TPM-assisted circuit construction outperforms the Flicker-assisted circuit construction which has a very high overhead per operation.

One way the two-phase framework can be improved is by using a more efficient OT protocol such as the one presented by Gunupudi and Tate [40] which makes the OT protocol non-interactive as opposed to the interactive OT used in this dissertation. Using a non-interactive OT protocol would significantly decrease the communication cost of the OT operation; but requires extra hardware security assumptions in the form of a trusted platform module chip [35] (TPM) on the server's side. In this single-user model, I did not consider the hardware-assisted model of computation and hence have not used [40]. One can also extend my framework beyond nearest neighbor queries such as k-nearest neighbor or other spatial queries.

In the garbled circuit-based protocol in Chapter 6, I had considered Kolesnikov and Schneider's free-XOR gate circuit construction technique which requires an unbounded amount of storage for the XOR gate option (for storing garbled input values of all gates). I side-stepped this problem by treating all gates as regular, non-XOR gates and constructed truth tables for all of them. One direction for future work is to find a solution where one can use XOR gates with limited storage. This would be useful since the circuit creator wouldn't have to compute and transmit extra truth tables to the evaluator. Another direction for future work is to take a different mathematical approach to homomorphic encryption other than the asymmetric ciphers mentioned in this dissertation (Paillier's system or its derivatives, ElGamal, etc.) and explore pairing-based asymmetric ciphers such as the Weil/Tate pairing and examine their suitability for firstly, homomorphic encryption, and next, explore whether usage of trusted hardware in them could improve them in a general sense, or for any specific applications.

BIBLIOGRAPHY

[1] F. Armknecht and A. Sadeghi, *A new approach for algebraically homomorphic encryption*, Cryptology ePrint Archive, Report 2008/422, 2008.

[2] Y. Aumann and Y. Lindell, *Security against covert adversaries*, TCC, 2007, pp. 137–156.

[3] J. Bar-Ilan and D. Beaver, *Non-cryptographic fault-tolerant computing in a constant number of rounds*, ACM PODC, 1989, pp. 201–219.

[4] D. Beaver, S. Micali, and P. Rogaway, *The round complexity of secure protocols*, ACM STOC, 1988, pp. 1–10.

[5] A. Beimal, Y. Ishai, and T. Malkin, *Reducing the servers' computation in private information retrieval: PIR with pre-processing*, Journal of Cryptology, 2004, pp. 125–151.

[6] M. Bellare, *New proofs for NMAC and HMAC: Security without collision-resistance*, CRYPTO, Springer-Verlag, 2006, pp. 602–619.

[7] J. C. Benaloh and D. Tuinstra, *Receipt-free secret-ballot elections (extended abstract)*, ACM STOC, 1994, pp. 544–553.

[8] G. Brassard, C. Crépeau, and J.M. Roberts, *All or nothing disclosures of secrets*, CRYPTO, 1986, pp. 234–238.

[9] E. Brickell, J. Camenisch, and L. Chen, *Direct anonymous attestation*, ACM CCS, 2004, pp. 132–145.

[10] R. Canetti and R. Ostrovsky, *Secure computation with honest-looking parties: What if nobody is truly honest*, ACM STOC, 1999, pp. 255–264.

[11] N. Chandran, V. Goyal, and A. Sahai, *New constructions for UC secure computation using tamper proof hardware*, EUROCRYPT, 2008, pp. 545–562.

[12] D. Chaum, C. Crépeau, and I. Damgård, *Multi-party unconditionally secure protocols*, ACM STOC, 1988, pp. 11–19.

[13] D. Chaum and T. P. Pedersen, *Wallet databases with observers (extended abstract)*, Advances in Cryptology - CRYPTO, Springer-Verlag, 1992, pp. 89–105.

[14] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, *Private information retrieval*, IEEE FOCS, 1995, pp. 41–50.

[15] C. Y. Chow, M. Mokbel, and X. Liu, *A peer-to-peer spatial cloaking algorithm for anonymous location-based services*, ACM GIS, 2006, pp. 247–256.

[16] R. Cramer and I. Damgård, *Secure distributed linear algrbra in a constant number of rounds*, CRYPTO, 2001, pp. 119–136.

[17] I. Dåmgard, M. Geisler, and M. Krøigaard, *Efficient and secure comparison for on-line auctions*, ACISP, 2007, pp. 416–430.

[18] I. Dåmgard and M. Jurik, *A generalization, a simplification and some applications of paillier's probabilistic public-key system*, PKC, 2001, pp. 119–136.

[19] I. Damgård, J. B. Nielsen, and D. Wichs, *Isolated proofs of knowledge and isolated zero knowledge*, EUROCRYPT, 2008, pp. 509–526.

[20] M. Van Dijk, C. Gentry, S. Halevi, and V. Vaikunthanathan, *Fully homomorphic encryption over the integers*, CRYPTO, 2010, pp. 24–43.

[21] U. Feige, J. Kilian, and M. Naor, *A minimal model for secure computation (extended abstract)*, ACM STOC, 1994, pp. 554–563.

[22] B. Gedik and L. Liu, *Privacy in mobile systems: A personalized anonymization model*, Proc. of ICDCS, 2005, pp. 620–629.

[23] C. Gentry, *Fully homomorphic encryption using ideal lattices*, ACM STOC, 2009, pp. 169–178.

[24] Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin, *Protecting data privacy in private information retrieval schemes*, Journal of Computer Systems Science, 2000, pp. 592–629.

[25] G. Ghinita, P. Kalnis, A. Khoshgozaran, C. Shahabi, and K.L. Tan, *Private queries in location based services: Anonymizers are not necessary*, ACM SIGMOD, 2008, pp. 121–132.

[26] G. Ghinita, P. Kalnis, and S. Skiadopoulos, *PRIVE: Anonymous location-based queries in distributed mobile systems*, WWW, 2007, pp. 371–380.

[27] O. Goldreich, *Foundations of Cryptography*, First ed., Cambridge University Press, New York, 2001.

[28] O. Goldreich, S. Micali, and A. Wigderson, *How to play any mental game*, ACM STOC, 1987, pp. 218–229.

[29] O. Goldreich and R. Ostrovsky, *Software protection and simulation on oblivious RAMs*, Jounal of ACM (1996), 431–473.

[30] S. Goldwasser, Y.T. Kalai, and G.N. Rothblum, *One-time programs*, CRYPTO, 2008, pp. 39–56.

[31] S. Goldwasser and S. Micali, *Probabilistic encryption and how to play mental poker keeping secret all partial information*, ACM STOC, 1982, pp. 365–377.

[32] V. Goyal, Y. Ishai, A. Sahai, R. Venkatesan, and A. Wadia, *Founding cryptography on tmaper-proof hardware tokens*, TCC, 2010, pp. 308–326.

[33] V. Goyal, P. Mohassel, and A. Smith, *Efficient two-party ad multi-party computation against covert adversaries*, Advances in Cryptology - EUROCRYPT, 2008, pp. 289–306.

[34] Trusted Computing Group, *Protection profile — PC client specific trusted platform module*, Available at https://www.trustedcomputinggroup.org/specs/TPM/, TPM Family 1.2; Level 2.

[35] ______, *Trusted Platform Module Specifications – Parts 1–3*, Available at https://www.trustedcomputinggroup.org/specs/TPM/.

[36] M. Gruteser and D. Grunwald, *Anonymous usage of location-based services through spatial and temporal cloaking*, ACM MobiSys, 2003, pp. 31–42.

[37] V. Gunupudi, *Exploring trusted platform module capabilities: a theoretical and experimental study*, Ph.D. thesis, University of North Texas, 2008.

[38] V. Gunupudi and S. R. Tate, *Timing-accurate TPM simulation for what-if explorations in trusted computing*, To appear in SPECTS '10.

[39] ______, *Random oracle instantiation of distributed protocols using trusted platform modules*, AINA, 2007, pp. 463–469.

[40] ______, *Generalized non interactive oblivious transfer using count-limited objects with applications to secure mobile agents*, FC, 2008, pp. 98–112.

[41] C. Hazay and Y. Lindell, *Constructions of truly practical secure protocols using standard smartcards*, ACM CCS, 2008, pp. 491–500.

[42] U. Hengartner, *Location privacy based on trusted computing and secure logging*, SecureComm, 2008, pp. 1–8.

[43] A. Illiev and S. Smith, *Private information storage with logarithmic-space secure hardware*, Workshop on Information Security, Management, Education and Privacy, 2004,

pp. 210–216.

[44] Y. Ishai and I. Damgaard, *Constant-round multiparty computation using a black-box pseudorandom generator*, Advances in Cryptology - CRYPTO, 2005, pp. 378–394.

[45] Y. Ishai, E.Kushilevitz, Y. Lindell, and E. Petrank, *Black-box constructions for secure computation*, ACM STOC, 2006, pp. 99–108.

[46] Y. Ishai and E. Kushilevitz, *Randomizing polynomials: A new representation with applications to round-efficient secure computation*, IEEE FOCS, 2000, pp. 294–304.

[47] S. Jarecki and V. Shmatikov, *Efficient two-party secure computation on committed inputs*, Advances in Cryptology - EUROCRYPT, 2007, pp. 97–114.

[48] K. Järvinen, V. Kolesnikov, A. Sadeghi, and T. Schneider, *Embedded SFE: Offloading server and network using hardware tokens*, FC, To appear.

[49] C.S. Jensen, J. Kolar, and T.B. Pedersen adn I. Timko, *Nearest neighbour queries in road networks*, ACM GIS, 2003, pp. 1–8.

[50] P. Kalnis, G. Ghinita, K. Mouratidis, and D. Papadias, *Preventing location-based identity inference in anonymous spatial queries*, IEEE TKDE, 2007, pp. 1719–1733.

[51] J. Katz, *Universally composable multi party computation using tamper-proof hardware*, EUROCRYPT, 2007, pp. 115–128.

[52] J. Katz, R. Ostrovsky, and A. Smith, *Round-efficiency of multi-party computation with a dishonest majority*, Advances in Cryptology - EUROCRYPT, 2003, pp. 578–595.

[53] A. Khoshgozaran and C. Shahabi, *Blind evaluation of nearest neighbour queries using spatial transformation to preserve location privacy*, SSTD, 2007, pp. 239–257.

[54] ———, *Private information retrieval techniques for enabling location privacy in location-based services*, Privacy in Location-based Applications, 2009, pp. 59–83.

[55] V. Koleshnikov, A. Sadeghi, and T. Schneider, *Improved garbled circuit building blocks and applications to auctions and computing minima*, CANS, 2009, pp. 1–20.

[56] V. Kolesnikov and T. Schneider, *Improved garbled circuit: Free xor gates and applications*, ICALP, 2008, pp. 486–498.

[57] E. Kushilevitz and R. Ostrovsky, *Replication is not needed: single database, computationally private information retrieval*, IEEE FOCS, 1997, pp. 364–373.

[58] F. Li, D. Cheng, M. Hadjieleftheriou, G. Kollios, and S. Teng, *On trip planning queries in spatial databases*, SSTD, 2005, pp. 273–290.

[59] Y. Lindell and B. Pinkas, *An efficient protocol for secure two-party computation in the presence of malicious adversaries*, Advances in Cryptology - EUROCRYPT, 2007, pp. 52–78.

[60] ———, *A proof of security of yao's protocol for two-party computation*, Journal of Cryptology 22 (2009), no. 2, 161–188.

[61] D. Malkhi, N. Nisan, B. Pinkas, and Yaron Sella, *Fairplay—a secure two-party computation system*, USENIX Security Symposium, 2004, pp. 1–20.

[62] J.M. McCune, B. Parno, A. Perrig, M. Reiter, and H. Isozaki, *Flicker: An execution infrastructure for TCB minimization*, EuroSys, 2008, pp. 315–328.

[63] C. Aguilar Melchor, P. Gaborit, and J. Herranz, *Additive homomorphic encryption with t-operand multiplications*, Cryptology ePrint Archive, Report 2008/378, 2008.

[64] P. Mohassel and M.K. Franklin, *Efficiency trade-offs for malicious two-party computation*, PKC, 2006, pp. 458–473.

[65] M. Mokbel, C. Chow, and W. Aref, *The new casper: Query processing for location services without compromising privacy*, VLDB, 2006, pp. 219–229.

[66] T. Moran and G. Segev, *David and Goliath commitments: UC computation for asymmetric parties using tamper-proof hardware*, EUROCRYPT, 2008.

[67] T. Muller, L.F.G. Sarmenta, and J. Rhodes, *TPM/J - Java based API for the Trusted Platform Module TPM*, Available at http://projects.csail.mit.edu/tc/tpmj/.

[68] M. Naor, B. Pinkas, and R. Sumner, *Privacy preserving auctions and mechanism design*, ACM Conference on Electronic Commerce, 1999, pp. 129–139.

[69] V. Nelson, H. Nagle, B. Carroll, and D. Irwin, *Digital logic circuit analysis and design*, Prentice Hall, New Jersey, 1995.

[70] National Institute of Standards and Technology, *Secure hash standard*, FIPS PUB 180-2, 2000.

[71] ______, *HMAC: Keyed hashing for message authentication*, RFC 2104, 2002.

[72] P. Paillier, *Public-key cryptosystems based on composite degree residuosity classes*, EUROCRYPT, 1999, pp. 223–238.

[73] D. Papadias, Q. Shen, Y. Tao, and K. Mouratidis, *Group nearest neighbour queries*, ICDE, 2004, pp. 301–313.

[74] D. Papadias, Y. Tao, J. Zhang, and N. Mamoulis, *Query processing in spatial network databases*, VLDB, 2003, pp. 802–813.

[75] R. Pass, *Bounded concurrent secure multi-party computation with a dishonest majority*, ACM STOC, 2004, pp. 232–241.

[76] B. Pinkas and M. Naor, *Computationally secure oblivious transfer*, Journal of Cryptology (2005), 1–35.

[77] R.L. Rivest, A. Shamir, and L. M. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the ACM (1978), 120–126.

[78] T. Sander, A. Young, and M. Yung, *Non-interactive cryptocomputing for nc1*, IEEE FOCS, 1999, pp. 554–566.

[79] C. Shahabi, M.R. Kolahdouzan, and M. Sharifzadeh, *A road network embedding technique for k-nearest neighbour serch in moving object databases*, ACM GIS, 2002, pp. 94–100.

[80] N. P. Smart and F. Vercauteren, *Fully homomorphic encryption with relatively small key and ciphertext sizes*, PKC, 2010, pp. 420–443.

[81] M. Strasser, H. Stamer, and J. Molina, *Software-based TPM simulator*, http://tpm-emulator.berlios.de.

[82] S.R. Tate and R. Vishwanathan, *Improving cut-and-choose in verifiable encryption and fair exchange protocols using trusted computing technology*, IFIP DBSeC, 2009, pp. 252–267.

[83] S.R. Tate and K. Xu, *On garbled circuits and constant round secure function evaluation*, Tech. Report 2003-02, CoPS Lab, University of North Texas, 2003.

[84] D. Woodruff, *Revisiting the efficiency of malicious two-party computation*, Advances in Cryptology - EUROCRYPT, 2007, pp. 79–96.

[85] A.C. Yao, *How to generate and exchange secrets*, IEEE FOCS, 1986, pp. 162–167.

[86] M.L. Yiu, C. Jensen, X. Huang, and H. Lu, *Spacetwist: managing the trade-offs among location privacy, query performance and query accuracy in road networks*, ICDE, 2008, pp. 366–375.

[87] M.L. Yiu, N. Mamoulis, and D. Papadias, *Aggregate nearest neighbour queries in road networks*, IEEE TKDE, 2005, pp. 820–833.