Privacy-Preserving Shortest Path Computation

David J. Wu, Joe Zimmerman, Jérémy Planul, John C. Mitchell
Stanford University
{dwu4, jzim, mitchell}@cs.stanford.edu, jeremy.planul@ens-lyon.org

***Abstract*—Navigation is one of the most popular cloud computing services. But in virtually all cloud-based navigation systems, the client must reveal her location and destination to the cloud service provider in order to learn the fastest route. In this work, we present a cryptographic protocol for navigation on city streets that provides privacy for both the client's location and the service provider's routing data. Our key ingredient is a novel method for compressing the next-hop routing matrices in networks such as city street maps. Applying our compression method to the map of Los Angeles, for example, we achieve over tenfold reduction in the representation size. In conjunction with other cryptographic techniques, this compressed representation results in an efficient protocol suitable for fully-private real-time navigation on city streets. We demonstrate the practicality of our protocol by benchmarking it on real street map data for major cities such as San Francisco and Washington, D.C.**

I. Introduction

Location privacy is a major concern among smartphone users, and there have been numerous controversies due to companies tracking users' locations [1], [17]. Among the various applications that require location information, navigation is one of the most popular. For example, companies such as Google, Apple, and Waze have built traffic-aware navigation apps to provide users with the most up-to-date routing information. But to use these services, users must reveal their location and destination to the cloud service provider. In doing so, they may also reveal other sensitive information about their personal lives, such as their health condition, their social and political affiliations, and more.

One way to provide location privacy is for the user to download the entire map from the cloud service provider and then compute the best route locally on her own mobile device. Unfortunately, since service providers invest significant resources to maintain up-to-date routing information, they are not incentivized to publish their entire routing database in real-time. Even in the case of a paid premium service, in which the service provider does not derive compensation from learning the user's location data, it is not obvious how to achieve fully-private navigation. The user does not trust the cloud provider with her location data, and the cloud provider does not trust the user with its up-to-date routing information, so neither party has all of the data to perform the computation. While general-purpose cryptographic tools such as multiparty computation solve this problem in theory (see Section VII), these protocols are prohibitively expensive in practice for applications such as real-time navigation.

Our results. In this work, we present an efficient cryptographic protocol for *fully-private* navigation: the user keeps private her location and destination, and the service provider keeps private its proprietary routing information (except for the routing information associated with the specific path requested by the user and a few generic parameters pertaining to the network). We give a complete implementation of our protocol and benchmark its performance on real street map data (Section V-C). Since our protocol is real-time (the user continues receiving directions throughout the route), we benchmark the performance "per hop", where each hop roughly corresponds to an intersection between streets.[1] For cities such as San Francisco and Washington, D.C., each hop in our protocol requires about 1.5 seconds and less than 100 KB of bandwidth. In addition, before the protocol begins, we execute a preprocessing step that requires bandwidth in the tens of megabytes. Since this preprocessing step can be performed at any time, in practice it would likely be run via a fast Wi-Fi connection, before the mobile user needs the real-time navigation service, and thus the additional cost is very modest. To our knowledge, ours is the first fully-private navigation protocol efficient enough to be feasible in practice.

Our technical contributions. In our work, we model street-map networks as graphs, in which the nodes correspond to street intersections, and edges correspond to streets. In our model, we assume that the network topology is public (i.e., in the case of navigation on city streets, the layout of the streets is publicly known). However, only the service provider knows the up-to-date traffic conditions, and thus the shortest path information. In this case, the server's "routing information" consists of the weights (that is, travel times) on the edges in the network.

By modeling street-maps as graphs, we can easily construct a straw-man private navigation protocol based on symmetric private information retrieval (SPIR) [24], [36], [48]. Given a graph $\mathcal{G}$ with n nodes, the server first constructs a database with n^2 records, each indexed by a source-destination pair (s, t). The record indexed (s, t) contains the shortest path from s to t. To learn the shortest path from s to t, the client engages in SPIR with the server for the record indexed (s, t). Security of SPIR implies that the client just learns the shortest path and the server learns nothing. While this method satisfies the basic security requirements, its complexity scales quadratically

NDSS '16, 21-24 February 2016, San Diego, CA, USA

http://dx.doi.org/10.14722/ndss.2016.23052

[1] In a few cases, hops in our construction occur mid-street or in instances such as traffic circles. These are rare enough that even in large cities such as Los Angeles, the total number of hops along any route is less than 200.

in the number of nodes in the graph. Due to the computational cost of SPIR, this solution quickly becomes infeasible in the size of the graph.

Instead, we propose a novel method to compress the routing information in street-map networks. Specifically, given a graph $\mathcal{G}$ with n nodes, we define the next-hop routing matrix $M \in \mathbb{Z}^{n \times n}$ for $\mathcal{G}$ to be the matrix where each entry M_{st} gives the index of the first node on the shortest path from node s to node t. To apply our compression method, we first preprocess the graph (Section III) such that each entry in the next-hop routing matrix M can be specified by two bits: $M_{st} = (M_{st}^{(\text{NE})}, M_{st}^{(\text{NW})})$ where $M^{(\text{NE})}, M^{(\text{NW})} \in \{-1, 1\}^{n \times n}$. We then compress $M^{(\text{NE})}$ by computing a *sign-preserving decomposition*: two matrices $A^{(\text{NE})}, B^{(\text{NE})} \in \mathbb{Z}^{n \times d}$ where $d \ll n$ such that $M^{(\text{NE})} = \text{sign}(A^{(\text{NE})} \cdot (B^{(\text{NE})})^T)$. We apply the same procedure to compress the other component $M^{(\text{NW})}$. The resulting compression is lossless, so there is no loss in accuracy in the shortest paths after applying our transformation. When applied to the road network for the city of Los Angeles, we obtain over 10x reduction in the size of the representation. Our compression method is highly parallelizable and by running our computation on GPUs, we can compress next-hop matrices with close to 50 million elements (for a 7000-node network) in under ten minutes.

Moreover, our compression method enables an efficient protocol for a fully-private shortest path computation. In our protocol, the rounds of interaction correspond to the nodes in the shortest path. On each iteration of the protocol, the client learns the next hop on the shortest path to its requested destination. Abstractly, if the client is currently at a node s and navigating to a destination t, then after one round of the protocol execution, the client should learn the next hop given by $M_{st} = (M_{st}^{(\text{NE})}, M_{st}^{(\text{NW})})$. Each round of our protocol thus reduces to a two-party computation of the components $M_{st}^{(\text{NE})}$ and $M_{st}^{(\text{NW})}$. Given our compressed representation of the next-hop routing matrices, computing $M_{st}^{(\text{NE})}$ reduces to computing the sign of the inner product between the s^{th} row of $A^{(\text{NE})}$ and the t^{th} row of $B^{(\text{NE})}$, and similarly for $M^{(\text{NW})}$. In our construction, we give an efficient method for inner product evaluation based on affine encodings, and use Yao's garbled circuits [63], [5] to evaluate the sign function. An important component of our protocol design is a novel way of efficiently combining affine encodings and garbled circuits. Together, these methods enable us to construct an efficient, fully-private navigation protocol.

Other approaches. An alternative method for private navigation is to use generic tools for two-party computation such as Yao's garbled circuits [63], [5] and Oblivious RAM (ORAM) [27], [58]. While these approaches are versatile, they are often prohibitively expensive for city-scale networks (in the case of Yao circuits), or do not provide strong security guarantees against malicious clients (in the case of ORAM). For instance, the garbled-circuit approach by Carter et al. [15], [14] requires several minutes of computation to answer a single shortest path query in a road network with just 100 nodes. Another generic approach combining garbled circuits and ORAM [41] requires communication on the order of GB and run-times ranging from tens of minutes to several hours for a single query on a network with 1024 nodes. Thus, current state-of-the-art tools for general two-party computation do not give a viable solution for private navigation in city-scale networks. We survey other related methods in Section VII.

II. Preliminaries and Threat Model

We begin with some notation. For a positive integer n, let $[n]$ denote the set of integers $\{1, \ldots, n\}$. For two ℓ-bit strings $x, y \in \{0,1\}^\ell$, we write $x \oplus y$ to denote their bitwise XOR. For a prime p, we write $\mathbb{F}_p$ to denote the finite field with p elements, and $\mathbb{F}_p^*$ to denote its multiplicative group. Let $\mathcal{D}$ be a probability distribution. We write $x \leftarrow \mathcal{D}$ to denote that x is drawn from $\mathcal{D}$. Similarly, for a finite set S we write $x \xleftarrow{\text{R}} S$ to denote that x is drawn uniformly at random from S. A function $f(\lambda)$ is negligible in a security parameter λ if $f = o(1/\lambda^c)$ for all $c \in \mathbb{N}$.

For two distribution ensembles $\{\mathcal{D}_1\}_\lambda, \{\mathcal{D}_2\}_\lambda$, we write $\{\mathcal{D}_1\}_\lambda \overset{c}{\approx} \{\mathcal{D}_2\}_\lambda$ to denote that $\{\mathcal{D}_1\}_\lambda$ and $\{\mathcal{D}_2\}_\lambda$ are computationally indistinguishable (i.e., no probabilistic polynomial-time algorithm can distinguish them, except with probability negligible in λ). We write $\{\mathcal{D}_1\}_\lambda \equiv \{\mathcal{D}_2\}_\lambda$ to denote that $\{\mathcal{D}_1\}_\lambda$ and $\{\mathcal{D}_2\}_\lambda$ are identically distributed for all λ. For a predicate $\mathcal{P}(x)$, we write $\mathbf{1}\{\mathcal{P}(x)\}$ to denote the indicator function for $\mathcal{P}(x)$, i.e., $\mathbf{1}\{\mathcal{P}(x)\} = 1$ if and only if $\mathcal{P}(x)$ is true, and otherwise, $\mathbf{1}\{\mathcal{P}(x)\} = 0$. If $\mathcal{G}$ is a directed graph, we write (u, v) to denote the edge from node u to node v.

A function $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ with key-space $\mathcal{K}$, domain $\mathcal{X}$, and range $\mathcal{Y}$ is a PRF [25] if no efficient adversary can distinguish outputs of the PRF (with key $k \xleftarrow{\text{R}} \mathcal{K}$, evaluated on inputs chosen adaptively by the adversary) from the corresponding outputs of a truly random function from $\mathcal{X} \to \mathcal{Y}$.

Threat model. We give a high-level survey of our desired security properties, and defer the details to Section IV-B. We operate in the two-party setting where both parties know the network topology as well as a few generic parameters about the underlying graph structure (described concretely in Section IV-B), but only the server knows the weights (the routing information). The client holds a source-destination pair. At the end of the protocol execution, the client learns the shortest path between its requested source and destination, while the server learns nothing. The first property we require is privacy for the client's location. Because of the sensitivity of location information, we require privacy to hold even against malicious servers, that is, servers whose behavior can deviate from the protocol specification.

The second requirement is privacy for the server's routing information, which may contain proprietary or confidential information. The strongest notion we can impose is that at the end of the protocol execution, the client does not learn anything more about the graph other than the shortest path between its requested source and destination and some generic parameters associated with the underlying network. While this property is not difficult to achieve if the client is semi-honest (that is, the client adheres to the protocol specification), in practice there is little reason to assume that the client will behave this way. Thus, we aim to achieve security against malicious clients. In our setting, we will show that a malicious client learns only the shortest path from its requested source to its requested destination, except for failure events that occur with probability

at most $\approx 2^{-30}$. For comparison, 2^{-30} is the probability that an adversary running in time $\approx 2^{50}$ is able to guess an 80-bit secret key.[2]

To summarize, we desire a protocol that provides privacy against a malicious server and security against a malicious client. We note that our protocol does not protect against the case of a server corrupting the map data; in practice, we assume that the map provider is trying to provide a useful service, and thus is not incentivized to provide misleading or false navigation information.

III. Graph Processing

As described in Section I, we model street-map networks as directed graphs, where nodes correspond to intersections, and edges correspond to streets. To enable an efficient protocol for fully-private shortest path computation, we first develop an efficient method for preprocessing and compressing the routing information in the network. In this section, we first describe our preprocessing procedure, which consists of two steps: introducing dummy nodes to constrain the out-degree of the graph, and assigning a cardinal direction to each edge. Then, we describe our method for compressing the routing information in the graph; here, we exploit the geometric structure of the graph.

Bounding the out-degree. Let $\mathcal{G}$ be the directed graph representing the road network. We assume that the nodes in $\mathcal{G}$ have low out-degree. In a road network (see Figure 1 for an example), the nodes correspond to street intersections, and thus typically have at most four outgoing edges, one in each cardinal direction. In the first step of our preprocessing procedure, we take a weighted, directed graph $\mathcal{G}$ and transform it into a weighted, directed graph $\mathcal{G}'$ where each node has maximum out-degree 4. Specifically, we start by setting $\mathcal{G}' = \mathcal{G}$. Then, as long as there is a node $u \in \mathcal{G}'$ with neighbors $v_1, \ldots, v_\ell$ and $\ell > 4$, we do the following. First, we add a new node u' to $\mathcal{G}'$. For $i \geq 4$, we add the edge (u', v_i) to $\mathcal{G}'$ and remove the edge (u, v_i) from $\mathcal{G}'$. We also add a zero-weight edge from u to u' in $\mathcal{G}'$. By construction, this transformation preserves the shortest-path between nodes in $\mathcal{G}$ and constrains the out-degree of all nodes in $\mathcal{G}'$ to 4.

Orienting the edges. In a road network, we can associate each node by an (x, y) pair in the coordinate plane (for example, the node's latitude and longitude). Consider a coordinate system aligned with the cardinal directions: the x-axis corresponds to the east-west axis and the y-axis corresponds to the north-south axis. Then, for each node u in the graph $\mathcal{G}$, we associate each of its neighbors v_i ($0 \leq i < 4$) with a direction $\mathsf{dir}_i \in \{\text{N}, \text{E}, \text{S}, \text{W}\}$ (for north, east, south, west, respectively) relative to u. For a concrete example, refer to the visualization of the preprocessed graph in Figure 1. Here, the center node (labeled "src") has three neighbors, each of which is associated with a cardinal direction: north, west, or south in this case. We define the orientation of an edge to be the direction associated with the edge.

[2]Even in the case of these low-probability failure events, one can show that a malicious client only learns a bounded-length path emanating from its requested source, though it may not be a shortest path to any particular destination.

To determine the orientation of the edges in $\mathcal{G}$, we proceed as follows. For each node $u \in \mathcal{G}$, we associate a *unique* direction $\mathsf{dir}_i \in \{\text{N}, \text{E}, \text{S}, \text{W}\}$ with each neighbor v_i of u. In assigning the four cardinal directions to each node's neighbors, we would like to approximate the true geographical locations of the nodes. In our setting, we formulate this assignment as a bipartite matching problem for each node u, with u's neighbors (at most 4) forming one partition of the graph, and the four cardinal directions $\{\text{N}, \text{E}, \text{S}, \text{W}\}$ forming the other. We define the cost of a matching between a neighbor v_i of u and a direction dir_j to be the angle formed by the vector from u to v_i and the unit vector aligned in the direction dir_j. In assigning directions to neighbors, we desire a matching that minimizes the costs of the matched neighbors. Such a matching can be computed efficiently using the Hungarian method [35]. In this way, we associate a cardinal direction with each edge in $\mathcal{G}$.

Compressing shortest paths. Next, we describe a method for compressing the next-hop routing matrix for a road network. Let $\mathcal{G}$ be a directed graph with n nodes and maximum out-degree 4. Using the method described above, we associate a direction $\mathsf{dir} \in \{\text{N}, \text{E}, \text{S}, \text{W}\}$ with each edge in $\mathcal{G}$. Since there are four possible values for dir, we can encode the direction using exactly two bits b_{NE} and b_{NW}, where $b_{\text{NE}} = 0$ if and only if $\mathsf{dir} \in \{\text{N}, \text{E}\}$, and $b_{\text{NW}} = 0$ if and only if $\mathsf{dir} \in \{\text{N}, \text{W}\}$. Intuitively, b_{NE} encodes the direction with respect to the northwest-southeast axis while b_{NW} encodes the direction with respect to the northeast-southwest axis. Thus, for each node $u \in \mathcal{G}$, we associate a unique two-bit index $(b_{\text{NE}}, b_{\text{NW}})$ with each of its outgoing edges. For notational convenience, we define a function $\mathsf{IndexToDirection}$ that maps an index $(b_{\text{NE}}, b_{\text{NW}})$ to the corresponding direction $\mathsf{dir} \in \{\text{N}, \text{E}, \text{S}, \text{W}\}$. For example, $\mathsf{IndexToDirection}(0, 0) = \text{N}$.

We next compute the shortest path p_{st} between all source-destination pairs (s, t) in $\mathcal{G}$. In our implementation, we run Dijkstra's algorithm [19] on each node in $\mathcal{G}$, but the precise choice of shortest-path algorithm does not matter for our compression procedure, as its cost is dominated by the other steps. After computing all-pairs shortest paths in $\mathcal{G}$, we define the next-hop routing matrices $M^{(\text{NE})},\ M^{(\text{NW})} \in \{0,1\}^{n \times n}$ for $\mathcal{G}$, where $(M^{(\text{NE})}_{st},\ M^{(\text{NW})}_{st})$ encodes the direction of the first edge in the shortest path p_{st}.

Just as the geometry of road networks enables us to orient the edges, the geometry also suggests a method for compressing the next-hop routing matrices. Take for example the road network in Figure 1. From the visualization, we observe that when the destination t lies to the north of the source s, the first hop on the shortest path is usually to take the edge directed north. In our framework, this means that both $M^{(\text{NE})}_{st}$ and $M^{(\text{NW})}_{st}$ are more likely to be 0 rather than 1. Thus, by orienting the edges in the graph consistently, we find that the resulting routing matrices $M^{(\text{NE})}$ and $M^{(\text{NW})}$ have potentially compressible structure.

To compress a matrix $M \in \{0,1\}^{n \times n}$, we first rescale the elements in M to be in $\{-1, 1\}$. Our goal is to find two matrices $A, B \in \mathbb{Z}^{n \times d}$ such that $\text{sign}(AB^T) = M$ with $d <$

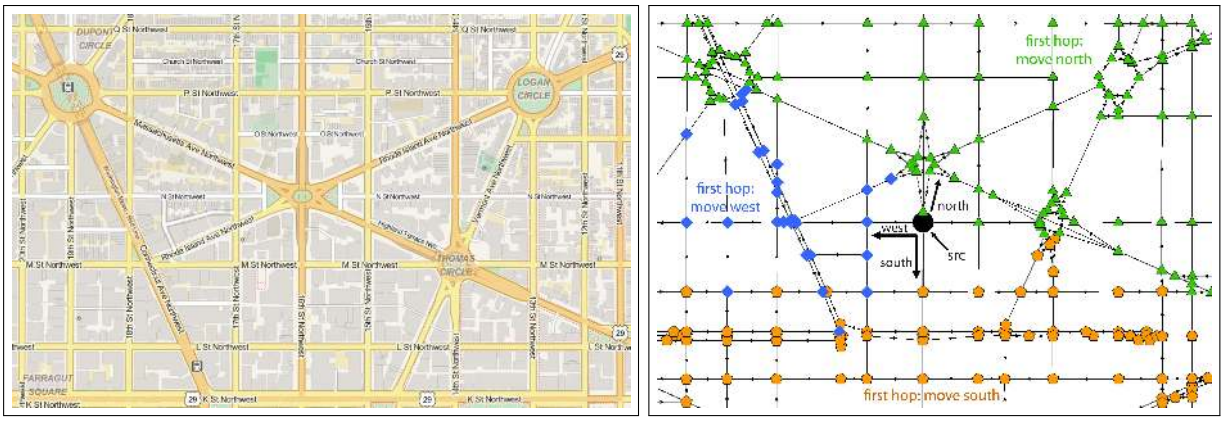

Fig. 1: Subsection of map of Washington, D.C. from OpenStreetMap [49] (left) and visualization of the routing network after preprocessing (right). The visualization on the right shows the first hop of the shortest path from the source node (denoted by the circle) to all other nodes in the graph (denoted by a polygon). In this example, the source node has three neighbors: to the north, west, and south (as indicated in the diagram). If the first hop in the shortest path from the source to a node is to move north, then the node is represented by a green triangle. If the first hop is to move west, then the node is represented by a blue diamond, and if the first hop is to move south, then the node is represented by an orange pentagon.

n.[3] We can formulate the problem of computing A and B as an optimization problem with objective function $J(A, B)$:

$$J(A,B) = \sum_{j=1}^{n}\sum_{k=1}^{n} \ell\left(\left(AB^T\right)_{jk}, M_{jk}\right), \tag{1}$$

where $\ell(x,t)$ is a loss function. A simple loss function is the 0-1 loss function $\ell(x,t) = \mathbf{1}\{\text{sign}(x) \neq t\}$, which assigns a uniform loss of 1 whenever the sign of the predicted value x does not match the target value t. However, from an optimization perspective, the 0-1 loss is not a good loss function since it is non-convex and neither continuous nor differentiable. Practitioners have instead used continuous convex approximations to the 0-1 loss, such as the SVM hinge loss $\ell_{\text{hinge}}(x,t) = \max(0, 1-tx)$ [54] and its quadratically smoothed variant, the modified Huber hinge loss [64]:

$$\ell_{\text{huber}}(x,t) = \begin{cases} \max(0,1-tx)^2 & tx \geq -1 \\ -4 \cdot tx & \text{otherwise.} \end{cases} \tag{2}$$

In our setting, we use the modified Huber hinge loss ℓ_{huber}. While ℓ_{huber} is convex in the input x, it is not convex in the optimization parameters A, B (due to the matrix product), and so the objective function $J(A,B)$ is not convex in A, B. Thus, standard optimization algorithms like LBFGS [11] are not guaranteed to find the global optimum. The hope is that even a local optimum will correspond to a low-rank, sign-preserving decomposition of the matrix M, and indeed, we confirm this empirically.

When we perform the optimization using LBFGS, the matrices A, B are real-valued. To obtain matrices over the integers, we scale the entries in A, B by a constant factor and round. The scaling factor is empirically chosen so as to preserve the relation $\text{sign}(AB^T) = M$. We describe this in greater detail in Section V-C.

[3]This is not the same as computing a low-rank approximation of M. Our goal is to find low-rank matrices whose product preserves the *signs* of the entries of M. In practice, the matrix M is full-rank, and not well-approximated by a low-rank product.

IV. Private Navigation Protocol

In this section, we describe our protocol for privately computing shortest paths. First, we describe the cryptographic building blocks we employ in our construction.

Private information retrieval. A computational private information retrieval (PIR) [12], [18], [36], [16], [23], [39], [50] protocol is a two-party protocol between a sender who holds a database $\mathcal{D} = \{r_1, \ldots, r_n\}$ and a receiver who holds an index $i \in [n]$. At the conclusion of the PIR protocol, the receiver learns r_i while the sender learns nothing. A PIR protocol only ensures privacy for the receiver's index (and not for the remaining records in the sender's database).

Oblivious transfer. Similar to PIR, an 1-out-of-n oblivious transfer (OT) protocol [46], [47], [48], [53] is a two-party protocol that allows the receiver to privately retrieve a record r_i from the sender who holds a database $\{r_1, \ldots, r_n\}$. In contrast with PIR, an OT protocol also provides privacy for the sender: the receiver only learns its requested record r_i, and nothing else about the other records. Closely related is the notion of symmetric PIR (SPIR) [36], [24], [48], which is functionally equivalent to OT.

Garbled circuits. Yao's garbled circuits [63], [38], [5] were initially developed for secure two-party computation. The core of Yao's construction is an efficient transformation that takes a Boolean circuit $C : \{0,1\}^n \to \{0,1\}^m$ and produces a garbled circuit $\tilde{C}$ along with n pairs of encodings $\left\{k_i^0, k_i^1\right\}_{i\in[n]}$. Then, for any input $x \in \{0,1\}^n$, the combination of the garbled circuit $\tilde{C}$ and the encodings $S_x = \left\{k_i^{x_i}\right\}_{i\in[n]}$ (where x_i denotes the i^{th} bit of x) enable one to compute $C(x)$, and yet reveal nothing else about x.

A. Protocol Design Overview

We first give an intuitive overview of our fully-private navigation protocol. As described in Section III, we first preprocess the network $\mathcal{G}$ to have maximum out-degree $d = 4$ and then associate a cardinal direction with each of the edges in $\mathcal{G}$. As in

Section III, let $(M^{(\text{NE})}, M^{(\text{NW})})$ be the precomputed next-hop routing matrices for $\mathcal{G}$, and let $(A^{(\text{NE})}, B^{(\text{NE})}), (A^{(\text{NW})}, B^{(\text{NW})})$ be the compressed representation of $M^{(\text{NE})}, M^{(\text{NW})}$, respectively.

Our private shortest paths protocol is an iterative protocol that reveals the shortest path from a source s to a destination t one hop at a time. When the client engages in the protocol with input (s, t), it learns which neighbor v of s is the next node on the shortest path from s to t. Then, on the next round of the protocol, the client issues a query (v, t) to learn the next node in the path, and so on, until it arrives at the destination node t. With this iterative approach, each round of our protocol can be viewed as a two-party computation of the entry $(M_{st}^{(\text{NE})}, M_{st}^{(\text{NW})})$ from the next-hop routing matrices. We give the full description of our private navigation protocol in Figure 3, and sketch out the important principles here. To simplify the presentation, we first present the core building blocks that suffice for semi-honest security. We then describe additional consistency checks that we introduce to obtain security against a malicious client and privacy against a malicious server.

1) Semi-honest Secure Construction: Abstractly, we can view each round of our protocol as computing the following two-party functionality twice (once for $M^{(\text{NE})}$ and once for $M^{(\text{NW})}$). The server has two matrices $A, B \in \mathbb{Z}^{n \times d}$, which we will refer to as the source and destination matrices, respectively, and the client has two indices $s, t \in [n]$. At the end of the protocol, the client should learn $\text{sign}(\langle A_s, B_t \rangle)$, where A_s and B_t are the s^{th} and t^{th} rows of A and B, respectively. The client should learn nothing else about A and B, while the server should not learn anything. Our protocol can thus be decomposed into two components:

1) Evaluation of the inner product $\langle A_s, B_t \rangle$ between the *source vector* A_s and the *destination vector* B_t.
2) Determining the sign of $\langle A_s, B_t \rangle$.

In the following, we will work over a finite field $\mathbb{F}_p$ large enough to contain the entries in A, B. In particular, we view A, B as $n \times d$ matrices over $\mathbb{F}_p$.

Evaluating the inner product. The first step in our protocol is evaluating the inner product between the source vector A_s and the destination vector B_t. Directly revealing the value of $\langle A_s, B_t \rangle$ to the client, however, leaks information about the entries in the compressed routing matrices A, B. To protect against this leakage, we instead reveal a blinded version of the inner product. Specifically, on each round of the protocol, the server chooses blinding factors $\alpha \xleftarrow{R} \mathbb{F}_p^*$ and $\beta \xleftarrow{R} \mathbb{F}_p$. We then construct the protocol such that at the end of the first step, the client learns the blinded value $\alpha\langle A_s, B_t \rangle + \beta$ instead of $\langle A_s, B_t \rangle$.

One candidate approach for computing the blinded inner product is to use a garbled circuit. However, while Yao's garbled circuits suffice for private evaluation of any two-party functionality, when the underlying operations are more naturally expressed as addition and multiplication over $\mathbb{F}_p$, it is more convenient to express the functionality in terms of an arithmetic circuit. In an arithmetic circuit (over $\mathbb{F}_p$), the "gates" correspond to field operations (addition and multiplication), and the values on the wires correspond to field elements.

In recent work, Applebaum et al. [2] construct the analog of Yao's garbling procedure for arithmetic circuits. In particular, evaluating a function of the form $f(x, y) = \langle x, y \rangle + \sum_{i \in [d]} z_i$, where $x, y \in \mathbb{F}_p^d$ and each $z_i \in \mathbb{F}_p$ is a constant can be done efficiently using the affinization gadgets from [2, §5]. Specifically, for each x_i, y_i, we define the following affine encoding functions $L_{x_i}^{\text{affine}}(x_i), L_{y_i}^{\text{affine}}(y_i)$:

$$L_{x_i}^{\text{affine}}(x_i) = \left(x_i - r_i^{(1)},\ x_i r_i^{(2)} + z_i + r_i^{(3)}\right)$$
$$L_{y_i}^{\text{affine}}(y_i) = \left(y_i - r_i^{(2)},\ y_i r_i^{(1)} - r_i^{(1)} r_i^{(2)} - r_i^{(3)}\right), \quad (3)$$

where $r_i^{(1)}, r_i^{(2)}, r_i^{(3)}$ are chosen uniformly from $\mathbb{F}_p$. We will also write $L_{x_i}^{\text{affine}}(x_i; r_i), L_{y_i}^{\text{affine}}(y_i; r_i)$ to denote affine encodings of x_i and y_i using randomness $r_i \in \mathbb{F}_p^3$. Given $L_{x_i}^{\text{affine}}(x_i)$ and $L_{y_i}^{\text{affine}}(y_i)$ for all $i \in [n]$, evaluating $f(x, y)$ corresponds to evaluating the expression

$$\sum_{i \in [n]} \left[L_{x_i}^{\text{affine}}(x_i)\right]_1 \cdot \left[L_{y_i}^{\text{affine}}(y_i)\right]_1 + \left[L_{x_i}^{\text{affine}}(x_i)\right]_2 + \left[L_{y_i}^{\text{affine}}(y_i)\right]_2, \quad (4)$$

where we write $[\cdot]_i$ to denote the i^{th} component of a tuple. For notational convenience, we also define $L_x^{\text{affine}}(x)$ and $L_y^{\text{affine}}(y)$ as

$$L_x^{\text{affine}}(x) = \left(L_{x_1}^{\text{affine}}(x_1), \ldots, L_{x_d}^{\text{affine}}(x_d)\right)$$
$$L_y^{\text{affine}}(y) = \left(L_{y_1}^{\text{affine}}(y_1), \ldots, L_{y_d}^{\text{affine}}(y_d)\right). \quad (5)$$

Similarly, we write $L_x^{\text{affine}}(x; r), L_y^{\text{affine}}(y; r)$ to denote the affine encoding of vectors $x, y \in \mathbb{F}_p^d$ using randomness $r \in \mathbb{F}_p^{3d}$. The affine encodings $L_x^{\text{affine}}(x), L_y^{\text{affine}}(y)$ provides statistical privacy for the input vectors x, y [2, Lemma 5.1].

Next, we describe how these affine encodings can be used to compute the blinded inner product in the first step of the protocol. At the beginning of each round, the server chooses blinding factors $\alpha \xleftarrow{R} \mathbb{F}_p^*$ and $\beta \xleftarrow{R} \mathbb{F}_p$. Then, it constructs the affine encoding functions $L_x^{\text{affine}}, L_y^{\text{affine}}$ for the function $f_{\alpha,\beta}(x, y) = \langle \alpha x, y \rangle + \beta$ according to Eq. (3). Next, the server prepares two encoding databases $\mathcal{D}_{\text{src}}$ and $\mathcal{D}_{\text{dst}}$ where the s^{th} record in $\mathcal{D}_{\text{src}}$ consists of the affine encodings $L_x^{\text{affine}}(A_s)$ of each source vector, and the t^{th} record in $\mathcal{D}_{\text{dst}}$ consists of $L_y^{\text{affine}}(B_t)$ of each destination vector. To evaluate the blinded inner product, the client performs two SPIR queries: one for the s^{th} record in $\mathcal{D}_{\text{src}}$ to obtain the encodings of A_s and one for the t^{th} record in $\mathcal{D}_{\text{dst}}$ to obtain the encodings of B_t.[4] The client then evaluates the arithmetic circuit using Eq. (4) to obtain $z = f_{\alpha,\beta}(A_s, B_t)$. To a malicious client, without knowledge of α or β, the value $f_{\alpha,\beta}(A_s, B_t)$ appears uniform over $\mathbb{F}_p$ and independent of A_s, B_t.

Determining the sign. To complete the description, it remains to describe a way for the client to learn the sign of the inner product $\langle A_s, B_t \rangle$. The client has the value $z = \alpha\langle A_s, B_t \rangle + \beta$ from the output of the arithmetic circuit while the server knows the blinding factors α, β. Since computing the sign function is equivalent to performing a comparison, arithmetic circuits are unsuitable for the task. Instead, we construct a separate Yao circuit to unblind the inner product and compare it against zero. More specifically, let $g(x, \gamma, \delta) = \mathbf{1}\{[\gamma x + \delta]_p > 0\}$, where

[4] The databases $\mathcal{D}_{\text{src}}$ and $\mathcal{D}_{\text{dst}}$ are each databases over n records (as opposed to n^2 in the straw-man protocol from Section I).

$[\cdot]_p$ denotes reduction modulo p, with output in the interval $(-p/2, p/2)$. Then,

$$g(z, \alpha^{-1}, -\alpha^{-1}\beta) = \text{sign}(A_s, B_t).$$

To conclude the protocol, the server garbles a Boolean circuit C^{unblind} for the unblinding function g to obtain a garbled circuit $\tilde{C}^{\text{unblind}}$ along with a set of encodings L^{unblind}. It sends the garbled circuit to the client, along with encodings of the unblinding coefficients $\gamma = \alpha^{-1}, \delta = \alpha^{-1}\beta$ to the client. The client engages in 1-out-of-2 OTs to obtain the input encodings of z, and evaluates the garbled circuit $\tilde{C}^{\text{unblind}}$ to learn $\text{sign}(\langle A_s, B_t \rangle)$.

2) Enforcing Consistency for Stronger Security: As described, the protocol reveals just a single edge in the shortest path. Repeated iteration of the protocol allows the client to learn the full shortest path. Moreover, since the server's view of the protocol execution consists only of its view in the PIR and OT protocols, privacy of these underlying primitives ensures privacy of the client's location, even against a malicious server.[5]

Security for the server only holds if the client follows the protocol and makes consistent queries on each round. However, a malicious client can request the shortest path for a different source and/or destination on each round, thereby allowing it to learn edges along arbitrary shortest paths of its choosing. To protect against a malicious client, we bind the client to making *consistent* queries across consecutive rounds of the protocol. We say that a sequence of source-destination queries $(s_1, t_1), \ldots, (s_\ell, t_\ell)$ is *consistent* if for all $i \in [\ell]$, $t_1 = t_i$, and $s_{i+1} = v_i$ where v_i is the first node on the shortest path from s_i to t_i.

Consistency for the destinations. To bind the client to a single destination, we do the following. At the beginning of the protocol, for each row $i \in [n]$ in $\mathcal{D}_{\text{dst}}$, the server chooses a symmetric encryption key $k_{\text{dst},i}$. Then, on each round of the protocol, it encrypts the i^{th} record in $\mathcal{D}_{\text{dst}}$ with the key $k_{\text{dst},i}$. Next, at the beginning of the protocol, the client OTs for the key $k_{\text{dst},t}$ corresponding to its destination t. Since this step is performed only once at the beginning of the protocol, the only record in $\mathcal{D}_{\text{dst}}$ that the client can decrypt is the one corresponding to its original destination. Because each record in $\mathcal{D}_{\text{dst}}$ is encrypted under a different key, the client can use a PIR protocol instead of an SPIR protocol when requesting the record from $\mathcal{D}_{\text{dst}}$.

Consistency for the sources. Maintaining consistency between the source queries is more challenging because the source changes each round. We use the fact that the preprocessed graph has out-degree at most four. Thus, on each round, there are at most four possible sources that can appear in a consistent query in the next round.

Our construction uses a semantically-secure symmetric encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$ with key-space $\{0,1\}^\ell$, and a PRF F with domain $\{\text{N}, \text{E}, \text{S}, \text{W}\}$ and range $\{0,1\}^\ell$. On each round of the protocol, the server generates a new set of source keys $k_{\text{src},1}, \ldots, k_{\text{src},n} \in \{0,1\}^\ell$ for encrypting the records in $\mathcal{D}_{\text{src}}$ in the *next* round of the protocol. The server also chooses four PRF keys $k^0_{\text{NE}}, k^1_{\text{NE}}, k^0_{\text{NW}}, k^1_{\text{NW}}$, which are used to derive directional keys $k_{\text{N}}, k_{\text{E}}, k_{\text{S}}, k_{\text{W}}$. Next, for each node $v \in [n]$ in $\mathcal{D}_{\text{src}}$, let v_{dir} be the neighbor of v in direction $\mathsf{dir} \in \{\text{N}, \text{E}, \text{S}, \text{W}\}$ (if there is one). The server augments the v^{th} record in $\mathcal{D}_{\text{src}}$ with an encryption of the source key $k_{\text{src},v_{\mathsf{dir}}}$ under the directional key k_{dir}.

When the client requests record v from $\mathcal{D}_{\text{src}}$, it also obtains encryptions of the keys of the neighbors of v for the *next* round of the protocol. By ensuring the client only learns one of the directional keys, it will only be able to learn the encryption key for a single source node on the next round of the protocol. We achieve this by including the PRF keys $k^0_{\text{NE}}, k^1_{\text{NE}}, k^0_{\text{NW}}, k^1_{\text{NW}}$ used to derive the directional keys as input to the garbled circuit. Then, in addition to outputting the direction, the garbled circuit also outputs the subset of PRF keys needed to derive exactly one of the directional keys $k_{\text{N}}, k_{\text{E}}, k_{\text{S}}, k_{\text{W}}$. This ensures that the client has at most one source key in the next round of the protocol.

Consistency within a round. In addition to ensuring consistency between consecutive rounds of the protocol, we also require that the client's input to the garbled circuit is consistent with the output it obtained from evaluating the affine encodings. To enforce this, we use the fact that the entries of the routing matrices A, B are bounded: there exists τ such that $\langle A_s, B_t \rangle \in [-2^\tau, 2^\tau]$ for all $s, t \in V$. Then, in our construction, we choose the size of the finite field $\mathbb{F}_p$ to be much larger than the size of the interval $[-2^\tau, 2^\tau]$. Recall that the arithmetic circuit computes a blinded inner product $z \leftarrow \alpha\langle A_s, B_t \rangle + \beta$ where α, β are uniform in $\mathbb{F}^*_p$ and $\mathbb{F}_p$, respectively. To unblind the inner product, the server constructs a garbled circuit that first evaluates the function $g_{\gamma,\delta}(z) = \gamma z + \delta$ with $\gamma = \alpha^{-1}$ and $\delta = -\alpha^{-1}\beta$. By construction, γ is uniform over $\mathbb{F}^*_p$ and δ is uniform over $\mathbb{F}_p$. Thus, using the fact that $\left\{g_{\gamma,\delta}(z) \mid \gamma \in \mathbb{F}^*_p, \delta \in \mathbb{F}_p\right\}$ is a pairwise independent family of functions, we conclude that the probability that $g_{\gamma,\delta}(z') \in [-2^\tau, 2^\tau]$ is precisely $2^{\tau+1}/p$ for all $z' \in \mathbb{F}_p$. By choosing $p \gg 2^{\tau+1}$, we can ensure that the adversary cannot successfully cheat except with very small probability.

Lastly, we remark that when the client issues a query (s, t) where $s = t$, the protocol should not reveal the key for any other node in the graph. To address this, we also introduce an equality test into the garbled circuit such that on input $s = t$, the output is $\perp$. We give a complete specification of the neighbor-computation function that incorporates these additional consistency checks in Figure 2.

B. Security Model

In this section, we formally specify our security model. To define and argue the security of our protocol, we compare the protocol execution in the real-world (where the parties interact according to the specification given in Figure 3) to an execution in an ideal world where the parties have access to a trusted party that computes the shortest path. Following the conventions in [13], we view the protocol execution as occurring in the presence of an adversary $\mathcal{A}$ and coordinated by an environment $\mathcal{E} = \{\mathcal{E}\}_\lambda$ (modeled as a family of polynomial size circuits parameterized by a security parameter λ). The

[5] While a malicious server can send the client malformed circuits or induce selective failure attacks, the server does not receive any output during the protocol execution nor does the client abort the protocol when malformed input is received. Thus, we achieve privacy against a malicious server.

Fix a security parameter λ and a statistical security parameter μ. Let $\mathcal{G} = (V, E)$ be a weighted directed graph with n vertices, such that the out-degree of every vertex is at most 4. The client's input to the protocol consists of two nodes, $s, t \in V$, representing the source and destination of the shortest path the client is requesting. The server's inputs are the compressed routing matrices $A^{(\text{NE})}, B^{(\text{NE})}, A^{(\text{NW})}, B^{(\text{NW})} \in \mathbb{Z}^{n \times d}$ (as defined in Section III).

We assume the following quantities are public and known to both the client and the server: the structure of the graph $\mathcal{G}$ (but not the edge weights); the number of columns d in the compressed routing matrices; a bound on the bit-length τ of the values in the products $A^{(\text{NE})} \cdot (B^{(\text{NE})})^T$ and $A^{(\text{NW})} \cdot (B^{(\text{NW})})^T$; and the total number of rounds R.

In the following description, let $(\mathsf{Enc}, \mathsf{Dec})$ be a CPA-secure symmetric encryption scheme with key space $\{0,1\}^\ell$, and let $F : \{0,1\}^\rho \times \{\text{N}, \text{E}, \text{S}, \text{W}\} \to \{0,1\}^\ell$ be a PRF (where $\ell, \rho = \mathsf{poly}(\lambda)$). Fix a prime-order finite field $\mathbb{F}_p$ such that $p > 2^{\tau+\mu+1}$.

Setup:

1) For each $i \in [n]$, the server chooses independent symmetric encryption keys $k^{(1)}_{\text{src},i},\ k_{\text{dst},i} \xleftarrow{\text{R}} \{0,1\}^\ell$.
2) The client and the server engage in two 1-out-of-n OT protocols with the client playing the role of the receiver:
   - The client requests the s^{th} record from the server's database $(k^{(1)}_{\text{src},1}, \ldots, k^{(1)}_{\text{src},n})$, receiving a value $\hat{k}^{(1)}_{\text{src}}$.
   - The client requests the t^{th} record from the server's database $(k_{\text{dst},1}, \ldots, k_{\text{dst},n})$, receiving a value $\hat{k}_{\text{dst}}$.

For each round $r = 1, \ldots, R$ of the protocol:

1) The server chooses blinding factors $\alpha_{\text{NE}}, \alpha_{\text{NW}} \xleftarrow{\text{R}} \mathbb{F}_p^*$ and $\beta_{\text{NE}}, \beta_{\text{NW}} \xleftarrow{\text{R}} \mathbb{F}_p$. Next, let $\gamma_{\text{NE}} = \alpha_{\text{NE}}^{-1}$ and $\delta_{\text{NE}} = -\alpha_{\text{NE}}^{-1}\beta_{\text{NE}} \in \mathbb{F}_p$. Define γ_{NW} and δ_{NW} analogously.
2) Let $f_{\text{NE}}, f_{\text{NW}} : \mathbb{F}_p^d \times \mathbb{F}_p^d \to \mathbb{F}_p$ where $f_{\text{NE}}(x, y) = \langle \alpha_{\text{NE}} x, y \rangle + \beta_{\text{NE}}$ and $f_{\text{NW}}(x, y) = \langle \alpha_{\text{NW}} x, y \rangle + \beta_{\text{NW}}$. The server then does the following:
   - Apply the affine encoding algorithm (Eq. 3) to f_{NE} to obtain encoding functions $L^{\text{affine}}_{\text{NE},x}, L^{\text{affine}}_{\text{NW},y}$, for the inputs x and y, respectively.
   - Apply the affine encoding algorithm to f_{NW} to obtain encoding functions $L^{\text{affine}}_{\text{NW},x}, L^{\text{affine}}_{\text{NW},y}$.
3) Let C^{unblind} be a Boolean circuit for computing the neighbor-computation function in Figure 2. The server runs Yao's garbling algorithm on C^{unblind} to obtain a garbled circuit $\tilde{C}^{\text{unblind}}$ along with encoding functions L^{unblind}_x, for each of the inputs x to the neighbor-computation function in Figure 2.
4) The server chooses symmetric encryption keys $k^{(r+1)}_{\text{src},1}, \ldots, k^{(r+1)}_{\text{src},n} \xleftarrow{\text{R}} \{0,1\}^\ell$. These are used to encrypt the contents of the source database on the next round of the protocol.
5) The server chooses four PRF keys $k^0_{\text{NE}}, k^1_{\text{NE}}, k^0_{\text{NW}}, k^1_{\text{NW}} \xleftarrow{\text{R}} \{0,1\}^\rho$, two for each axis. Then, the server defines the encryption keys for each direction as follows:

$$k_{\text{N}} = F(k^0_{\text{NE}}, \text{N}) \oplus F(k^0_{\text{NW}}, \text{N}), \quad k_{\text{E}} = F(k^0_{\text{NE}}, \text{E}) \oplus F(k^1_{\text{NW}}, \text{E}), \quad k_{\text{S}} = F(k^1_{\text{NE}}, \text{S}) \oplus F(k^1_{\text{NW}}, \text{S}), \quad k_{\text{W}} = F(k^1_{\text{NE}}, \text{W}) \oplus F(k^0_{\text{NW}}, \text{W}).$$

6) The server prepares the source database $\mathcal{D}_{\text{src}}$ as follows. For each node $u \in [n]$, the u^{th} record in $\mathcal{D}_{\text{src}}$ is an encryption under $k^{(r)}_{\text{src},u}$ of the following:
   - The arithmetic circuit encodings $L^{\text{affine}}_{\text{NE},x}(A^{(\text{NE})}_u)$, $L^{\text{affine}}_{\text{NW},x}(A^{(\text{NW})}_u)$ of the source vectors $A^{(\text{NE})}_u$ and $A^{(\text{NW})}_u$.
   - The garbled circuit encodings $L^{\text{unblind}}_s(u)$ of the source node u.
   - Encryptions of the source keys for the neighbors of u in the next round of the protocol under the direction keys:

$$\kappa_{\text{N}} = \mathsf{Enc}(k_{\text{N}}, k^{(r+1)}_{\text{src},v_{\text{N}}}), \quad \kappa_{\text{E}} = \mathsf{Enc}(k_{\text{E}}, k^{(r+1)}_{\text{src},v_{\text{E}}}), \quad \kappa_{\text{S}} = \mathsf{Enc}(k_{\text{S}}, k^{(r+1)}_{\text{src},v_{\text{S}}}), \quad \kappa_{\text{W}} = \mathsf{Enc}(k_{\text{W}}, k^{(r+1)}_{\text{src},v_{\text{W}}}),$$

     where $v_{\text{N}}, v_{\text{E}}, v_{\text{S}}, v_{\text{W}}$ is the neighbor of u to the north, east, south, or west, respectively. If u does not have a neighbor in a given direction $\mathsf{dir} \in \{\text{N}, \text{E}, \text{S}, \text{W}\}$, then define $k^{(r+1)}_{\text{src},v_{\mathsf{dir}}}$ to be the all-zeroes string 0^ℓ.
7) The server prepares the destination database $\mathcal{D}_{\text{dst}}$ as follows. For each node $u \in [n]$, the u^{th} record in $\mathcal{D}_{\text{dst}}$ is an encryption under $k_{\text{dst},u}$ of the following:
   - The arithmetic circuit encodings $L^{\text{affine}}_{\text{NE},y}(B^{(\text{NE})}_u)$, $L^{\text{affine}}_{\text{NW},y}(B^{(\text{NW})}_u)$ of the destination vectors $B^{(\text{NE})}_u$ and $B^{(\text{NW})}_u$.
   - The garbled circuit encodings $L^{\text{unblind}}_t(u)$ of the destination node u.
8) The client and server engage in two PIR protocols with the client playing role of receiver:
   - The client requests record s from the server's database $\mathcal{D}_{\text{src}}$ and obtains a record $\hat{c}_{\text{src}}$.
   - The client requests record t from the server's database $\mathcal{D}_{\text{dst}}$ and obtains a record $\hat{c}_{\text{dst}}$.
9) The client decrypts the records: $\hat{r}_{\text{src}} \leftarrow \mathsf{Dec}(\hat{k}^{(r)}_{\text{src}}, \hat{c}_{\text{src}})$ and $\hat{r}_{\text{dst}} \leftarrow \mathsf{Dec}(\hat{k}_{\text{dst}}, \hat{c}_{\text{dst}})$:
   - It parses $\hat{r}_{\text{src}}$ into two sets of arithmetic circuit encodings $\hat{L}^{\text{affine}}_{\text{NE},x}$ and $\hat{L}^{\text{affine}}_{\text{NW},x}$, a set of garbled circuit encodings $\hat{L}^{\text{unblind}}_s$, and four encryptions $\hat{\kappa}_{\text{N}}, \hat{\kappa}_{\text{E}}, \hat{\kappa}_{\text{S}}, \hat{\kappa}_{\text{W}}$ of source keys for the next round.
   - It parses $\hat{r}_{\text{dst}}$ into two sets of arithmetic circuit encodings for $\hat{L}^{\text{affine}}_{\text{NE},y}$ and $\hat{L}^{\text{affine}}_{\text{NW},y}$, and a set of garbled circuit encodings $\hat{L}^{\text{unblind}}_t$.

   Using the encodings $\hat{L}^{\text{affine}}_{\text{NE},x}$ and $\hat{L}^{\text{affine}}_{\text{NE},y}$, the client evaluates the arithmetic circuit (Eq. 4) to learn $\hat{z}_{\text{NE}}$. Similarly, using the encodings $\hat{L}^{\text{affine}}_{\text{NW},x}$ and $\hat{L}^{\text{affine}}_{\text{NW},y}$, the server evalautes to learn $\hat{z}_{\text{NW}}$. If the parsing of $\hat{r}_{\text{src}}$ or $\hat{r}_{\text{dst}}$ fails or the arithmetic circuit encodings are malformed, the client sets $\hat{z}_{\text{NE}}, \hat{z}_{\text{NW}} \xleftarrow{\text{R}} \mathbb{F}_p$.

Fig. 3: The fully-private navigation protocol, as outlined in Section IV. The protocol description continues on the next page.

10) The client engages in a series of 1-out-of-2 OTs with the server to obtain the garbled circuit encodings $L^{\text{unblind}}_{z_{\text{NE}}}(\hat{z}_{\text{NE}})$ and $L^{\text{unblind}}_{z_{\text{NW}}}(\hat{z}_{\text{NW}})$ of $\hat{z}_{\text{NE}}$ and $\hat{z}_{\text{NW}}$, respectively. Let $\hat{L}^{\text{unblind}}_{z_{\text{NE}}}$ and $\hat{L}^{\text{unblind}}_{z_{\text{NW}}}$ denote the encodings the client receives.
11) The server sends to the client the garbled circuit $\tilde{C}^{\text{unblind}}$ and encodings of the unblinding coefficients

$$L^{\text{unblind}}_{\gamma_{\text{NE}}}(\gamma_{\text{NE}}),\ L^{\text{unblind}}_{\gamma_{\text{NW}}}(\gamma_{\text{NW}}),\ L^{\text{unblind}}_{\delta_{\text{NE}}}(\delta_{\text{NE}}),\ L^{\text{unblind}}_{\delta_{\text{NW}}}(\delta_{\text{NW}}),$$

as well as encodings of the PRF keys

$$L^{\text{unblind}}_{k^0_{\text{NE}}}(k^0_{\text{NE}}),\ L^{\text{unblind}}_{k^1_{\text{NE}}}(k^1_{\text{NE}}),\ L^{\text{unblind}}_{k^0_{\text{NW}}}(k^0_{\text{NW}}),\ L^{\text{unblind}}_{k^1_{\text{NW}}}(k^1_{\text{NW}}).$$

12) The client evaluates the garbled circuit $\tilde{C}^{\text{unblind}}$. If the garbled circuit evaluation is successful and the client obtain outputs $(\hat{b}_{\text{NE}}, \hat{b}_{\text{NW}}, \hat{k}_{\text{NE}}, \hat{k}_{\text{NW}})$, then the client computes a direction $\mathsf{dir} = \mathsf{IndexToDirection}(\hat{b}_{\text{NE}}, \hat{b}_{\text{NW}}) \in \{\text{N}, \text{E}, \text{S}, \text{W}\}$ (Section III).
   a) The client computes the direction key $\hat{k}_{\mathsf{dir}} = F(\hat{k}_{\text{NE}}, \mathsf{dir}) \oplus F(\hat{k}_{\text{NW}}, \mathsf{dir})$. Next, the client decrypts the encrypted source key $\hat{\kappa}_{\mathsf{dir}}$ to obtain the source key $\hat{k}^{(r+1)}_{\text{src}} = \mathsf{Dec}(\hat{k}_{\mathsf{dir}}, \hat{\kappa}_{\mathsf{dir}})$ for the next round of the protocol.
   b) Let v_{dir} be the neighbor of s in the direction given by dir (define v_{dir} to be $\perp$ if s does not have a neighbor in the direction dir). If $v_{\mathsf{dir}} \neq \perp$, the client outputs v_{dir} and updates $s = v_{\mathsf{dir}}$. Otherwise, if $v_{\mathsf{dir}} = \perp$, the client outputs $\perp$ and leaves s unchanged.

   If the OT for the input wires to the garbled circuit fails, the garbled circuit evaluation fails, or the output of the garbled circuit is $\perp$, then the client outputs $\perp$, but continues with the protocol: it leaves s unchanged and sets $\hat{k}^{(r+1)}_{\text{src}} \xleftarrow{\text{R}} \{0,1\}^{\ell}$.

Fig. 3 (Continued): The complete secure private routing protocol, as outlined in Section IV.

Inputs: Tuples $(z_{\text{NE}}, \gamma_{\text{NE}}, \delta_{\text{NE}}), (z_{\text{NW}}, \gamma_{\text{NW}}, \delta_{\text{NW}}) \in \mathbb{F}_p^3$, PRF keys $k^0_{\text{NE}}, k^1_{\text{NE}}, k^0_{\text{NW}}, k^1_{\text{NW}} \in \{0,1\}^{\rho}$, and the source and destination nodes $s, t \in [n]$. The bit-length τ is public and fixed (hard-wired into g).

Operation of g:

- If $s = t$, then output $\perp$.
- If $[\gamma_{\text{NE}} z_{\text{NE}} + \delta_{\text{NE}}]_p \notin [-2^\tau, 2^\tau]$ or $[\gamma_{\text{NW}} z_{\text{NW}} + \delta_{\text{NW}}]_p \notin [-2^\tau, 2^\tau]$, output $\perp$.
- Let $b_{\text{NE}} = \mathbf{1}\{[\gamma_{\text{NE}} z_{\text{NE}} + \delta_{\text{NE}}]_p > 0\}$, and let $b_{\text{NW}} = \mathbf{1}\{[\gamma_{\text{NW}} z_{\text{NW}} + \delta_{\text{NW}}]_p > 0\}$. Output $(b_{\text{NE}}, b_{\text{NW}}, k^{b_{\text{NE}}}_{\text{NE}}, k^{b_{\text{NW}}}_{\text{NW}})$.

Fig. 2: Neighbor-computation function g for the private routing protocol.

environment $\mathcal{E}$ is responsible for choosing the inputs to the protocol execution and plays the role of distinguisher between the real and ideal experiments.

As specified in Figure 3, we assume that the following quantities are public to the protocol execution: the topology of the network $\mathcal{G} = (V, E)$, the number of columns d in the compressed routing matrices, a bound on the bit-length τ of the values in the matrix products $A^{(\text{NE})} \cdot (B^{(\text{NE})})^T$ and $A^{(\text{NW})} \cdot (B^{(\text{NW})})^T$, and the total number of rounds R (i.e., the number of hops in the longest possible shortest path). We now define the real and ideal models of execution.

Definition IV.1 (Real Model of Execution)**.** Let π be a private navigation protocol. In the real world, the parties interact according to the protocol specification π. Let $\mathcal{E}$ be the environment and let $\mathcal{A}$ be an adversary that corrupts either the client or the server. The protocol execution in the real world proceeds as follows:

1) **Inputs:** The environment $\mathcal{E}$ chooses a source-destination pair $s, t \in V$ for the client and compressed next-hop routing matrices $A^{(\text{NE})}, B^{(\text{NE})}, A^{(\text{NW})}, B^{(\text{NW})} \in \mathbb{Z}^{n \times d}$ for the server. The bit-length of all entries in the matrix products $A^{(\text{NE})} \cdot (B^{(\text{NE})})^T$ and $A^{(\text{NW})} \cdot (B^{(\text{NW})})^T$ must be at most τ. Finally, the environment gives the input of the corrupted party to the adversary.
2) **Protocol Execution:** The parties begin executing the protocol. All honest parties behave according to the protocol specification. The adversary $\mathcal{A}$ has full control over the behavior of the corrupted party and sees all messages received by the corrupted party.
3) **Output:** The honest party computes and gives its output to the environment $\mathcal{E}$. The adversary computes a function of its view of the protocol execution and gives it to $\mathcal{E}$.

At the conclusion of the protocol execution, the environment $\mathcal{E}$ outputs a bit $b \in \{0,1\}$. Let $\mathsf{REAL}_{\pi,\mathcal{A},\mathcal{E}}(\lambda)$ be the random variable corresponding to the value of this bit.

Definition IV.2 (Ideal Model of Execution)**.** In the ideal world, the client and server have access to a trusted party $\mathcal{T}$ that computes the shortest paths functionality f.

1) **Inputs:** Same as in the real model of execution.
2) **Submission to Trusted Party:** If a party is honest, it gives its input to the trusted party. If a party is corrupt, then it can send any input of its choosing to $\mathcal{T}$, as directed by the adversary $\mathcal{A}$.
3) **Trusted Computation:** From the next-hop routing matrices, the trusted party computes the first R hops on the shortest path from s to t: $s = v_0, v_1, \ldots, v_R$. If $v_i = t$ for some $i < R$, then the trusted party sets $v_{i+1}, \ldots, v_R$ to $\perp$. If the next hop in the path at v_i for some i refers to a node not in $\mathcal{G}$, then the trusted party sets $v_{i+1}, \ldots, v_R$ to $\perp$. The trusted party sends the path $v_0, \ldots, v_R$ to the client. The server receives no output.
4) **Output:** An honest party gives the sequence of messages (possibly empty) it received from $\mathcal{T}$ to $\mathcal{E}$. The adversary computes a function of its view of the protocol execution and gives it to $\mathcal{E}$.

At the conclusion of the protocol execution, the environment $\mathcal{E}$ outputs a bit $b \in \{0,1\}$. Let $\mathsf{IDEAL}_{f,\mathcal{A},\mathcal{E}}(\lambda)$ be the random

variable corresponding to the value of this bit.

To state our security theorems, we now define the environment's distinguishing advantage. Informally, we will say that a protocol is secure if no polynomial-size environment is able to distinguish the real execution from the ideal execution with non-negligible probability.

Definition IV.3 (Distinguishing Advantage — Security)**.** Let π be a private navigation protocol, and let f be the shortest path functionality. Fix an adversary $\mathcal{A}$, simulator $\mathcal{S}$, and an environment $\mathcal{E}$. The distinguishing advantage $\mathsf{Adv}^{(\text{sec})}_{\pi,f,\mathcal{A},\mathcal{S},\mathcal{E}}(\lambda)$ of $\mathcal{E}$ in the security game is given by

$$\left|\Pr[\mathsf{REAL}_{\pi,\mathcal{A},\mathcal{E}}(\lambda)=0]-\Pr[\mathsf{IDEAL}_{f,\mathcal{A},\mathcal{E}}(\lambda)=0]\right|.$$

We will also work with a weaker notion of *privacy* against a malicious adversary. Informally, we say that the protocol is private if an adversary is unable to learn anything about the inputs of the other party beyond what is explicitly leaked by the inputs and outputs of the computation. To formalize this notion, we use the conventions in [31] and define the distinguishing advantage in the privacy game.

Definition IV.4 (Distinguishing Advantage — Privacy)**.** Let π be a private navigation protocol, and let f be the shortest path functionality. Fix an adversary $\mathcal{A}$, simulator $\mathcal{S}$, and an environment $\mathcal{E}$. Define $\mathsf{REAL}'_{\pi,\mathcal{A},\mathcal{E}}(\lambda)$ exactly as $\mathsf{REAL}_{\pi,\mathcal{A},\mathcal{E}}(\lambda)$ (Definition IV.1), except in the final step of the protocol execution, the environment only receives the adversary's output (and *not* the honest party's output). Define $\mathsf{IDEAL}'_{f,\mathcal{S},\mathcal{E}}(\lambda)$ analogously. The distinguishing advantage $\mathsf{Adv}^{(\text{priv})}_{\pi,f,\mathcal{A},\mathcal{S},\mathcal{E}}(\lambda)$ of $\mathcal{E}$ in the privacy game is given by

$$\left|\Pr[\mathsf{REAL}'_{\pi,\mathcal{A},\mathcal{E}}(\lambda)=0]-\Pr[\mathsf{IDEAL}'_{f,\mathcal{A},\mathcal{E}}(\lambda)=0]\right|.$$

C. Security Theorems

The first requirement is that our protocol provides security against a malicious client. This captures the notion that a malicious client does not learn anything more about the server's routing information beyond the shortest path between its requested endpoints and the publicly available information. In our setting, we allow a privacy-performance trade-off where the client has a small probability ($R \cdot 2^{-\mu}$, where μ is the statistical security parameter) of learning additional information about the routing information. Since the order p of the finite field must satisfy $p > 2^{\tau+\mu+1}$, using larger finite fields will decrease the failure probability, but at the expense of performance. In our experiments, $R \cdot 2^{-\mu} \approx 2^{-30}$. We now state the formal security guarantee, but defer its formal proof to the extended version of this paper.

Theorem IV.5 (Security Against a Malicious Client)**.** *Let π be the protocol in Figure 3 instantiated with a CPA-secure encryption scheme $(\mathsf{Enc},\mathsf{Dec})$, a secure PRF F, and an OT scheme secure against a malicious client. Let λ,μ be the security parameter and statistical security parameter, respectively. Let f be the ideal shortest-paths functionality. Then, for all* PPT *adversaries $\mathcal{A}$, there exists a* PPT *adversary $\mathcal{S}$ such that for every polynomial-size circuit family $\mathcal{E}=\{\mathcal{E}\}_\lambda$,*

$$\mathsf{Adv}^{(sec)}_{\pi,f,\mathcal{A},\mathcal{S},\mathcal{E}}(\lambda) \le \mathsf{negl}(\lambda) + R\cdot 2^{-\mu},$$

where $\mathsf{negl}(\lambda)$ denotes a negligible function in λ.

In addition to security against a malicious client, we require our protocol to provide privacy against a malicious server. In other words, while a malicious server might be able to cause the client to receive an invalid path, it still cannot learn any information about the client's source or destination. We formalize this in the following theorem.

Theorem IV.6 (Privacy Against a Malicious Server)**.** *Let π be the protocol in Figure 3 instantiated with PIR and OT primitives that provide privacy against a malicious server. Let λ be a security parameter and let f be the ideal shortest-paths functionality. Then, for all* PPT *adversaries $\mathcal{A}$, there exists a* PPT *adversary $\mathcal{S}$ such that for every polynomial-size circuit family $\mathcal{E}=\{\mathcal{E}\}_\lambda$,*

$$\mathsf{Adv}^{(priv)}_{\pi,f,\mathcal{A},\mathcal{S},\mathcal{E}}(\lambda) \le \mathsf{negl}(\lambda),$$

where $\mathsf{negl}(\lambda)$ denotes a negligible function in λ.

Proof (Sketch): We give a sketch of the proof, and defer the full argument to the extended version of this paper. We argue that the server's view of the protocol execution can be simulated independently of the client's input. At a high-level, this follows from the fact that the server's view in the protocol execution consists only of its view in OT and PIR protocols. By assumption, privacy of the OT and PIR protocols implies the existence of a simulator that can simulate the server's view of the OT or PIR protocol independently of the client's input. Thus, for any adversarial server $\mathcal{A}$ in the real-world, we can construct a simulator $\mathcal{S}$ that is able to simulate a view computationally indistinguishable from that of $\mathcal{A}$ in the real protocol (by invoking the underlying PIR and OT simulators for each sub-protocol). ■

V. Experiments

In this section, we describe our implementation of the private routing protocol from Figure 3. Then, we describe our procedure for preprocessing and compressing actual road networks for major cities taken from OpenStreetMap [49]. Finally, we give concrete performance benchmarks for our preprocessing and compression pipeline as well as our private routing protocol on actual road networks.

A. Protocol Implementation

To evaluate the performance of the protocol in Figure 3, we implemented the complete protocol in C++. In this section, we describe the building blocks of our implementation. For each primitive, we choose the parameters to guarantee a minimum of 80 bits of security. The complete protocol implementation contains approximately 4000 lines of code.

PIR. We implemented the (recursive) PIR protocol based on additive homomorphic encryption from [36], [50]. We instantiate the additive homomorphic encryption scheme with Paillier's cryptosystem [51], and use NTL [56] over GMP [28] to implement the necessary modular arithmetic. We use a 1024-bit RSA modulus for the plaintext space in the Paillier cryptosystem, which provides 80 bits of security. We use two levels of recursion in the PIR protocol, so the communication scales as $O(\sqrt[3]{n})$ for an n-record database.

OT. We instantiate the OT protocol with the protocol from [29, §7.3] which provides security against malicious clients and privacy against malicious servers. This protocol is a direct generalization of the Naor-Pinkas OT protocol [47] based on the decisional Diffie-Hellman (DDH) assumption. Security against a malicious client is enforced by having the client include a zero-knowledge proof of knowledge (specifically, a Schnorr proof [55]) with its OT request. To decrease the number of rounds of communication, we apply the Fiat-Shamir heuristic [21] to transform the interactive proof of knowledge into a non-interactive one by working in the random oracle model. We instantiate the random oracle with the hash function SHA-256. For improved performance, we implement the Naor-Pinkas OT protocol over the 256-bit elliptic curve group `numsp256d1` from [9]. We use the MSR-ECC [9] library for the implementation of the underlying elliptic curve operations. The 256-bit curve provides 128 bits of security.

Arithmetic and Yao's circuits. We implement our arithmetic circuits over the finite field $\mathbb{F}_p$ where $p = 2^{61}-1$ is a Mersenne prime. Then, reductions modulo p can be performed using just two p-bit additions. We use NTL [56] over GMP [28] for the finite field arithmetic.

For the garbled circuit implementation, we use JustGarble [4] with the "free XOR" [34] and row-reduction optimizations [52]. We set the parameters of the garbling framework to obtain 80-bits of security. We use the optimized addition, comparison, and multiplexer circuits from [33] to implement the neighbor-computation function shown in Figure 2. For multiplication, we implement the basic "school method."

Record encryption and PRF. We instantiate the CPA-secure encryption scheme in Figure 3 with AES in counter mode. We also instantiate the PRF used for deriving the neighbor keys (Step 5 in Figure 3) with AES. We use the implementation of AES from OpenSSL [59].

B. Preprocessing and Map Compression.

We extract the street maps for four major cities (San Francisco, Washington, D.C., Dallas, and Los Angeles) from OpenStreetMap [49]. For each city, we take its most important roadways based on annotations in OpenStreetMap, and construct the resulting graph $\mathcal{G}$. Specifically, we introduce a node for each street intersection in the city and an edge for each roadway. We assign edge weights based on the estimated time needed to traverse the associated road segment (computed by taking the length of the segment and dividing by the approximated speed limit along the segment). Using the procedure described in Section III, we preprocess the graph to have out-degree at most 4. We then associate each edge of $\mathcal{G}$ with a cardinal direction by solving the assignment problem from Section III. We use Stachniss' implementation [57] of the Hungarian method [35] to solve this assignment problem.

Given the graph $\mathcal{G}$ corresponding to the road network for each city, we run Dijkstra's algorithm [19] on each node s in $\mathcal{G}$ to compute the shortest path between all pairs of nodes. Then, using the all-pairs shortest paths information, we construct the next-hop routing matrices $(M^{(\text{NE})}, M^{(\text{NW})})$ for $\mathcal{G}$. We remark that we can substitute any all-pairs shortest path algorithm for Dijkstra's in this step. The underlying principle we exploit in the construction of our protocol is the fact that next-hop routing matrices for road networks have a simple compressible structure amenable to cryptography.

City	n	Preprocessing Time (s)	Compression Time (s)
San Francisco	1830	0.625	97.500
Washington, D.C.	2490	1.138	142.431
Dallas	4993	4.419	278.296
Los Angeles	7010	9.188	503.007

TABLE I: Average time to preprocess and compress the next-hop routing matrices for different networks. The second column gives the number of nodes n in each city's road network. The preprocessing time column gives the average time needed to orient the edges, compute all-pairs shortest paths, and construct the next-hop routing matrix for the network. The compression time column gives the average time needed to compress the NE or NW component of the next-hop routing matrices.

Finally, we implement the optimization-based compression approach described in Section III to compress the next-hop routing matrices $M^{(\text{NE})}$ and $M^{(\text{NW})}$. We minimize the objective function from Eq. (1) with the loss function set to the modified Huber hinge loss from Eq. (2). Because of the highly parallelizable nature of the objective function, we write specialized CUDA kernels to evaluate the objective function and its derivative on the GPU. In our experiments, we use the LBFGS optimization algorithm [11] from the Python scientific computation libraries NumPy and SciPy [3] to solve the optimization problem.

C. Experiments

Graph preprocessing and compression. We first measure the time needed to preprocess and compress the next-hop routing matrices for several road networks. The preprocessing time includes the time needed to orient the edges, compute all-pairs shortest paths, and construct the next-hop routing matrix for the network (as described in Section V-B).

We also measure the time needed to compress the resulting next-hop routing matrices for the different networks. Recall that our compression method takes a matrix $M \in \{-1, 1\}^{n \times n}$ and produces two matrices $A, B \in \mathbb{Z}^{n \times d}$ such that $\text{sign}(AB^T)$ is a good approximation of M. Since the modified Huber hinge loss (Eq. 2) is an upper bound on the 0-1 loss function $\ell(x, t) = \mathbf{1}\{\text{sign}(x) = t\}$, when the objective value $J(A, B)$ is less than 1 (where $J(A, B)$ is the objective function in Eq. 1), we have $\text{sign}(AB^T) = M$, i.e., the matrices A, B *perfectly reconstruct* M. The parameter d is the number of columns in the matrices A and B. Because our objective function is non-convex in the variables A and B, LBFGS is neither guaranteed to find the globally optimal solution, nor even to converge in a reasonable number of iterations. As a heuristic for deciding whether a candidate value of d admits a feasible solution that perfectly reconstructs $M^{(\text{NE})}$ and $M^{(\text{NW})}$, we run up to 5000 iterations of LBFGS and check whether the resulting solution gives a perfect reconstruction of M. To determine the most compact representation, we search over a range of possible values for d, and choose the smallest value d that yields a perfect reconstruction of M.

We apply our compression method to the routing matrices for road networks from four cities of varying size. Then, we compare the size of the original matrix M to the size of its compressed representation A, B. The number of bits

City	n	d	ν	τ	Compression Factor
San Francisco	1830	12	10	20	7.63
Washington, D.C.	2490	14	10	19	8.89
Dallas	4993	19	12	23	10.95
Los Angeles	7010	26	12	24	11.23

TABLE II: Parameters for the compressed representation of the road networks for each city: n is the number of nodes in each network, d and ν are the number of columns and the precision, respectively, in the routing matrices $A^{(\text{NE})}, B^{(\text{NE})}, A^{(\text{NW})}, B^{(\text{NW})}$ of the compressed representation, and τ is the maximum number of bits needed to represent an element in the products $A^{(\text{NE})}(B^{(\text{NE})})^T$ and $A^{(\text{NW})}(B^{(\text{NW})})^T$. The last column gives the compression factor attained for each network (ratio of size of uncompressed representation to size of compressed representation).

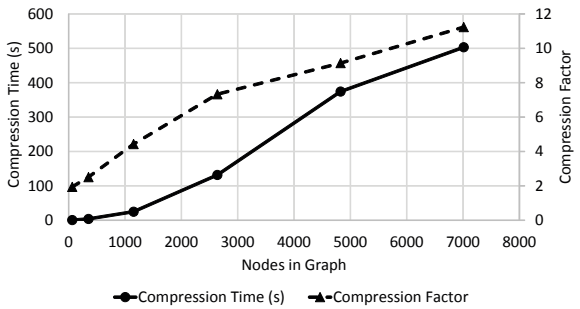

Fig. 4: Average time needed to compress the next-hop routing matrix and the resulting compression factor for networks constructed from subgraphs of the road network of Los Angeles.

needed to represent A, B is determined by two factors: the number of columns d in each matrix A, B and the precision ν (measured in number of bits) needed to represent each entry in A, B. Recall that the optimization procedure outputs two *real-valued* matrices such that $\text{sign}(AB^T) = M$. To obtain a representation over the integers (as required by the arithmetic circuits), we scale the entries of A, B by a constant factor and round each of the resulting entries to the nearest integer. The precision ν is the number of bits needed to represent each integer component of A, B after rescaling. We choose the smallest scaling factor such that the rescaled matrices perfectly reconstruct the routing matrix M.

We run the preprocessing and compression experiments on a machine running Ubuntu 14.04 with an 8-core 2.3 GHz Intel Core i7 CPU, 16 GB of RAM, and an Nvidia GeForce GT 750M GPU. The preprocessing and compression times for the different networks are summarized in Table I. A description of the compressed representation of the routing matrices for each network is given in Table II.

In Figure 4, we show the time needed to compress a single component of the next-hop routing matrix, as well as the resulting compression factor, for subgraphs of the road network for Los Angeles. The compression is quite effective, and the achievable compression factor increases with the size of the network. Moreover, even though the sizes of the next-hop routing matrices increase quadratically in the number of nodes in the graph, the optimization time remains modest. For graphs with 7000 nodes (and 350,000 optimization variables), finding a compact representation that perfectly reconstructs the next-hop matrix completes in under 10 minutes. Since we compress both the NE and NW components of the routing matrix, the total time to both preprocess and compress the shortest path information for the full city of Los Angeles is just over 15 minutes. Lastly, we note that the preprocessing time for each network is small: orienting the edges and computing all-pairs shortest paths via Dijkstra's algorithm completes in under 10 seconds.

Performance on road networks. Next, we measure the run-time and bandwidth requirements of our private routing protocol from Figure 3. Table II gives the number of columns d, and the precision ν of the compressed representation of the networks for the different cities. In addition, we also compute the maximum number of bits τ needed to encode an element in the products $A^{(\text{NE})}(B^{(\text{NE})})^T$ and $A^{(\text{NW})}(B^{(\text{NW})})^T$. From Theorem IV.5, a malicious client can successfully cheat with probability at most $R \cdot 2^{-\mu}$, where μ is the statistical security parameter, and R is the total number of rounds in the protocol. For each network in our experiments, we set the number of rounds R to be the maximum length over all shortest paths between any source-destination pair in the network. This ensures both correctness (at the end of the protocol execution, the client obtains the complete shortest path from its source to its destination) as well as hides the length of the requested shortest path from the server (since the number of rounds in the protocol is independent of the client's input). Next, recall the relation between μ and the order p of the finite field for the affine encodings: $p > 2^{\tau+\mu+1}$. In our experiments, we fix $p = 2^{61} - 1$, and R is at most $2^8 = 256$. These choice of parameters translates to μ ranging from 36 to 41, or analogously, a failure probability of 2^{-33} for the smaller networks to 2^{-28} for larger networks. Using larger fields will reduce this probability, but at the expense of performance.

To reduce the communication in each round of the protocol in Figure 3, we note that it is not necessary for the server to prepare and send a garbled circuit to the client on each round of the routing protocol. Since the neighbor-computation circuit is independent of the state of the protocol execution, the circuits can be generated and stored long before the protocol execution begins. Thus, in an *offline* phase, the server can prepare and transmit to the client a large number of garbled circuits. During the online protocol execution, on the r^{th} round, the server just sends the encodings corresponding to its input to the client; it does *not* send the description of the garbled circuit. This significantly reduces the communication cost of each round of the online protocol. We note that even if the routing matrices $A^{(\text{NE})}, B^{(\text{NE})}, A^{(\text{NW})}, B^{(\text{NW})}$ changed (for instance, due to updates in traffic or weather conditions in the network) during the protocol execution, as long as the bound τ on the bit-length of entries in the products $A^{(\text{NE})}(B^{(\text{NE})})^T$ and $A^{(\text{NW})}(B^{(\text{NW})})^T$ remain fixed, the client and server do *not* have to redo this offline setup phase. We describe our extension for supporting live updates to the routing information in greater detail in Section VI.

We run the server on a compute-optimized Amazon EC2 instance (running Ubuntu 14.04 with a 32-core 2.7 GHz Intel Xeon E5-2680v2 processor and 60 GB of RAM) to model the computing resources of a cloud-based map provider. The throughput of our protocol is bounded by the PIR computation on the server's side. We use up to 60 threads on the server for the PIR computation. All other parts of our system are single-threaded. For the client, we use a laptop running Ubuntu

City	Total Time (s) (Single Round)	Client Computation (s)				Server Computation (s)			Bandwidth (KB)	
		Total	PIR	OT	GC	Total	PIR	OT	Upload	Download
San Francisco	1.44 ± 0.16	0.35	0.31	0.02	0.02	0.88	0.80	0.08	51.74	36.50
Washington, D.C.	1.64 ± 0.13	0.38	0.34	0.02	0.02	1.07	1.00	0.08	52.49	37.51
Dallas	2.91 ± 0.19	0.45	0.41	0.02	0.02	2.19	2.11	0.08	55.50	39.52
Los Angeles	4.75 ± 0.22	0.55	0.51	0.02	0.02	3.70	3.62	0.08	57.01	43.53

TABLE III: Performance benchmarks (averaged over at least 90 iterations) for a single round of the private routing protocol described in Figure 3 on road networks for different cities. The "Total Time" column gives the average runtime and standard deviation for a single round of the protocol (including network communication times between a client and a server on Amazon EC2). The PIR, OT, and GC columns in the table refer to the time to perform the PIR for the affine encodings, the time to perform the OT for the garbled circuit encodings, and the time needed to evaluate the garbled circuit, respectively. The bandwidth measurements are taken with respect to the client ("upload" refers to communication from the client to the server)

City	R	Offline Setup		Online Setup		Total Online Time (s)	Total Online Bandwidth (MB)
		Time (s)	Band. (MB)	Time (s)	Band. (MB)		
San Francisco	97	0.135	49.08	0.73	0.021	140.39	8.38
Washington, D.C.	120	0.170	60.72	0.76	0.023	197.48	10.57
Dallas	126	0.174	63.76	0.92	0.027	371.44	11.72
Los Angeles	165	0.223	83.49	1.00	0.028	784.34	16.23

TABLE IV: End-to-end performance benchmarks for the private routing protocol in Figure 3 on road networks for different cities. For each network, the number of iterations R is set to the maximum length of the shortest path between two nodes in the network. The offline computation refers to the server preparation and garbling of the R circuits for evaluating the neighbor-computation function from Figure 2. The offline computation time just includes the computational cost and does *not* include the garbled circuit download time. The online setup measurements correspond to computation and communication in the "Setup" phase of the protocol in Figure 3. The "Total Online Time" and "Total Online Bandwidth" columns give the total end-to-end time (including network communication) and total communication between the client and server in the *online* phase (navigation component) of the protocol.

14.04 with a 2.3 GHz Intel Core i7 CPU and 16 GB of RAM. The connection speed on the client is around 50 Mbps. Both client and server support the AES-NI instruction set, which we leverage in our implementation.

First, we measure the cost of one round of the private navigation protocol. We assume that the client has already downloaded the garbled circuits prior to the start of the protocol. Table III gives the total computation time and bandwidth per round of the routing protocol. When measuring the total time, we measure the end-to-end time on the client, which includes the time for the network round trips. Table III also gives a breakdown of the computation in terms of each component of the protocol: PIR for the arithmetic circuit encodings, OT for the garbled circuit encodings, and garbled-circuit evaluation for computing the next-hop.

We also measure the total end-to-end costs for a single shortest path query. As noted earlier, we set the number of rounds R for each network to be the maximum length of any shortest path in the network. Irrespective of the client's source or destination, the client and server always engage in exactly R rounds of the private navigation protocol. Table IV shows the total computation time and bandwidth required to complete a shortest-path query in the different networks. In the end-to-end benchmarks, we also measure the offline costs of the protocol, that is, the time needed for the server to garble R neighbor-computation circuits and the amount of communication needed for the client to download the circuits. In addition, we measure the computation and bandwidth needed in the online setup phase of the routing protocol (Figure 3).

In our protocol, the online setup phase of the protocol consists of three rounds of interaction. First, the server sends the client the public description of the map. Then the client OTs for the source and destination keys for the first round of the protocol, which requires two rounds of communication. As shown in Table IV, the online setup procedure completes in at most a second and requires under 30 KB of communication in our example networks.

Next, we consider the performance of each round of the protocol. From Table III, the most computationally intensive component of our protocol is computing the responses to the PIR queries. In our implementation, we use a Paillier-based PIR, so the server must perform $O(n)$ modular exponentiations on each round of the protocol. While it is possible to use a less computationally-intensive PIR such as [42], the bandwidth required is much higher in practice. Nonetheless, our results demonstrate that the performance of our protocol is within the realm of practicality for real-time navigation in cities like San Francisco or Washington, D.C.

Lastly, we note that the offline costs are dominated essentially by communication. With hardware support for AES, garbling 100 neighbor-computation circuits on the server completes in just a quarter of a second. While garbling is fast, the size of each garbled circuit is 518.2 KB. For city networks, we typically require 100-150 circuits for each shortest-path query; this corresponds to 50-100 MB of offline download prior to the start of the navigation protocol. The experimental results, however, indicate that the number of garbled circuits required for an end-to-end execution grows sublinearly in the size of the graph. For example, the total number of rounds (and correspondingly, the number of required garbled circuits) for a graph with 1800 nodes is just under 100, while for a graph with almost four times more nodes, the number of rounds only increases by a factor of 1.7. We also note that each neighbor-computation circuit consists of just under 50,000 non-XOR gates. In contrast, generic protocols for private navigation that construct a garbled circuit for Dijkstra's algorithm yield

circuits that contain hundreds of millions to tens of billions of non-XOR gates [15], [14], [41].

Finally, we see that despite needing to pad the number of rounds to a worst-case setting, the total cost of the protocol remains modest. For the city of Los Angeles, which contains over 7000 nodes, a shortest-path query still completes in under 15 minutes and requires just over 16 MB of total bandwidth. Moreover, since the path is revealed edge-by-edge rather than only at the end of the computation, the overall protocol is an efficient solution for fully-private navigation.

Comparison to other approaches for private navigation. Many protocols [20], [37], [62] have been proposed for private navigation, but most of them rely on heuristics and do not provide strong security guarantees [20], [37], or guarantee privacy only for the client's location, and not the server's routing information [62]. A different approach to fully-private navigation is to leverage generic multiparty computation techniques [63], [26]. For instance, a generic protocol for private navigation is to construct a garbled circuit for a shortest-path algorithm and apply Yao's protocol. This approach is quite expensive since the entire graph structure must be embedded in the circuit. For instance, Liu et al. [41] demonstrate that a garbled circuit for evaluating Dijkstra's algorithm on a graph with just 1024 nodes requires over 10 *billion* AND gates. The bandwidth needed to transmit a circuit of this magnitude quickly grows to the order of GB. In contrast, even for a larger graph with 1800 nodes, the total online and offline communication required by our protocol is under 60 MB (and the online communication is under 10 MB). Carter et al. [15], [14] describe methods for reducing the computational and communicational cost of Yao's protocol by introducing a third (non-colluding) party that facilitates the computation. Even with this improvement, evaluating a single shortest path on a graph of 100 nodes still requires over 10 minutes of computation. As a point of comparison, our protocols complete in around 2-3 minutes for graphs that are 15-20 times larger. Evidently, while the generic tools are powerful, they do not currently yield a practical private navigation protocol. We survey additional related work and techniques in Section VII.

VI. Extensions

In this section, we describe several extensions to our protocol: supporting navigation between cities, handling updates to the routing information, and updating the source node during the protocol execution (for instance, to accommodate detours and wrong turns).

Navigating between cities. The most direct method for supporting navigation across a multi-city region is to construct a network that spans the entire region and run the protocol directly. However, since the server's computation in the PIR protocols grows as $O(nd \log p)$, where n is the number of nodes in the graph, d is the number of columns in the compressed representation, and p is the order of the finite field used for the affine encodings, this can quickly become computationally infeasible for the server.

An alternative method that provides a performance-privacy trade-off is to introduce a series of publicly-known waypoints for each city. For example, suppose a user is navigating from somewhere in Los Angeles to somewhere in San Diego. In this case, the user would first make a private routing request to learn the fastest route from her current location to a waypoint in Los Angeles. Once the user arrives at the waypoint in Los Angeles, she requests the fastest route to a waypoint in San Diego. This second query is performed entirely in the clear, so the user reveals to the server that she is traveling from Los Angeles to San Diego. Once the user arrives at a waypoint in San Diego, she makes a final private routing request to learn the fastest route to her destination. In this solution, the server only obtains a macro-view of the user's location: it learns only the user's source and destination cities, and no information about the user's particular location within the city. As we have demonstrated, the protocol in Figure 3 is able to handle real-time navigation for a single city; thus, using this method of waypoints, we can also apply our protocol to navigation between cities with limited privacy loss.

Live updates to routing information. Routing information in road networks is dynamic, and is influenced by current traffic conditions, weather conditions, and other external factors. Ideally, the edges revealed in an iterative shortest-path protocol should always correspond to the shortest path to the destination given the current network conditions. It is fairly straightforward to allow for updates to the routing information in our protocol. Specifically, we observe that the compressed routing matrices $A^{(\text{NE})}, A^{(\text{NW})}, B^{(\text{NE})}, B^{(\text{NW})}$ need not be fixed for the duration of the protocol. As long as the total number of columns d, the bound on the bit-length τ of the values in the matrix products $A^{(\text{NE})} \cdot (B^{(\text{NE})})^T$ and $A^{(\text{NW})} \cdot (B^{(\text{NW})})^T$, and the total number of rounds R in the protocol remain fixed, the server can use a different set of routing matrices on each round of the protocol. Therefore, we can accommodate live updates to the routing information during the protocol execution by simply setting a conservative upper bound on the parameters d, τ, R. Note that we can always pad a routing matrix with fewer than d columns to one with exactly d columns by adding columns where all entries are 0. Since computing the shortest path information for a city-wide network and compressing the resulting routing matrices completes in just a few minutes, it is possible to ensure accurate and up-to-date routing information in practice.

Updating sources and destinations. Typically, in navigation, the user might take a detour or a wrong turn. While the protocol is designed to constrain the client to learn a single contiguous route through the network, it is possible to provide a functionality-privacy trade-off to accommodate deviations from the actual shortest path. One method is to introduce an additional parameter K, such that after every K iterations of the protocol, the server chooses fresh source keys for the next round of the protocol. After every K rounds, the client would also OT for a new source key. Effectively, we are resetting the protocol every K rounds and allowing the client to choose a new source from which to navigate. Correspondingly, we would need to increase the total number of rounds R in order to support the potential for detours and wrong turns. Though we cannot directly bound the number of rounds R, we can use a conservative estimate. Of course, a dishonest client can now learn multiple sub-paths to its chosen destination, namely, one sub-path each time it is allowed to choose a different source. In a similar manner, we can support updates to the destination.

VII. Related Work

Numerous approaches have been proposed for private shortest path computation [20], [37], [45], [44], [62], [15], [14], [8], [61], [32], [41]. Early works such as [20], [37] propose hiding the client's location by either providing approximate locations to the server [20] or by having the client submit dummy sources and destinations with each shortest path query [37]. However, these approaches only provide limited privacy for the client's location. Later works [44], [45], [62] describe PIR-based solutions for hiding the client's location. In [44], [45], the client first privately retrieves subregions of the graph that are relevant to its query [44], [45] and then locally computes the shortest path over the subgraph. In [62], the client privately requests for columns of the next-hop routing matrix to learn the next hop in the shortest path. While these methods provide privacy for the client's location, they do not hide the server's routing information.

There is also work on developing secure protocols for other graph-theoretic problems and under different models [10], [22]. For example, Brickell and Shmatikov [10] consider a model where two parties hold a graph over a common set of vertices, and the goal is to compute a function over their joint graphs. Their protocols do not extend to navigation protocols where one party holds the full graph, and only the client should learn the result of the computation. In [22], the authors describe protocols for parties who each hold a subset of a graph to privately reconstruct the joint graph. Their methods are designed for social network analysis and do not directly apply to private navigation.

Another line of work has focused on developing data-oblivious algorithms for shortest path computation [8] or combining shortest path algorithms such as Dijkstra's with oblivious data structures or ORAM [61], [32]. In these methods, the routing data is stored in an ORAM or an oblivious data structure on the server. The client then executes the shortest-path algorithm on the server to learn the path between its source and destination. Since the pattern of memory accesses is hidden from the server, these approaches provide client privacy. While these protocols can be efficient in practice, they do not provide security against a malicious client trying to learn additional details about the routing information on the server. Thus, for scenarios where the map data is proprietary (for instance, in the case of real-time traffic routing), or when the routing information itself is sensitive (for instance, when providing navigational assistance for a presidential motorcade or coordinating troop movements in a military scenario [15], [14]), the ORAM-based solutions do not provide sufficient security.

Also relevant are the works in secure multiparty computation (MPC) [63], [26]. While these methods can be successfully used to build private navigation protocols [41], [15], [14], they do not currently yield a practical private navigation protocol. A more comprehensive comparison of our protocol to these generic methods is provided at the end of Section V-C.

There is also a vast literature on graph compression algorithms. For planar graphs, there are multiple methods based on computing graph separators [40], [6], [7]. Other methods based on coding schemes [30] have also been proposed and shown to achieve information-theoretically optimal encoding. While these algorithms are often viable in practice, it is not straightforward to represent them compactly as a Boolean or an arithmetic circuit. Thus, it is unclear how to combine them with standard cryptographic primitives to construct a private shortest path protocol.

Finally, there has also been work on developing compact representations of graphs for answering *approximate* distance queries in graphs [60]. These techniques have been successfully applied for privacy-preserving approximate distance computation on graphs [43]. However, these distance-oracle-based methods only provide an estimate on the *length* of the shortest path, and do not give a private navigation protocol.

VIII. Conclusion

In this work, we constructed an efficient protocol for privately computing shortest paths for navigation. First, we developed a method for compressing the next-hop matrices for road networks by formulating the compression problem as that of finding a sign-preserving, low-rank matrix decomposition. Not only did this method yield a significant compression, it also enabled an efficient cryptographic protocol for fully private shortest-path computation in road networks. By combining affine encodings with Yao's circuits, we obtained a fully-private navigation protocol efficient enough to run at a city-scale.

Acknowledgments

The authors would like to thank Dan Boneh, Roy Frostig, Hristo Paskov, and Madeleine Udell for many helpful comments and discussions. While conducting this work, authors David Wu and Joe Zimmerman were supported by NSF Graduate Research Fellowships. This work was further supported by the DARPA PROCEED research program. Opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA or NSF.

References

[1] J. Angwin and J. Valentino-Devries, "Apple, Google collect user data," The Wall Street Journal, 2011.

[2] B. Applebaum, Y. Ishai, and E. Kushilevitz, "How to garble arithmetic circuits," *SIAM J. Comput.*, vol. 43, no. 2, pp. 905–929, 2014.

[3] D. Ascher, P. F. Dubois, K. Hinsen, J. Hugunin, and T. Oliphant, "Numerical python," Lawrence Livermore National Laboratory, Tech. Rep., 2001.

[4] M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway, "Efficient garbling from a fixed-key blockcipher," in *IEEE Symposium on Security and Privacy*, 2013, pp. 478–492.

[5] M. Bellare, V. T. Hoang, and P. Rogaway, "Foundations of garbled circuits," Cryptology ePrint Archive, Report 2012/265, 2012, http://eprint.iacr.org/.

[6] D. K. Blandford, G. E. Blelloch, and I. A. Kash, "Compact representations of separable graphs," in *SODA*, 2003, pp. 679–688. [Online]. Available: http://dl.acm.org/citation.cfm?id=644108.644219

[7] ——, "An experimental analysis of a compact graph representation," in *Workshop on Analytic Algorithmics and Combinatorics*, 2004, pp. 49–61.

[8] M. Blanton, A. Steele, and M. Aliasgari, "Data-oblivious graph algorithms for secure computation and outsourcing," in *ASIA CCS*, 2013, pp. 207–218.

[9] J. Bos, C. Costello, P. Longa, and M. Naehrig, "Specification of curve selection and supported curve parameters in MSR ECCLib," Microsoft Research, Tech. Rep. MSR-TR-2014-92, June 2014.

[10] J. Brickell and V. Shmatikov, "Privacy-preserving graph algorithms in the semi-honest model," in *ASIACRYPT*, 2005, pp. 236–252.

[11] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A limited memory algorithm for bound constrained optimization," *SIAM J. Comput.*, vol. 16, no. 5, pp. 1190–1208, 1995.

[12] C. Cachin, S. Micali, and M. Stadler, "Computationally private information retrieval with polylogarithmic communication," in *EUROCRYPT*, 1999, pp. 402–414.

[13] R. Canetti, "Security and composition of cryptographic protocols: a tutorial (part I)," *SIGACT News*, vol. 37, no. 3, pp. 67–92, 2006.

[14] H. Carter, C. Lever, and P. Traynor, "Whitewash: outsourcing garbled circuit generation for mobile devices," in *ACSAC*, 2014, pp. 266–275.

[15] H. Carter, B. Mood, P. Traynor, and K. R. B. Butler, "Secure outsourced garbled circuit evaluation for mobile devices," in *USENIX*, 2013, pp. 289–304.

[16] Y. Chang, "Single database private information retrieval with logarithmic communication," in *ACISP*, 2004, pp. 50–61.

[17] J. Cheng, "How Apple tracks your location without consent, and why it matters," Ars Technica, 2011.

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

[19] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[20] M. Duckham and L. Kulik, "A formal model of obfuscation and negotiation for location privacy," in *PERVASIVE*, 2005, pp. 152–170.

[21] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *CRYPTO*, 1986, pp. 186–194.

[22] K. B. Frikken and P. Golle, "Private social network analysis: how to assemble pieces of a graph privately," in *WPES*, 2006, pp. 89–98.

[23] C. Gentry and Z. Ramzan, "Single-database private information retrieval with constant communication rate," in *ICALP*, 2005, pp. 803–815.

[24] Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin, "Protecting data privacy in private information retrieval schemes," *J. Comput. Syst. Sci.*, vol. 60, no. 3, pp. 592–629, 2000.

[25] O. Goldreich, S. Goldwasser, and S. Micali, "How to construct random functions," *J. ACM*, vol. 33, no. 4, pp. 792–807, 1986.

[26] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game or A completeness theorem for protocols with honest majority," in *STOC*, 1987, pp. 218–229.

[27] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious RAMs," *J. ACM*, vol. 43, no. 3, pp. 431–473, 1996.

[28] T. Granlund and the GMP development team, *GNU MP: The GNU Multiple Precision Arithmetic Library*, 5th ed., 2012, http://gmplib.org/.

[29] C. Hazay and Y. Lindell, *Efficient Secure Two-Party Protocols - Techniques and Constructions*, ser. Information Security and Cryptography. Springer, 2010.

[30] X. He, M. Kao, and H. Lu, "A fast general methodology for information-theoretically optimal encodings of graphs," *SIAM J. Comput.*, vol. 30, no. 3, pp. 838–846, 2000.

[31] Y. Ishai, J. Katz, E. Kushilevitz, Y. Lindell, and E. Petrank, "On achieving the "best of both worlds" in secure multiparty computation," *SIAM J. Comput.*, vol. 40, no. 1, pp. 122–141, 2011.

[32] M. Keller and P. Scholl, "Efficient, oblivious data structures for MPC," in *ASIACRYPT*, 2014, pp. 506–525.

[33] V. Kolesnikov, A.-R. Sadeghi, and T. Schneider, "Improved garbled circuit building blocks and applications to auctions and computing minima," Cryptology ePrint Archive, Report 2009/411, 2009, http://eprint.iacr.org/.

[34] V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free XOR gates and applications," in *ICALP*, 2008, pp. 486–498.

[35] H. W. Kuhn and B. Yaw, "The Hungarian method for the assignment problem," *Naval Res. Logist. Quart*, pp. 83–97, 1955.

[36] E. Kushilevitz and R. Ostrovsky, "Replication is NOT needed: SINGLE database, computationally-private information retrieval," in *FOCS*, 1997, pp. 364–373.

[37] K. C. K. Lee, W. Lee, H. V. Leong, and B. Zheng, "Navigational path privacy protection: navigational path privacy protection," in *CIKM*, 2009, pp. 691–700.

[38] Y. Lindell and B. Pinkas, "A proof of security of Yao's protocol for two-party computation," *J. Cryptology*, vol. 22, no. 2, pp. 161–188, 2009.

[39] H. Lipmaa, "An oblivious transfer protocol with log-squared communication," in *ISC*, 2005, pp. 314–328.

[40] R. J. Lipton and R. E. Tarjan, "A separator theorem for planar graphs," *SIAM J. Appl. Math*, no. 2, pp. 177–189, 1979.

[41] C. Liu, X. S. Wang, K. Nayak, Y. Huang, and E. Shi, "ObliVM: A programming framework for secure computation," in *IEEE Symposium on Security and Privacy*, 2015.

[42] C. A. Melchor, J. Barrier, L. Fousse, and M. Killijian, "Xpire: Private information retrieval for everyone," *IACR Cryptology ePrint Archive*, vol. 2014, p. 1025, 2014. [Online]. Available: http://eprint.iacr.org/2014/1025

[43] X. Meng, S. Kamara, K. Nissim, and G. Kollios, "GRECS: graph encryption for approximate shortest distance queries," *IACR Cryptology ePrint Archive*, vol. 2015, p. 266, 2015. [Online]. Available: http://eprint.iacr.org/2015/266

[44] K. Mouratidis, "Strong location privacy: A case study on shortest path queries," in *ICDE*, 2013, pp. 136–143.

[45] K. Mouratidis and M. L. Yiu, "Shortest path computation with no information leakage," *PVLDB*, vol. 5, no. 8, pp. 692–703, 2012.

[46] M. Naor and B. Pinkas, "Oblivious transfer and polynomial evaluation," in *STOC*, 1999, pp. 245–254.

[47] ——, "Efficient oblivious transfer protocols," in *SODA*, 2001, pp. 448–457.

[48] ——, "Computationally secure oblivious transfer," *J. Cryptology*, vol. 18, no. 1, pp. 1–35, 2005.

[49] OpenStreetMap Contributors, "OpenStreetMap," http://www.openstreetmap.org/.

[50] R. Ostrovsky and W. E. S. III, "A survey of single-database private information retrieval: Techniques and applications," in *Public Key Cryptography*, 2007, pp. 393–411.

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

[52] B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams, "Secure two-party computation is practical," in *ASIACRYPT*, 2009, pp. 250–267.

[53] M. O. Rabin, "How to exchange secrets with oblivious transfer," *IACR Cryptology ePrint Archive*, vol. 2005, p. 187, 2005.

[54] L. Rosasco, E. D. Vito, A. Caponnetto, M. Piana, and A. Verri, "Are loss functions all the same?" *Neural Computation*, vol. 16, no. 5, pp. 1063–107, 2004.

[55] C.-P. Schnorr, "Efficient identification and signatures for smart cards," in *CRYPTO*, 1989, pp. 239–252.

[56] V. Shoup, "NTL: A library for doing number theory," http://www.shoup.net/ntl/.

[57] C. Stachniss, "C implementation of the Hungarian method," http://www2.informatik.uni-freiburg.de/~stachnis/misc.html, 2004.

[58] E. Stefanov, M. van Dijk, E. Shi, C. W. Fletcher, L. Ren, X. Yu, and S. Devadas, "Path ORAM: an extremely simple oblivious RAM protocol," in *CCS*, 2013, pp. 299–310.

[59] The OpenSSL Project, "OpenSSL: The open source toolkit for SSL/TLS," April 2003, www.openssl.org.

[60] M. Thorup and U. Zwick, "Approximate distance oracles," in *STOC*, 2001, pp. 183–192. [Online]. Available: http://doi.acm.org/10.1145/380752.380798

[61] X. S. Wang, K. Nayak, C. Liu, T. H. Chan, E. Shi, E. Stefanov, and Y. Huang, "Oblivious data structures," in *CCS*, 2014, pp. 215–226.

[62] Y. Xi, L. Schwiebert, and W. Shi, "Privacy preserving shortest path routing with an application to navigation," *Pervasive and Mobile Computing*, vol. 13, pp. 142–149, 2014.

[63] A. C. Yao, "How to generate and exchange secrets (extended abstract)," in *FOCS*, 1986, pp. 162–167.

[64] T. Zhang, "Solving large scale linear prediction problems using stochastic gradient descent algorithms," in *ICML*, 2004.