Accurate, Secure, and Efficient Semi-Constrained Navigation Over Encrypted City Maps

Meng Li, *Senior Member, IEEE*, Yifei Chen, Jianbo Gao, Jingyu Wu, Zijian Zhang, *Senior Member, IEEE*, Jialing He, Liehuang Zhu, *Senior Member, IEEE*, Mauro Conti, *Fellow, IEEE*, and Xiaodong Lin, *Fellow, IEEE*

***Abstract*—Navigation services enable users to find the shortest path from a starting point S to a destination D, reducing time, gas, and traffic congestion. Still, navigation users risk the exposure of their sensitive location data. Our motivation arises from how users can accurately, securely, and efficiently navigate from S to D while passing through k *unordered stops*, i.e., midway locations with a non-fixed visiting order. In this work, we formally define Semi-Constrained Navigation (SCN) and present a novel scheme Hermes to achieve accurate, secure, and efficient SCN. Specifically, we propose a divide-and-conquer approach to strike a good balance between accuracy and efficiency. It recursively depth-first-searches the whole area (a navigation tree) and invokes five carefully-crafted strategies stop-by-stop to compute three subpaths in three sequential subareas. We construct a path-distance oracle to encrypt the road graph and securely implement the strategies by using homomorphic encryption and garble circuits. We formally prove the security in the random oracle model and analyze the search complexity to be less than $O(k^2)$. We experiment over a real-world city map and compare with six baselines. Results show that path search with $k = 4$ among $N = 1000$ intersections requires 5.58 seconds with a 3.2% distance deviation rate and an 82.5% path similarity.**

***Index Terms*—Navigation services, unordered stops, accuracy, security, efficiency.**

Received 18 November 2023; revised 19 November 2024; accepted 19 December 2024. Date of publication 23 December 2024; date of current version 15 May 2025. This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant 62372149 and Grant U23A20303, and in part by EU LOCARD Project under Grant H2020-SU-SEC-2018-832735. *(Corresponding authors: Jianbo Gao; Zijian Zhang.)*

Meng Li and Yifei Chen are with the Key Laboratory of Knowledge Engineering with Big Data, Ministry of Education, Hefei University of Technology, Hefei 230002, China, also with the School of Computer Science and Information Engineering, Hefei University of Technology, Hefei 230002, China, and also with the Anhui Province Key Laboratory of Industry Safety and Emergency Technology; and Intelligent Interconnected Systems Laboratory of Anhui Province, Hefei University of Technology, Hefei 230002, China (e-mail: mengli@hfut.edu.cn; yifeichen@mail.hfut.edu.cn).

Jianbo Gao and Liehuang Zhu are with the School of Cyberspace Science and Technology, Beijing Institute of Technology, Beijing 100811, China (e-mail: jianbogao@bit.edu.cn; liehuangz@bit.edu.cn).

Jingyu Wu is with the School of Computer Science and Technology, University of Science and Technology of China, Hefei 230026, China (e-mail: andyng@mail.ustc.edu.cn).

Zijian Zhang is with the School of Cyberspace Science and Technology, Beijing Institute of Technology, Beijing 100811, China, and also with the Southeast Institute of Information Technology, Beijing Institute of Technology, Putian, Fujian 351100, China (e-mail: zhangzijian@bit.edu.cn).

Jialing He is with the College of Computer Science, Chongqing University, Chongqing 400044, China (e-mail: hejialing@cqu.edu.cn).

Mauro Conti is with the Department of Mathematics and HIT Center, University of Padua, 35131 Padua, Italy, and also with the Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, 2628 CD Delft, The Netherlands (e-mail: mauro.conti@math.unipd.it).

Xiaodong Lin is with the School of Computer Science, University of Guelph, Guelph, ON N1G 2W1, Canada (e-mail: xlin08@uoguelph.ca).

This article has supplementary downloadable material available at https://doi.org/10.1109/TDSC.2024.3521396, provided by the authors.

Digital Object Identifier 10.1109/TDSC.2024.3521396

I. Introduction

NAVIGATION services [1], [2] in vehicular networks allow users to upload a starting point S and a destination D to a Navigation Service Provider (NSP) that returns a path between S and D to users. Such a path is generally the shortest path or optimal, e.g., a path that has the minimum travel time but not the shortest distance. Being one of the convenient applications for modern transportation, navigation services offer significant social benefits such as reducing travelling time, saving gasoline, and relieving traffic congestion. The Google Maps [3] was the most downloaded app in the US [4], and a Morgan Stanley analyst estimated that it would be worth $11 billion in 2023 [5]. Although convenient, navigation services pose security and privacy threats [6] to users as untrusted NSPs collect sensitive location data that expose private activities [7], [8] and profile users [9], [10]. Therefore, immense research efforts are spawned on graph encryption [11], [12], [13], [14], [15] and privacy-preserving navigation [1], [2].

We observe a promising function in Google Maps [16]: a user can select a few stops, i.e., midway locations, between S and D. For example, as shown in Fig. 1, we add two ordered stops between S and D on the webpage and the NSP returns a path that passes stop 1 and stop 2in the same order as we have specified. Another example is that a rider can add extra stops when hailing a ride on Uber [17]. The new function provides several benefits for users. It saves planning time and creates a visual itinerary making the trip lively [18].

Inspired by this user-friendly function, *our motivation is to navigate from S to (not necessarily different) D while passing through k unordered stops*, i.e., users submit k stops besides S and D while not requiring the visiting order of the k stops. We name it *Semi-Constrained Navigation Problem (SCNP)*. For example, Michelle who just moved to New York recently opens the Google Maps app to query the shortest path from home to work while passing the Empire State Building, Rockefeller Center, and Bryant Park. Meanwhile, she does not care about how the NSP orders the three locations if the returned path is

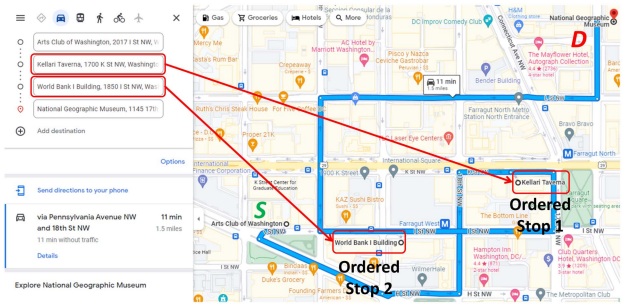


Fig. 1. Google navigation with two Ordered Stops.

the shortest. This new feature also applies to ride-hailing and intercity travel.

SCN is appealing for offering more flexibility to users. We also justify our motivation on Mechanic Turk [19] (See Section VIII-A). SCNP is different from the Constrained Shortest Path (CSP) problem, which looks for the shortest path under additional constraints. SCNP is indeed strongly associated with the Travelling Salesman Problem (TSP) and Path TSP. *The difference is that TSP has the same S and D, Path TSP requires S and D to be distinct [20], whereas our SCNP allows both assumptions.* Research on CSP [21], [22], [23], [24], [25], graph encryption [11], [12], [13], [14], [15], and privacy-preserving navigation [1], [2] do not support unordered stops. Therefore, accurate, secure, and efficient SCN exhibits a different and complex structure that cannot be efficiently handled by using current schemes. This necessitates new navigation strategies, which induces *three problems*. (1) How to *accurately* request the (approximately) shortest path from S to D while passing through k unordered stops? Unlike classic TSP where $D = S$, S and D in SCNP can be different. Although finding the shortest path is significant, we choose to acquire an approximate path that balances accuracy, security, and efficiency. (2) How to *securely* solve SCNP? As security matters for navigation users, we strive to protect users' location information (S, D, and k stops) from the untrusted NSP. Intuitively, all the locations and distances should be encrypted somehow. (3) How to *efficiently* navigate and compute over an encrypted city map? The nature of navigation services requires the NSP to provide efficient query services to users who value their time. (See discussion in Section VIII-B). However, the map is a complicated graph and we recommend a path over ciphertexts, thereby raising the bar for maintaining acceptable efficiency.

A straightforward approach to solving problem 1 is to first compute $k!$ permutations of the k unordered stops and compute the shortest path following the order in $k!$ permutations. For example, we can use Dijkstra's algorithm [26] to compute $4! = 24$ shortest paths given four stops P_1, P_2, P_3, and P_4. To guarantee security for this approach, we can adopt graph encryption [11], [12], [13], [14], [15] to secure all locations. However, it will incur a huge computational and communication cost.

In this work, *we propose a divide-and-conquer approach to group the k unordered stops into different subareas and compute a subpath in each subarea sequentially.* In this regard, we need to cope with *four technical challenges*. (1) How to group stops into subareas and orient them toward vector **SD** when locations are encrypted? After graph encryption, it is difficult for NSP to group and orient stops over ciphertexts. (2) How to bridge the navigation between two sequential subareas? The connections between two sequential subareas are essential to computing two corresponding subpaths, e.g., looking for a contemporary D for current subarea. (3) How to compute a subpath within each subarea over the encrypted graph? Different subareas have different navigation features, calling for different navigation strategies. For instance, navigation from S to the stops below S somehow circles back to S, but the navigation between S to D appears to be a zigzag. (4) How to reduce the query time to provide efficient navigation services while not sacrificing accuracy and security? Users are eager to acquire a navigation path, thereby putting a strict requirement on the NSP to complete path searching.

We address the four challenges by designing a secure navigation algorithm consisting of five strategies. A straightforward way is to hop to the nearest stop. However, it may not produce the shortest path. To guarantee navigation accuracy and reduce the search time, we strike a balance between accuracy and efficiency. We propose to organize the whole area into a navigation tree, recursively depth-first-search the tree, and securely invoke the five strategies stop-by-stop to compute three subpaths in three subareas.

1) *Group:* The k stops into three subareas SA^{Low}, SA^{Mid}, SA^{High} according to S, D, vector **SD**, and two orthogonal lines. We determine which subareas each P is located in by securely computing the Numerator of Cosine Similarity (NCS) between **SD** and **SP** and NCS between **DS** and **DP**, and securely comparing with 0, solving the first challenge. This is facilitated by performing ciphertext computation via additively homomorphic encryption [27] and secure comparison via garbled circuits [28], [29].

2) *Bridge:* The three subareas by computing two bridge points B_1 and B_2, solving the second challenge. B_1 is the vertically nearest (to S) stop in SA^{Mid} as a contemporary D' computed by ciphertext computation and secure comparison. B_2 is just set as the final D since SA^{Mid}'s destination cannot be computed in advance. After grouping and bridging, we invoke three navigation strategies to compute a sub-path in each subarea, solving the third challenge.

3) *Orient-Split-sCan-Hop (OSCH)* for SA^{Low}*: orient* B_1 and stops $\{P_i\}$ in SA^{Low} toward vector **SD** by computing the NCS of **SD**'s orthogonal vector and $\mathbf{SB_1}$ ($\mathbf{SP_i}$) over ciphertexts, decide which semispace SS of SA^{Low} to visit first by observing the orientation of P and $\{P_i\}$; *split* the first SS by drawing vector **SD'**; *scan* the P with the maximum/minimum angle $\angle PB_1S$; *hop* to P through greedy heuristic, and restart grouping. (4) *Vertically Nearest Hop (VNH)* for SA^{Mid}: hop to the vertically nearest (to S) stop and group. (5) *Orient-Split-Pull-Hop (OSPH)* for SA^{High}: *orient* current S and stops in SA^{High} to sequentially visit the two semispaces of SA^{High}, *split* the first semispace by drawing vector **S'D** where S' is the current S; *pull* the P with the max/min angle $\angle PB_2S'$; *hop* to the P and group. Finally, we obtain a complete path.

We design a data structure called the path-distance oracle to lay the foundation for efficient constrained navigation and avoid executing excess security primitives during a path query, thereby striking a good balance between accuracy and efficiency, solving the fourth challenge. Our contributions are:

- We propose the concept of SCN and propose Hermes to facilitate accurate, secure, and efficient SCN. We design a depth-first-search to recursively invoke five strategies stop-by-stop to compute subpaths.
- We propose a secure navigation algorithm without sacrificing accuracy or efficiency. It invokes seven secure algorithms constructed from well-integrated cryptographic primitives, garbled circuits, and data structures. We formally prove the security in the random oracle model.
- We implement a prototype of Hermes. We analyze its time complexity and thoroughly experiment over the map of Los Angeles to evaluate the accuracy and efficiency. We also compare with six baselines to show Hermes' advantages.

Organizations: We review related work in Section II. We introduce our system architecture, threat model, and performance objectives in Section III. We review five preliminaries in Section IV. We present the proposed scheme Hermes in Section V. In Section VI, we formally analyze the security. In Section VII, we show the implementation and performance analysis. Finally, we give two discussions in Section VIII, provide some justifications in Section IX, and conclude our work in Section X.

II. Related Work

A. K-Stops Shortest Path Computation and Path TSP

Terrovitis et al. [30] tackled the a-autonomy and k-stops shortest path problem in large spatial databases. a-autonomy means the distance between any two stops in the path is not greater than a and k-stops stipulate that the number of intermediate stops is k. Their solution first applies heuristics to efficiently retrieve a path that satisfies the constraint a or k. Then it prunes the search space and eliminates most data points by using the path length. Lastly, it computes the actual shortest path, w.r.t. a and k, by using only non-eliminated points. The search complexity is $O(k^2N + kN_1^2)$ where N_1 is the number of nodes after pruning. It is worth noting that their objective is to find k stops from N nodes and then compute a shortest path, which is different from SCNP.

Zenklusen [20] proposed a 1.5-approximation for the Path TSP. Given a complete undirected graph $G = (V, E)$, the objective is to compute a shortest path between two different vertices and the path visits every vertex exactly once. He shows that it suffices to obtain a Held-Karp solution with a well-defined set of properties to obtain a 1.5-approximation through dynamic programming. The search complexity is $O(2^N N^2 + N^4 \log_2 N + |E| \log_2 N + |E|N^2) > O(2^N)$. The difference between Path TSP and our SCN is that S and D are different in Path TSP.

Traub et al. [31] proposed a black-box reduction from the Path TSP to the classical TSP, i.e., Path TSP can be approximated equally well as TSP, by transforming approximation algorithms for TSP into ones for Path TSP.

B. Constrained Shortest Path Query

In a constrained shortest path (CSP) query, the objective is to find the shortest path from S to D whose total cost does not exceed C. Wang et al. [21] proposed COLA for approximate CSP over large road networks. They index the vertices locating on partition boundaries, and apply an on-the-fly algorithm to compute a path within a partition, which effectively prunes paths based on landmarks. The point CSP query and interval CSP query over a large time-dependent graph were studied in [22]. They are dynamic and evolve over time. Approximate sequential algorithms are proposed to answer the two queries. Lu et al. [24] proposed a framework Vine to parallelize the labeling algorithm to efficiently find the exact CSP solution using GPUs. They develop a two-level pruning approach to solve the subproblems by utilizing the GPU's hierarchical memory and propose an adaptive parallelism control model to adjust the degree of parallelism. Gong et al. [23] proposed a skyline query whose evaluation is constrained by a multi-cost transportation network and whose answers are off the network. They propose two approaches based on best first search to find exact answers and utilize heuristic rules to find approximate solutions. Liu et al. [25] propose a skyline path concatenation approach to efficiently construct a 2-hop labeling index for the CSP queries, which avoids the expensive skyline path search. They design a rectangle-based technique to prune the concatenation space from multiple hops and use a constraint pruning method to expedite the CSP query processing.

C. Graph Encryption

Meng et al. [11] proposed graph encryption for approximate shortest distance queries with three oracle encryption schemes. Only the third one is both computationally-efficient and achieves optimal communication complexity. Their key idea is to build sketched-based distance oracles and find the set of nodes $\mathcal{N}$ in common between sketches Sk_S and Sk_D and return the encrypted $\mathsf{Dist}(S, v) + \mathsf{Dist}(v, D)$ for all $v \in \mathcal{N}$. Wu et al. [12] proposed a cryptographic protocol for city navigation that provides privacy for user's location and the NSP's routing data. They compress the routing information in road graphs such that each entry in the next-hop routing matrix is specified by two bits. The user acquires the next hop on the shortest path to D after each iteration of the protocol based on affine encodings and garbled circuits. The number of nodes in the shortest path determines the rounds of interactions. Wang et al. [13] proposed a secure graph encryption scheme SecGDB to encrypt adjacency lists and enable private graph queries. They leverage additive homomorphic encryption, garbled circuits, and Fibonacci [32] heap to implement Dijkstra's algorithm with optimal time and storage complexities. Based on this work, Du et al. [14] proposed a structured encryption scheme GraphShield. Besides shortest distance queries, it supports some graph-based queries (e.g., maximum flow and minimum spanning tree) and more complicated analysis (e.g., PageRank). This is done by designing some tailored primitives including representation of decimals, secure multiplication or division protocols, and activation functions. Unlike the previous works, Ghosh et al. [15] proposed

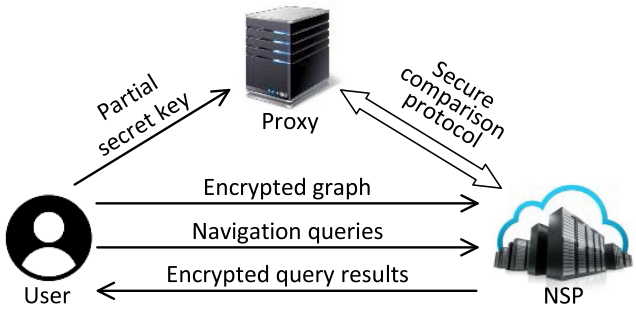


Fig. 2. System model of hermes.

the first graph encryption scheme for Single-Pair Shortest Path (SPSP) queries. They encrypt SP-matrix of a graph and process the recursion, for an unlimited number of recursive steps and with an encrypted structure growing in graph size.

Hermes advances the state of the art from three aspects: facilitate an accurate SCN service with k stops that do not have a fixed visiting order, provide a secure SCN service by protecting users' locations from the untrusted NSP, and guarantee efficient SCN regarding the NSP's search time.

III. Problem Statement

A. System Model

As portrayed in Fig. 2, the system model of Hermes consists of three entities, namely user U, an NSP, and a proxy. In the Setup stage, the user U processes a road graph $\mathcal{RG}$ to obtain path-distance oracles of intersections and an encrypted road graph $\mathcal{ERG}$. U outsources the $\mathcal{ERG}$ to the NSP and distributes a partial secret key sk from the secret key set $\mathcal{SK}$ to the proxy. We assume that the NSP and the proxy do not collude which has been widely adopted in the literature [13], [33], [34], [35]. Next, holding a constrained navigation query q with k unordered stops $\{P_1, P_2, \ldots, P_k\}$, U generates a query token t_q and sends it to the NSP. The NSP searches $\mathcal{ERG}$ by using t_q, computes an encrypted approximately shortest path $Easp_q$ with the aid of the proxy, and returns $Easp_q$ along with the encrypted distance $Edist_q$ to U. We formally define the SCN and secure SCN as below.

Definition 1. SCN: Given a starting point S, a destination D, a stop set $\mathcal{P} = \{P_1, \ldots, P_k\}$, and a navigation function F, SCN is to find a sequence S of k different stops in $\mathcal{P}$ constituting a shortest path from S to D passing $\mathcal{P}$:

$$\arg\min_{P_{i_1},\ldots,P_{i_k}} F(S, D, \{P\}) = \{(P_{i_1}, \ldots, P_{i_k}) | \forall \mathsf{S}(P_1, \ldots, P_k) : \mathsf{Dist}(S, P_{i_1}, \ldots, P_{i_k}, D) \leq \mathsf{Dist}(S, \mathsf{S}(P_1, \ldots, P_k), D)\}.$$

Definition 2. Secure SCN: A secure SCN system supporting approximately shortest path query with k unordered stops consists of the following four Probabilistic Polynomial-Time (PPT) algorithms:

- $(\mathcal{SK}, \mathcal{PK}, \mathcal{RG}) \leftarrow \mathsf{Gen}(1^\lambda, \mathcal{CM})$: is a probabilistic generation algorithm executed by user. It takes security parameter λ and city map $\mathcal{CM}$ as input, and outputs a secret key set $\mathcal{SK}$, a public key set $\mathcal{PK}$, and a road graph $\mathcal{RG}$.
- $\mathcal{ERG} \leftarrow \mathsf{Enc}(\mathcal{SK}, pk, \mathcal{RG})$: is a probabilistic algorithm executed by the user. It takes as input a secret key set $\mathcal{SK}$, a public key pk (part of $\mathcal{PK}$), and a road graph $\mathcal{RG}$, and outputs an encrypted road graph $\mathcal{ERG}$.
- $\mathcal{EQR}_q \leftarrow \mathsf{PathQuery}(\mathcal{SK}, q; \mathcal{ERG}, \mathcal{PK})$: is an interactive protocol executed between the user and the NSP. The user takes a secret key set $\mathcal{SK}$ and a constrained navigation query $q = (S, D, P_1, P_2, \ldots, P_k)$ as input. The NSP takes a query token t_q, an encrypted road graph $\mathcal{ERG}$, and a public key set $\mathcal{PK}$ as input. During the protocol, the user computes a query token t_q based on q and sends it to the NSP. After the protocol, the user receives an encrypted result $\mathcal{EQR}_q = (Easp_q, Edist_q)$.
- $\mathcal{QR}_q \leftarrow \mathsf{PathRecover}(\mathcal{SK}, \mathcal{EQR}_q)$: is a deterministic algorithm executed by the user. It takes a secret key set $\mathcal{SK}$ and an encrypted query result $\mathcal{EQR}$, and outputs the query result $\mathcal{QR} = (asp_q, dist_q)$.

B. Security Model

Now we present the security definition for our constrained navigation scheme. At a high level, the security protection we expect from our constrained navigation scheme is that: (1) given an encrypted road graph $\mathcal{ERG}$, no adversary can acquire any useful information about the road graph $\mathcal{RG}$ and (2) given the view of a polynomial number of PathQuery executions on an adaptively chosen constrained navigation queries $\mathcal{Q} = \{q_1, q_2, \ldots, q_o\}$, no adversary can acquire any useful information about q or $\mathcal{RG}$.

Such a security notion is not easy to achieve efficiently, thereby many efficient secure searchable encryption or structured encryption schemes [11], [36], [37], [38], [39] allow for some leakage to trade for better performance. We adopt this relaxation as well. According to [36], [40], [41], this is achieved by parameterizing the security definition with leakage functions for system functions which in this work include Enc algorithm and PathQuery protocol. The leakage functions precisely capture what information is leaked by the encrypted road graph and query tokens. Besides the leakage of interactions between the user and the NSP [11], [36], [38], we should consider the one between NSP and proxy. We adapt the notion of adaptive semantic security [13], [15], [36], [40], [41] and give its definition as follows.

Definition 1 (Adaptive Semantic Security): Let $\Pi = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{PathQuery}, \mathsf{PathRecover})$ be a constrained navigation scheme. Consider the following two probabilistic experiments with a semi-honest adversary $\mathcal{A}$, a challenger $\mathcal{C}$, a simulator $\mathcal{S}$, and two (stateful) leakage functions $\mathcal{L}_1$, $\mathcal{L}_2$:

Real $_{\mathcal{A}}(1^\lambda)$:

- $\mathcal{A}$ outputs a road graph $\mathcal{RG}$.
- $\mathcal{C}$ calculates $(\mathcal{SK}, \mathcal{ERG}) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{RG})$ and sends $\mathcal{ERG}$ to $\mathcal{A}$.
- $\mathcal{A}$ generates a polynomial number of adaptively chosen constrained navigation queries $\mathcal{Q} = (q_1, q_2, \ldots, q_o)$. For query q_i $(1 \leq i \leq o)$, $\mathcal{A}$ and $\mathcal{C}$ execute $\mathsf{PathQuery}(\mathcal{SK}, q_i; \mathcal{ERG}, \mathcal{PK})$. $\mathcal{C}$ acts as a user and $\mathcal{A}$ acts as the NSP.

- $\mathcal{A}$ computes a bit b that is output by the experiment.

IDeal $_{\mathcal{A},\mathcal{S}}(1^\lambda)$:

- $\mathcal{A}$ outputs a road graph $\mathcal{RG}$.
- Given $\mathcal{L}_1$, $\mathcal{S}$ sends encrypted road graph $\mathcal{ERG}$ to $\mathcal{A}$.
- $\mathcal{A}$ generates a polynomial number of adaptively chosen constrained navigation queries $\mathcal{Q} = (q_1, q_2, \ldots, q_o)$. For each q_i $(1 \leq i \leq o)$, $\mathcal{S}$ is given $\mathcal{L}_2$. $\mathcal{A}$ and $\mathcal{S}$ execute a simulation of $\mathsf{PathQuery}$. $\mathcal{S}$ acts as a user and $\mathcal{A}$ acts as the NSP.
- $\mathcal{A}$ computes a bit b that is output by the experiment.

We say that Π is adaptively $(\mathcal{L}_1, \mathcal{L}_2)$- semantically secure if for all PPT adversaries $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ such that $|\Pr[\mathbf{Real}_{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(1^\lambda) = 1]| \leq \mathrm{negl}(\lambda)$.

We adopt a semi-honest model for the cloud server, but it is interesting to consider a malicious model in the future work. When the cloud server is malicious, it can be assumed to alter and neglect both the indexes and the tokens. Such attacks require checking integrity and completeness by the user. Meanwhile, efficiency cannot be sacrificed in defending against the malicious cloud server.

C. Performance Objectives

There are three performance objectives in this work. *Function*. SCN returns to users an approximate shortest path from S to D while passing through k unordered stops. *Efficiency*. The navigation system achieves good efficiency regarding computational costs, communication overhead, and scalability. *Accuracy*. The navigation system achieves good accuracy, i.e., low distance deviation rate and high path similarity.

IV. Preliminaries

A. Bilinear Pairing

Let $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ be cyclic groups of prime order p. A bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is referred to as a Type 3 (asymmetric) pairing if it satisfies the following conditions: (1) Efficient Computability: $e(g_1, g_2)$ can be efficiently computed for any $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$. (2) Bilinearity: For any $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$, and $a_1, a_2 \in \mathbb{Z}_p^*$, the pairing satisfies $e(g_1^{a_1}, g_2^{a_2}) = e(g_1, g_2)^{a_1 \cdot a_2}$. (3) Non-degeneracy: For any $g_1 \in \mathbb{G}_1 \backslash \{1_{\mathbb{G}_1}\}$ and $g_2 \in \mathbb{G}_2 \backslash \{1_{\mathbb{G}_2}\}$, $e(g_1, g_2) \neq 1_{\mathbb{G}_T}$.

B. Hash Function

A hash function is a deterministic algorithm that takes an input of arbitrary length and produces a fixed-size output. A cryptographic hash function $H : \{0,1\}^* \to \{0,1\}^n$ maps an input of arbitrary length to an output of fixed size n. One of its main properties is collision resistance: it is computationally infeasible to find any two distinct inputs x_1 and x_2 such that $H(x_1) = H(x_2)$.

C. Homomorphic Encryption

A public-key encryption scheme $\Omega = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is homomorphic if it has an evaluation algorithm Eval that takes a function E and a series of ciphertexts $(c_1, c_2, \ldots, c_n)$ where $c_i = \mathsf{Enc}_{pk}(m_i)$ as input, and outputs a ciphertext c such that $\mathsf{Dec}_{sk}(c) = E(m_1, m_2, \ldots, m_n)$. If E supports addition: $\mathsf{Enc}_{pk}(m_1 + m_2) = \mathsf{Eval}(+, \mathsf{Enc}_{pk}(m_1), \mathsf{Enc}_{pk}(m_2))$, Ω is called additive homomorphic encryption. Concrete instantiations of homomorphic encryption schemes include Paillier cryptosystem [42] and BGN [27]. We denote the ciphertext of a message m under the public key pk as $[\![m]\!]$.

D. Pseudo-Random Function (PRF) and Pseudo-Random Permutation (PRP)

Let $F : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ be a PRF. The output of a PRF cannot be distinguished from the output of a random function by any PPT distinguisher, i.e., $|\Pr[D^{F_k(\cdot)}(1^n) = 1] - \Pr[D^{f(\cdot)}(1^n) = 1]| \leq \mathrm{negl}(\lambda)$, where a key k is chosen uniformly from $\{0,1\}^n$. In other words, D is given access to an oracle $\mathcal{O}$ which is either F or f. D can query $\mathcal{O}$ at any point i to receive $\mathcal{O}(i)$. For every D that receives a description of F outputs 1 with almost the same probability as when it receives a description of a random function [43]. A PRF is assumed to be a PRP when it is bijective.

E. Cosine Similarity

Cosine similarity is a measure of similarity between two vectors $\mathbf{v}_1, \mathbf{v}_2$ in an inner product space. It is the cosine of the angle between $\mathbf{v}_1$ and $\mathbf{v}_2$, i.e., the dot product of the $\mathbf{v}_1$ and $\mathbf{v}_2$ divided by the product of their lengths. The cosine similarity depends only on their angle θ and belongs to the interval $[-1, 1]$. Given two vectors of attributes, $\mathbf{A}$ and $\mathbf{B}$, the cosine similarity is computed as:

$$\mathsf{Sim}(\mathbf{A}, \mathbf{B}) = \mathsf{Cos}(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{||\mathbf{A}||||\mathbf{B}||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}.$$

F. Garbled Circuits (GC)

Yao introduced GC [28], [29] for secure two-party computation in the presence of semi-honest adversaries [44], [45], [46], [47]. It allows two parties with secret values a and b respectively to an arbitrary function $F(a, b)$. The first party C called circuit generator creates a circuit $\widetilde{C}$ with wires. For each wire w_i, C chooses two random garbled values $\widetilde{w}_i^0, \widetilde{w}_i^1$, where $\widetilde{w}_i^j$ is the garbled value of w_i's value j. For each gate g_i, C creates a garbled table $\widetilde{T}_i$ that allows to obtain only the garbled value of the corresponding g_i's output given a set of garbled values of g_i's inputs. C sends the garbled tables, i.e., $\widetilde{C}$, to the second party called circuit evaluator E. E obtains the garbled inputs $\widetilde{w}_i$ corresponding to inputs of both parties via oblivious transfer. E evaluates $\widetilde{C}$ on the garbled inputs to obtain the garbled outputs simply by evaluating the garbled circuit gate by gate, using $\widetilde{T}$ [44]. GC's formal security proof in the semi-honest model is given in [48]. GC also finds its adoptions in some privacy-preserving vehicular applications [49], [50].

G. Oblivious Transfer (OT)

OT_l^t, parallel 1-out-of-2 OT of t l-bit strings [51], [52], is a two-party protocol executed between a chooser A and a sender

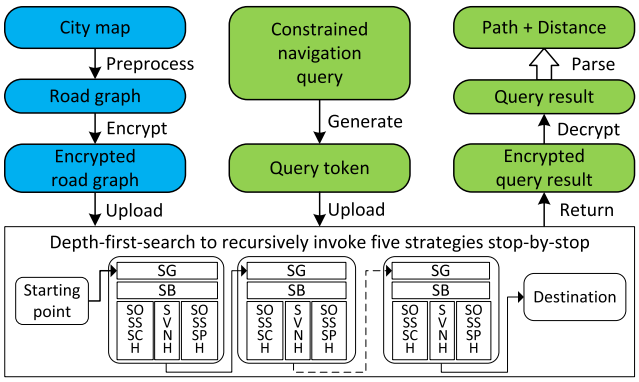


Fig. 3. Overview of Hermes.

B. For $i = 1, 2, \ldots, t$, B inputs a pair of l-bit strings s_i^0, s_i^1 and A inputs t chosen bits b_i. Upon the completion of the protocol, A acquires the chosen strings $s_i^{b_i}$ but not the unchosen strings $s_i^{1-b_i}$, and B learns nothing about b_i.

V. The Proposed Scheme Hermes

A. Overview

At a high level, Hermes consists of four phases. In setup, the user preprocesses an original city map $\mathcal{CM}$ to obtain a road graph $\mathcal{RG}$ and generates cryptographic keys. A partial secret is sent to the proxy. In graph encryption, the user computes a path-distance oracle $\mathcal{PD}_v$ for each intersection v in $\mathcal{V}$ and encrypts $\mathcal{RG}$ to obtain the encrypted road graph $\mathcal{ERG}$. The $\mathcal{ERG}$ is uploaded to the NSP. In path search, the users holds a constrained navigation query q and generates a query token t_q based on q. The user sends t_q to the NSP, which searches $\mathcal{ERG}$ by using t_q and returns an encrypted query result $\mathcal{EQR}_q$ to the user. In path recovery, the user decrypts $\mathcal{EQR}_q$ to recover the query result $\mathcal{QR}_q$ and obtain the approximately shortest path asp_q and its distance $dist_q$. We provide an overview of Hermes in Fig. 3 and a table of key notations in Table I.

B. Setup

The setup consists of Gen that preprocesses a city map $\mathcal{CM}$ and generates keys $\mathcal{SK}, \mathcal{PK}$. Hermes makes use of a public-key encryption scheme $\Omega = \{\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}\}$ [27], a PRF $F : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^*$, a PRP $T : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^\lambda$, a collision-resistant hash function H_1, and a random oracle $H_2 : \{0,1\}^* \to \{0,1\}^*$. Given a security parameter 1^λ and a city map $\mathcal{CM}$, the user proceeds as follows:

- Transform $\mathcal{CM}$ into a road graph $\mathcal{RG} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of vertices, $\mathcal{E}$ is the set of undirected edges, and $w_{v_i v_j}$ is the length for each edge $(v_i, v_j) \in \mathcal{E}$.
- Sample four secret keys k_1, k_2, k_3, k_4 for T_1, T_2, F_3, F_4. We use $T_1(\cdot), T_2(\cdot), F_3(\cdot), F_4(\cdot)$ to denote $T_{k_1}(\cdot)$, $T_{k_2}(\cdot), F_{k_3}(\cdot), F_{k_4}(\cdot)$, respectively.
- Run $\Omega.\mathsf{Gen}(1^\lambda)$ to obtain a tuple $(q_1, q_2, n, g, u, h, \mathcal{G}, \mathcal{G}_1, e)$, where $\mathcal{G}$, $\mathcal{G}_1$ are groups of order $n = q_1 q_2$, g and u are two generators of $\mathcal{G}$, $h = u^{q_2}$, and $e : \mathcal{G} \times \mathcal{G} \to \mathcal{G}_1$ is a bilinear map. Set $pk = (n, \mathcal{G}, \mathcal{G}_1, e, g, h)$ and $sk = q_1$; compute $g_1 = e(g, g)$ of order n, $h_1 = e(g, h)$ of order q_1. We denote $\Omega.\mathsf{Enc}_{pk}(m) = g^m h^r \bmod n$ by $[\![m]\!]$ [27].
- Publish $\mathcal{PK} = (pk, g_1, h_1)$ and set $\mathcal{SK} = (k_1, k_2, k_3, k_4, sk)$; send sk to the proxy through a secure channel.

TABLE I
Key Notations of Hermes

Notation	Definition
S, D, B_1, B_2	Starting point, destination, bridge points
$\{P_1, P_2, \cdots, P_k\}$	k unordered stops
S'	Current starting point
D'	Contemporary destination
SD	Vector from S to D
OV	Orthogonal vector of **SD**
$-$**OV**	Opposite direction of **OV**
SA^{Low}, SA^{Mid}, SA^{High}; SS	Subareas; semispace
$q, t_q, \mathcal{QR}$	Query, token, query result
$asp_q, dist_q$	Shortest path, shortest distance
$\mathcal{SK}, \mathcal{PK}$	Secret key, public key
$F(\cdot)$	Pseudo-Random Function
$T(\cdot)$	Pseudo-Random Permutation
H_1, H_2	Hash function, random oracle
$\mathcal{RG} = (\mathcal{V}, \mathcal{E})$	Road graph with vertices and edges
$\{\mathcal{PD}_{v_i}\}_{i=1}^n$	Path-distance oracles
$(v^1_{v_i v_j}, v^2_{v_i v_j}, \cdots, v^L_{v_i v_j})$	Shortest path from v_i to v_j
sp_{v_i, v_j}	Shortest path from v_i to v_j
sd_{v_i, v_j}	Shortest distance from v_i to v_j
(x, y)	Pair of coordinates of node
$k_{v_i}, r_{v_i}, Arr_{v_i}$	Secret key, random number, array
DX, α, β	Dictionary and temporary parameters
add_{v_i}	Location of Arr_{v_i}
$\mathcal{ERG}$	Encrypted road graph
$[\![x_s]\!], [\![y_s]\!], [\![x_d]\!], [\![y_d]\!]$	Encrypted coordinates of S and D

C. Graph Encryption

The graph encryption consists of Enc that encrypts a road graph $\mathcal{RG}$. Given a road graph $\mathcal{RG}$, the user computes the shortest paths and distances for all nodes by using the Floyd-Warshall algorithm [53] to obtain path-distance oracles $\{\mathcal{PD}_{v_i}\}_{i=1}^n$. $\mathcal{PD}_{v_i} = \{\mathcal{PD}_{v_i}^{v_j}\}_{(v_j \in \mathcal{V})}$, which contains the shortest path and its distance from v_i to other nodes:

$$\mathcal{PD}_{v_i}^{v_j} = (v_i, v_j, v^1_{v_i v_j}, v^2_{v_i v_j}, \ldots, v^L_{v_i v_j}, sd_{v_i v_j}), \quad (1)$$

where $(v^1_{v_i v_j}, v^2_{v_i v_j}, \ldots, v^L_{v_i v_j})$ is the sequence of nodes that the shortest path from v_i to v_j traverses by. For example, as depicted in Fig. 4, there is a shortest path $sp_{v_1 v_2}$ from node v_1 to node v_2 and the shortest distance is $sd_{v_1 v_2}$. $\mathcal{PD}_{v_1}^{v_2} = (v_1, v_2, v_3, v_4, sd_{v_1 v_2})$.

Next, the user encrypts $\mathcal{RG}$ by encrypting the N path-distance oracles as follows.

- For each node v_i with a pair of coordinates (x, y), the user randomly chooses a secret k_{v_i} and a number r_{v_i}.
- For each $\mathcal{PD}_{v_i}^{v_j}$, the user creates an array Arr_{v_i} with the first location $Arr_{v_i}[0] = ((\mathsf{Enc}_{pk}(x) || \mathsf{Enc}_{pk}(y)) \oplus H_2(k_{v_1} || r_{v_1}), r_{v_1})$. The user computes the permutation of the N nodes $(T_1(v_1), \ldots, T_1(v_n))$ to store N path-distance oracles $\mathcal{PD}_{v_i}$. For each location $T_1(v_j)$, the user chooses a random number r_{v_j}

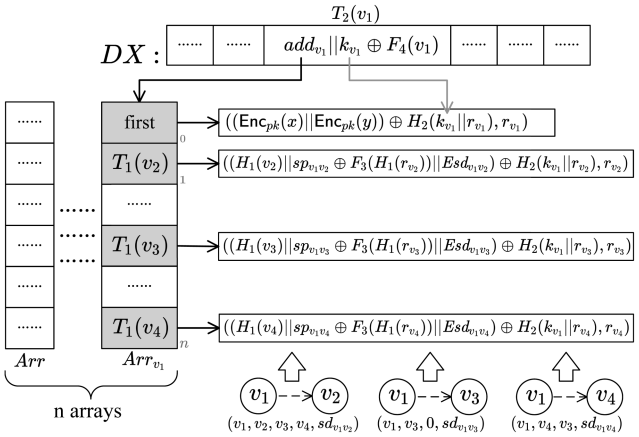


Fig. 4. An example of encrypting and storing $\mathcal{PD}_{v_1}$.

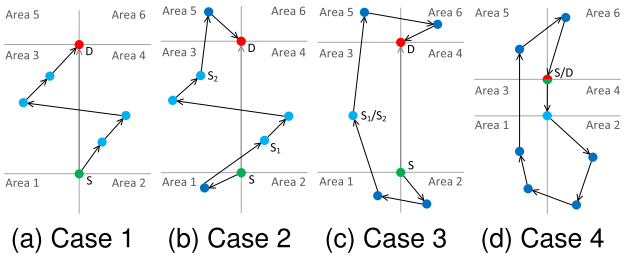


Fig. 5. Four cases of constrained navigation.

and stores $((H_1(v_j)||sp_{v_iv_j} \oplus F_3\ (H_1(r_{v_j}))||Esd_{v_iv_j}) \oplus H_2(k_{v_i}||r_{v_j}), r_{v_j})$ at $Arr_{v_i}[T_1(v_j)]$, where $Esd_{v_iv_j} = \Omega.\mathsf{Enc}_{pk}(sd_{v_iv_j})$.

- The user creates a dictionary DX to store the pairs $(T_2(v_i), add_{v_i}\ ||k_{v_i} \oplus F_4(v_i))$ for all $v_i \in \mathcal{V}$, where add_{v_i} is the location of Arr_{v_i}. Send $\mathcal{ERG} = (DX, Arr)$ to the NSP.

D. Path Search

The navigation query PathQuery, given a query token t_q, returns an encrypted query result $\mathcal{EQR}_q$. We first explain the rationale behind our divide-and-conquer approach. As drawn in Fig. 5, we divide the whole area into six subareas according to vector **SD** and two orthogonal lines (crossing S and D), leading to four constrained navigation cases.

In case 1, the stops only locate in Area 3&4. Intuitively, the navigation path can adopt a greedy-heuristic, e.g., always visiting the vertically nearest stops or always visiting the nearest stop. The two strategies are not perfect and we will introduce our strategy VNH in Section V-D5.

In case 2, there is one stop in Area 1&2 (5&6). It is not difficult to see that the navigation path visits the stop in Area 1&2 before going into Area 3&4; otherwise, it will incur unnecessary travel. Similarly, the navigation path should pass the stop in in Area 5&6 before finishing the journey at D.

In case 3, there are more than one stop in Area 1&2 and Area 5&6. The local navigation below S and the one above D resemble the overall navigation. Starting from S, the local navigation traverses the stops below S and ends up in the first-to-visit stop S_1 above S, which is a temporary D. Landing at a temporary destination S_2 between S and D, the local navigation traverses the stops above D and ends up in D. *Interestingly, we discover a navigation pattern here that can be extracted as recursively finding a local navigation path from a current S to a contemporary D, and updating (S, D) until a local path reaches the final D.* This pattern is expressed as $\mathsf{Path}(S, D) = \mathsf{Path}(S, S_1) + \mathsf{Path}(S_1, S_2) + \mathsf{Path}(S_2, D)$.

$\mathsf{Path}(S, D)$, which returns a navigation path from S to D, is divided into three subpaths $\mathsf{Path}(S, S_1)$, $\mathsf{Path}(S_1, S_2)$, and $\mathsf{Path}(S_2, D)$. The navigation in each of the three subpaths can be further divided into three subtasks by using the divide-and-conquer approach. Simply put, given a secure navigation algorithm SN, we have $\mathsf{SN}(Whole\ area) = \mathsf{SN}(Low) + \mathsf{SN}(Mid) + \mathsf{SN}(High)$.

In Case 4 where $D = S$, it is the classic TSP and also a normal navigation query. Although it seems difficult to navigate by using our divide-and-conquer approach, we transform it into one of the three cases above by *first hopping to the nearest stop and then restarting grouping*.

1) Query Token Generation: Given a constrained navigation query $q = (S, D, P_1, \cdots, P_k)$, the user computes a query token $t_q = (\mathcal{I}(S), \mathcal{I}(D), \mathcal{I}(P_1), \ldots, \mathcal{I}(P_k))$, where a triad $\mathcal{I}(D) = (T_2(D), F_4(D), H_1(D))$, and sends it to the NSP.

2) Strategy Secure Grouping: As shown in Fig. 6, we divide the whole area into three subareas: SA^{Low}, SA^{Mid}, and SA^{High}, corresponding to Area 1&2, Area 3&4, and Area 5&6in Fig. 5, respectively.

Given the location coordinates of S and D, we compute vector **SD** and draw two lines L_1, L_2 orthogonal to **SD** that cross S and D, respectively. Based on $\mathsf{Sim}(\cdot)$, we determine which subarea a stop locates in, $SA(P) =$

$$\begin{cases} SA^{\text{Low}}, & \mathsf{Sim}(\mathbf{SD}, \mathbf{SP}) \in [-1, 0] \wedge \mathsf{Sim}(\mathbf{DS}, \mathbf{DP}) \in (0, 1] \\ SA^{\text{Mid}}, & \mathsf{Sim}(\mathbf{SD}, \mathbf{SP}) \in (0, 1] \wedge \mathsf{Sim}(\mathbf{DS}, \mathbf{DP}) \in (0, 1] \\ SA^{\text{High}}, & \mathsf{Sim}(\mathbf{SD}, \mathbf{SP}) \in (0, 1] \wedge \mathsf{Sim}(\mathbf{DS}, \mathbf{DP}) \in [-1, 0] \end{cases}.$$

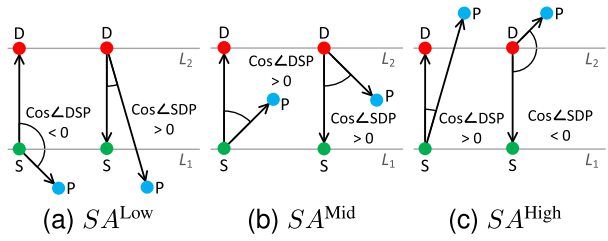


Fig. 6. Three cases of grouping A stop P.

To simplify the comparison, we only need the sign of the NCS, i.e., < 0, $= 0$, and > 0. From here, we design a concrete algorithm based on the homomorphic encryption [27]. Given the $\mathcal{ERG}$ and t_q, the NSP proceeds as follows.

- *Retrieve $[\![x_s]\!]$ and $[\![y_s]\!]$:* retrieve $\alpha = DX[T_2(S)]$ to obtain $add||k \oplus F_4(S)$; compute $add||k = \alpha \oplus F_4(S)$; recover the array pointed to by add and parse $Arr_S[0]$ as (β, r); retrieve $[\![x_s]\!]$ and $[\![y_s]\!]$ by computing $\beta \oplus H_2(k||r)$.

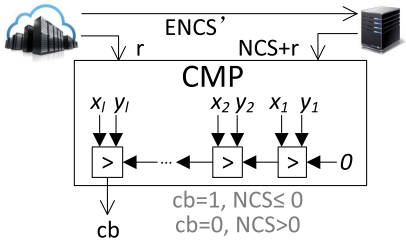


Fig. 7. Comparison circuit.

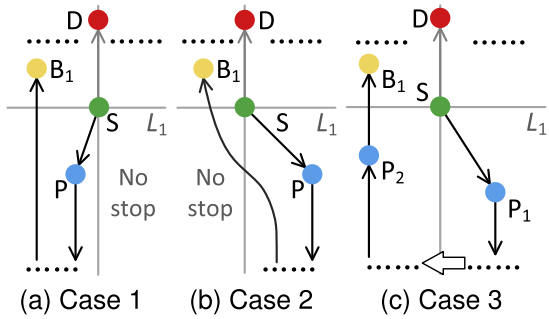


Fig. 8. Three local navigation cases of entering SA^{Low}.

- Retrieve $[\![x_d]\!]$ and $[\![y_d]\!]$ by using $T_2(D)$ and $F_4(D)$.
- Compute the encrypted **SD** and the encrypted **DS**:

$$\mathbf{ESD} = (\mathbf{ESD}^1, \mathbf{ESD}^2) = ([\![x_d]\!]/[\![x_s]\!], [\![y_d]\!]/[\![y_s]\!]) \quad (2)$$

$$\mathbf{EDS} = (\mathbf{EDS}^1, \mathbf{EDS}^2) = ([\![x_s]\!]/[\![x_d]\!], [\![y_s]\!]/[\![y_d]\!]) \quad (3)$$

- Initialize three sets of stops $\mathcal{P}^{\text{Low}}$, $\mathcal{P}^{\text{Mid}}$, and $\mathcal{P}^{\text{High}}$, corresponding to SA^{Low}, SA^{Mid}, and SA^{High}, respectively.
- For each P_i, compute its group as follows.
  - Retrieve $[\![x_{p_i}]\!]$ and $[\![y_{p_i}]\!]$ by using $T_2(P_i)$ and $F_4(P_i)$ similarly; compute the encrypted $\mathbf{SP_i}$:

$$\begin{aligned}\mathbf{ESP_i} &= (\mathbf{ESP_i^1}, \mathbf{ESP_i^2}) = ([\![x_{p_i}{-}x_s]\!], [\![y_{p_i}{-}y_s]\!]) \\ &= ([\![x_{p_i}]\!]/[\![x_s]\!], [\![y_{p_i}]\!]/[\![y_s]\!]) = (g^{x_{p_i}-x_s}h^r, g^{y_{p_i}-y_s}h^{r'})\end{aligned} \quad (4)$$

  - Compute the encrypted numerator of $\mathsf{Sim}(\mathbf{SD}, \mathbf{SP_i})$:

$$\begin{aligned}\mathrm{ENCS}_{\mathbf{SP_i}}^{\mathbf{SD}} &= [\![(x_{p_i}{-}x_s)(x_d{-}x_s) + (y_{p_i}{-}y_s)(y_d{-}y_s)]\!]_1 \\ &= g_1^{(x_{p_i}-x_s)(x_d-x_s)+(y_{p_i}-y_s)(y_d-y_s)} h_1^{r_1} \\ &= g_1^{(x_{p_i}-x_s)(x_d-x_s)} h_1^{r_2} g_1^{(y_{p_i}-y_s)(y_d-y_s)} h_1^{r_3} \\ &= e(\mathbf{ESP_i^1}, \mathbf{ESD}^1) h_1^{\hat{r}_2} e(\mathbf{ESP_i^2}, \mathbf{ESD}^2) h_1^{\hat{r}_3}.\end{aligned} \quad (5)$$

  - Choose a random number r and compute a randomized version of $\mathrm{ENCS}_{\mathbf{SP_i}}^{\mathbf{SD}}$:

$$\begin{aligned}\mathrm{ENCS}'^{\mathbf{SD}}_{\mathbf{SP_i}} &= [\![(\mathrm{x_{p_i}}{-}\mathrm{x_s})(\mathrm{x_d}{-}\mathrm{x_s}){+}(\mathrm{y_{p_i}}{-}\mathrm{y_s})(\mathrm{y_d}{-}\mathrm{y_s})]\!]_1 [\![\mathrm{r}]\!]_1 \\ &= [\![(x_{p_i}{-}x_s)(x_d{-}x_s){+}(y_{p_i}{-}y_s)(y_d{-}y_s){+}r]\!]_1,\end{aligned} \quad (6)$$

    where $[\![r]\!]_1$ is the encryption of r in $\mathbb{G}_1$.
  - Perform comparison on $\mathrm{NCS}'^{\mathbf{SD}}_{\mathbf{SP_i}}$ with the proxy based on the garbled circuits to select the minimum of $\mathrm{NCS}'^{\mathbf{SD}}_{\mathbf{SP_i}}$ and r. The NSP's input is r and the proxy's input is $(x_{p_i} - x_s)(x_d - x_s) + (y_{p_i} - y_s)(y_d - y_s) + r$. If the output bit $cb_1 = 0$, then $\mathrm{NCS}_{\mathbf{SP_i}}^{\mathbf{SD}} > 0$; otherwise $\mathrm{NCS}_{\mathbf{SP_i}}^{\mathbf{SD}} \le 0$. $\mathsf{CT}(r, \mathrm{NCS}'^{\mathbf{SD}}_{\mathbf{SP_i}})$ is implemented by a secure comparison circuit [44], [46] as shown in Fig. 7. We note that the random number r has to be an enough big positive integer such that the decryption of $\mathrm{ENCS}'^{\mathbf{SD}}_{\mathbf{SP_i}}$ succeeds since the message space consists of non-negative integers [27].
  - Retrieve **EDS**, compute $\mathbf{EDP_i}$, $\mathrm{ENCS}_{\mathbf{DP_i}}^{\mathbf{DS}}$, perform comparison on $\mathrm{ENCS}_{\mathbf{DP_i}}^{\mathbf{DS}}$ to obtain an output cb_2.
  - Insert $\mathcal{I}(P_i)$ into one of the three sets of stops $\mathcal{P}^{\text{Low}}$, $\mathcal{P}^{\text{Mid}}$, and $\mathcal{P}^{\text{High}}$ according to the two comparison results. $\mathcal{P}(P_i) = SA^{\text{Low}}$ $(cb_1 = 1 \wedge cb_2 = 0)$, SA^{Mid} $(cb_1 = 0 \wedge cb_2 = 0)$, and SA^{High} $(cb_1 = 0 \wedge cb_2 = 1)$.

3) Strategy Secure Bridging: After secure grouping, the NSP computes two bridge points B_1 and B_2. We compute B_1 as the stop that is vertically nearest to S in SA^{Mid} and set the final D as B_2. Specifically, we compute B_1 as follows (see Fig. 11(a)) and will explain why we choose it in Section V-D5 Strategy VNH for SA^{Mid}.

- Use the euclidean distance and cosine similarity to compute the difference between projection of line SP_1 on SD and projection of line SP_2 on SD:

$$\begin{aligned}&\mathsf{Euc}(S, P_1)\mathsf{Sim}(\mathbf{SP_1}, \mathbf{SD}){-}\mathsf{Euc}(S, P_2)\mathsf{Sim}(\mathbf{SP_2}, \mathbf{SD}) \\ &= \frac{(x_{p_1}{-}x_{p_2})(x_d{-}x_s){+}(y_{p_1}{-}y_{p_2})(y_d{-}y_s)}{\sqrt{(x_d{-}x_s)^2{+}(y_d{-}y_s)^2}}\end{aligned} \quad (7)$$

- Compute the encrypted numerator of (8):

$$\begin{aligned}\mathrm{ENum}_{\mathbf{SD},S}^{\mathbf{P_1},\mathbf{P_2}} &= [\![(x_{p_1}{-}x_{p_2})(x_d{-}x_s){+}(y_{p_1}{-}y_{p_2})(y_d{-}y_s)]\!]_1 \\ &= e(\mathbf{EP_2P_1^1}, \mathbf{ESD}^1) h_1^{\hat{r}_2} e(\mathbf{EP_2P_1^2}, \mathbf{ESD}^2) h_1^{\hat{r}_3}.\end{aligned} \quad (8)$$

- Randomize $\mathrm{ENum}_{\mathbf{SD},S}^{\mathbf{P_1},\mathbf{P_2}}$ to $\mathrm{ENum}'^{\mathbf{P_1},\mathbf{P_2}}_{\mathbf{SD},\mathrm{S}}$ and compare $\mathrm{Num}'^{\mathbf{P_1},\mathbf{P_2}}_{\mathbf{SD},\mathrm{S}}$ with r by invoking $cb \leftarrow \mathsf{CT}(r, \mathrm{Num}'^{\mathbf{P_1},\mathbf{P_2}}_{\mathbf{SD},\mathrm{S}})$.
- If $cb = 0$, set P_1 as P_2 and continue.

Note that if $\mathcal{P}^{\text{Mid}} = \emptyset$, there are four cases to consider depending on whether $SS_L^{\text{High}} \stackrel{?}{=} \emptyset$ and $SS_R^{\text{High}} \stackrel{?}{=} \emptyset$ (please refer to Algorithms 3 and 8 in Appendix, available online).

4) Strategy OSCH for SA^{Low}: After bridging, the NSP computes a subpath in SA^{Low}. We first explain the rationale behind OSCH, i.e., a locally greedy navigation strategy for SA^{Low}, by giving our observations of three local navigation cases of entering SA^{Low} with at least two stops in Fig. 8. (For the one stop case, we just hop to it; for the no stop case, we hop from S to B_1.)

In case 1 and case 2, only one semispace of SA^{Low} has stops. A local greedy strategy passes P, $\mathcal{P}^{\text{Low}} \setminus \{P\}$, and then circles to the bridge point B_1. Therefore, the locally greedy shortest path is S-P-$\cdots$-B_1. In case 3, there are stops in both semispaces of SA^{Low} while P_2 and B_1 are on the same side of SD. Since B_1 is the exit of SA^{Low}, we should visit the right semispace SS_L^{Low} first and then visit the left semispace SS_R^{Low} to avoid detours. The locally greedy sub-path is S-P_1-$\cdots$-P_2-B_1. (If B_1 is on SD or $B_1 = D$, we choose one of the semispaces randomly.)

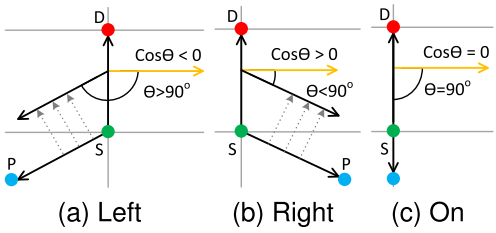


Fig. 9. Orienting P to **SD** in SA^{Low}.

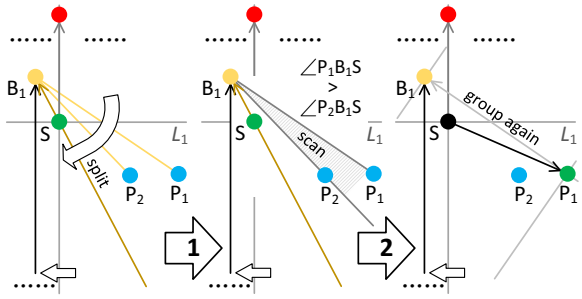


Fig. 10. Split, scan, hop, and group for SA^{Low}.

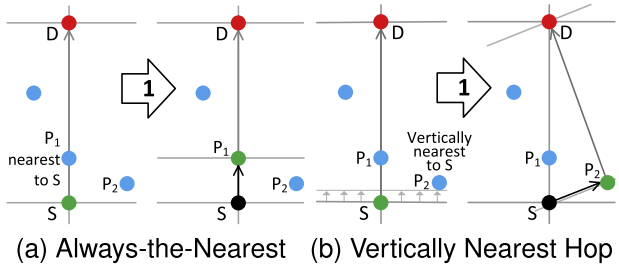


Fig. 11. Two navigation cases of entering SA^{Mid}.

We observe that the navigation trend in the three cases is clockwise. Note that if B_1 is on **SD**'s right side, we have three similar cases, which have trend a counter-clockwise trend. This observation is an important inspiration for OSCH after we experiment on multiple methods, including nearest, vertically/horizontally nearest, and radiation.

Secure Orienting for SA^{Low}: We orient B_1 and $\mathcal{P}^{\text{Low}}$ in Fig. 9 to see which side of **SD** they locate in. Given S, D, and P, we draw two vectors **SD** and **SP**, calculate **SD**'s orthogonal vector **OV** marked yellow in Fig. 9, and compute $\mathsf{Sim}(\mathbf{OV}, \mathbf{SP})$. We determine $Side(P) =$

$$\begin{cases} L, & \mathsf{Sim}(\mathbf{OV}, \mathbf{SP}) \in (-1, 0) \\ R, & \mathsf{Sim}(\mathbf{OV}, \mathbf{SP}) \in (0, 1) \\ On, & \mathsf{Sim}(\mathbf{OV}, \mathbf{SP}) = 0 \end{cases}.$$

Given the encrypted coordinates of S ($[\![x_s]\!], [\![y_s]\!]$), D ($[\![x_d]\!], [\![y_d]\!]$), and P ($[\![x_p]\!], [\![y_p]\!]$), NSP proceeds as follows.

- Retrieve the preciously computed **ESD**.
- Calculate the encrypted **OV**:

$$\mathbf{EOV} = (\mathbf{OV}^1, \mathbf{OV}^2) = ([\![y_d]\!]/[\![y_s]\!], [\![x_s]\!]/[\![x_d]\!]) \quad (9)$$

- Compute the encrypted numerator of $\mathsf{Sim}(\mathbf{SP}, \mathbf{OV})$:

$$\text{ENCS}'^{\mathbf{SP}}_{\mathbf{OV}} = e(\mathbf{EOV}^1, \mathbf{ESP}^1) h_1^r e(\mathbf{EOV}^2, \mathbf{ESP}^2) h_1^{r'}. \quad (10)$$

- Perform comparison on $\text{ENCS}'^{\mathbf{SP}}_{\mathbf{OV}}$ and $[\![r]\!]$ with the proxy for selecting the minimum of $\text{NCS}'^{\mathbf{SP}}_{\mathbf{OV}}$ and r (the same way we compare $\text{NCS}'^{\mathbf{SD}}_{\mathbf{SP_i}}$ and r) to obtain an output cb_3. If $cb_3 = 0$, $Side(P) = R$; otherwise, $Side(P) = L$. Note that we include the case of On to the case of L.

Secure Splitting for SA^{Low}: After secure orienting, we split the stops in the first semispace of SA^{Low} by drawing vector $\mathbf{SB_1}$ as shown in Fig. 10. Next, we orient the stops toward $\mathbf{SB_1}$ to split them into two parts $\mathcal{P}_L^{\text{Semi}}$, $\mathcal{P}_R^{\text{Semi}}$. Note that if we do not split, we may find a P on the wrong side.

Secure Scanning for SA^{Low}: After secure splitting, we scan the P that has the *maximum (minimum)* angle $\angle PB_1S$ when the stops are on $\mathbf{SB_1}$'s *right (left)* side in the three clockwise cases or the P that has the *maximum (minimum)* angle $\angle PB_1S$ when the stops are on $\mathbf{SB_1}$'s *left (right)* side in the three counter-clockwise cases. It is achieved by comparing two angles in plaintexts. However, it is difficult to compare two cosine similarities even by the idea of numerator comparison given that the two denominators are not the same. We transform it to secure orienting as follows.

- *For the three clockwise cases:* choose a stop P randomly, draw vector $\mathbf{PB_1}$, orient the remaining stops in the subgroup toward $\mathbf{PB_1}$: if a stop P' on $\mathbf{PB_1}$'s *right* side is found, set P' as temporary D; otherwise, set P as temporary D.
- *For the three counter-clockwise cases:* choose a stop P randomly, draw vector $\mathbf{PB_1}$, orient the remaining stops in the subgroup toward $\mathbf{PB_1}$: if a stop P' on $\mathbf{PB_1}$'s *left* side is found, set P' as temporary D; otherwise, set P as temporary D.
- If there are more than one stop on L_1 and the right side of **SD**, we choose the nearest stop by first computing the difference between projection of SP_1 on **OV** and projection of SP_2 on **OV**: $\text{ENum}^{\mathbf{P_1},\mathbf{P_2}}_{\mathbf{OV},S} = [\![(x_{p_1} - x_{p_2})(y_d - y_s) + (y_{p_1} - y_{p_2})(x_d - x_s)]\!]_1$, computing $\text{ENum}'^{\mathbf{P_1},\mathbf{P_2}}_{\mathbf{OV},S}$, and then invoking $\mathsf{CT}(r, \text{Num}'^{\mathbf{P_1},\mathbf{P_2}}_{\mathbf{OV},S})$ to select P with a smaller $|SP|$. If the stops are on the left of **SD**, we use $-\mathbf{OV}$ instead.

We exemplify case 2 in Fig. 8(c). There are two stops locating in the same side of $\mathbf{SB_1}$ after splitting as shown in Fig. 10. We choose P_2 first to orient P_1. P_1 is computed on the right side of $\mathbf{PB_2}$, so we set P_1 as temporary D. Given that there are only two stops on $\mathbf{SB_1}$'s *right* side, we hop to P_1 and restart secure grouping.

5) Strategy VNH for SA^{Mid}: Intuitively, we can hop to the stop P_1 nearest to S and then restart secure grouping as shown in Fig. 11(a). However, this approach is not the best for incurring a detour to P_2. We propose to hop to the stop that is vertically nearest to S in the direction of **SD** and set it as new S, as shown in Fig. 11(b). This explains why we chose it as B_1 in secure bridging. After making the vertical hop, we restart secure grouping.

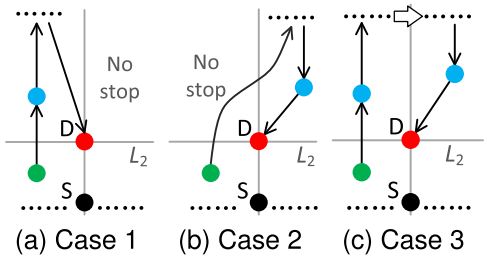


Fig. 12. Three navigation cases of entering SA^{High}.

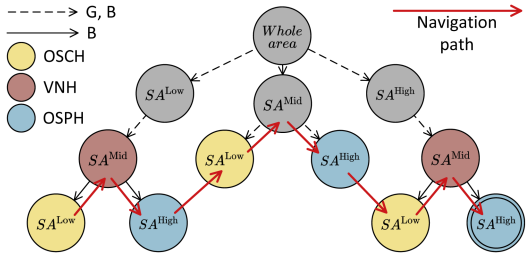


Fig. 14. Search tree via depth-first-search.

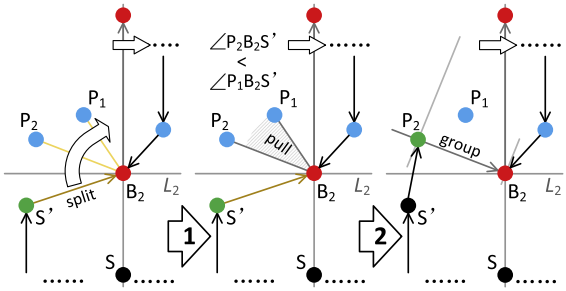


Fig. 13. Split, pull, hop, and group for SA^{High}.

Note that if there are more than one stop having the shortest vertical offset in $\mathcal{P}_R^{\text{Mid}}$ ($\mathcal{P}_L^{\text{Mid}}$), we choose the rightmost (leftmost) P in $\mathcal{P}_R^{\text{Mid}}$ ($\mathcal{P}_L^{\text{Mid}}$). If $\mathcal{P}^{\text{Mid}} = \emptyset$, we choose D which requires one stop back in the secure navigation SN when entering a non-empty $\mathcal{P}^{\text{High}}$. Please refer to Algorithm 8.

6) Strategy OSPH for SA^{High}: The three navigation cases of entering SA^{High} when there are more than one stop in SA^{High} are sketched in Fig. 12. They resemble SA^{Low} and we omit further elaboration.

Secure Orienting Algorithm for SA^{High}: The idea of orienting a stop P in SA^{High} is the same as the one in SA^{Low}. The NSP computes $\text{ENCS}'^{\mathbf{SP}}_{\mathbf{OV}}$, performs a comparison on $\text{ENCS}'^{\mathbf{SP}}_{\mathbf{OV}}$ and $[\![r]\!]$ with the proxy for selecting the smaller one of $\text{NCS}'^{\mathbf{SP}}_{\mathbf{OV}}$ and r, and obtains an output cb_3. If $cb_3 = 0$, $Side(P) = L$; otherwise, $Side(P) = R$.

Secure Splitting for SA^{High}: After secure orienting, we split the stops in the first semispace of SA^{Low} by drawing vector $\mathbf{S'B_2}$ as shown in Fig. 13. Recall that S' is the current starting point and B_2 is D. Next, we orient the stops toward $\mathbf{S'B_2}$ to split them into two parts $\mathcal{P}_L^{\text{Semi}}$, $\mathcal{P}_R^{\text{Semi}}$.

Secure Pulling for SA^{High}: After secure splitting, we scan the P that has the *minimum (maximum)* angle $\angle PB_2S'$ when the stops are on $\mathbf{S'B_2}$'s *left (right)* side in the three clockwise cases. The method is similar to the one in secure scanning for SA^{Low}. Note that if there are no points in $\mathcal{P}^{\text{Mid}}$, i.e., $B_1 = B_2 = D$, we use D as an anchor to pull the points in $\mathcal{P}^{\text{High}}$ clockwise (counter-clockwise) if the last visited P in $\mathcal{P}^{\text{Low}}$ locates on the left (right) of $\mathbf{SD}$.

• If there are more than one stop on L_2, we choose the farthest one by computing the difference of projection of DP_1 on $\mathbf{OV}$ ($-\mathbf{OV}$) and projection of DP_2 on $\mathbf{OV}$ if P_1 and P_2 are on the right (left) side of $\mathbf{SD}$, and then invoke $\text{CT}(r, \text{Num}'^{\mathbf{P_1},\mathbf{P_2}}_{\mathbf{OV},\text{D}})$ to select the stop with a bigger $|DP|$.

7) Summary on the Five Strategies: As depicted in Fig. 14, the whole area is divided into three subareas (grey nodes). By recursively grouping and bridging, we process the nodes until we do not need to group. The connection of red lines form the complete navigation path.

We give a concrete navigation example in Fig. 15. We first use the Always-the-Nearest approach as a baseline to obtain a path with a $dist_{S\text{-}D} = 32.5$. By adopting our divide-and-conquer approach, we recursively invoke Group-Bridge-OSCH/VNH/OSPH. In step 1, we start from S to group 10 unordered stops, compute two bridge points marked in yellow. Given B_1 and $\mathcal{P}^{\text{Low}}$, we visit the left semispace of SA^{Low}. In step 2, we split the left semispace and get two stops on $\mathbf{SB_1}$'s right side. We scan the two stops to obtain the stop P with the minimum angle $\angle PB_1S$. In step 3, we hop to P_1 via OSCH, mark S as black, and mark P_1 as green. In step 4, we restart grouping with P_1 and B_1 to find that there is only one stop P_2 in the new subarea SA^{Low} and the vertically nearest stop in the new SA^{Mid} is P_3. In step 5, we hop to P_2 and group. In step 6, we hop to the vertically nearest stop P_3 by strategy VNH and group. In step 7, we hop to the last stop P_4 in the initial SA^{Low} and then hop to initial B_1. In step 8, we group with P_5 and the initial B_2 (D). From step 9 to step 10, we hop to P_6 and P_7. From step 11 to step 13, we group with P_7 and D, and hop to the only stop P_8 in a new SA^{Mid}. In step 14, we group with P_8 and D. From step 15 to step 16, we apply OSPH to hop to P_9, P_{10}, and D, successively.

Optimization: Since we have B_1 in $\mathcal{P}^{\text{Mid}}$, it is possible to optimize the navigation process by navigating in SA^{Low} and SA^{Mid} simultaneously. If either of them is further grouped, its sub-optimization is potentially possible.

Eight algorithms: We put our designed eight secure algorithms in Appendix, available online. Secure navigation algorithm invokes other seven algorithms. Blue lines indicate recursion entry.

E. Path Recover

After receiving $(Easp_q, Edist_q)$, the user decrypts it to get the query result $(asp_q, dist_q)$ as follows.

- For each triad $\gamma = (\mathcal{I}(P), sp \oplus F_3(H_1(r_{v_p})), Esd)$ after the first $\mathcal{I}(S)$ in the $Easp$, recover sp by retrieving the unique r_{v_p} corresponding to $H_1(P)$ and previous $\mathcal{I}(P)$,

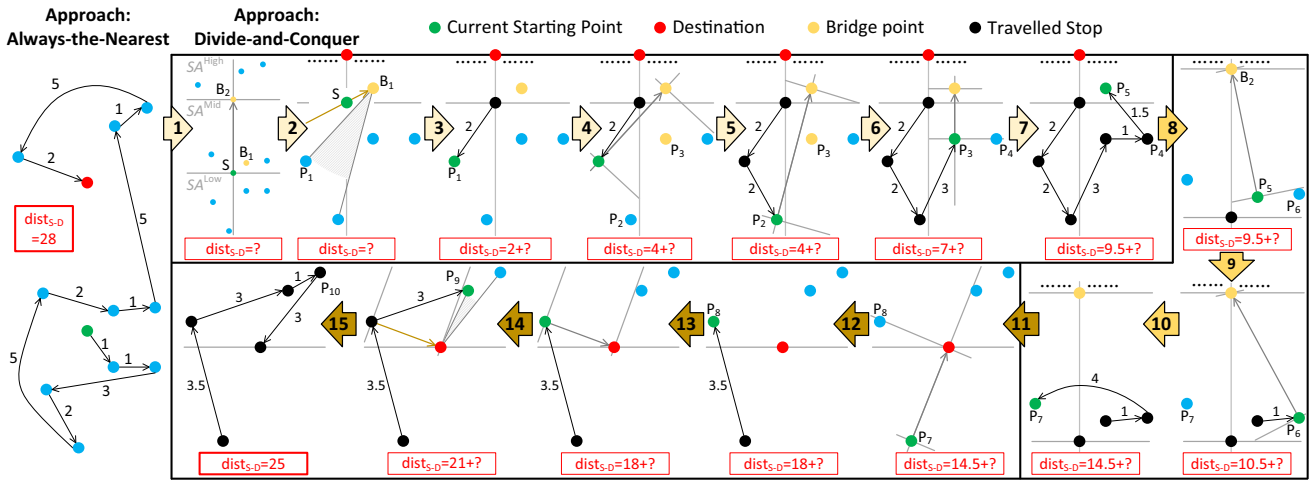


Fig. 15. A detailed example of how the five navigation strategies work.

and computing $sp = \gamma[1] \oplus F_3(H_1(r_{v_p})$; concatenate all recovered sp to form the complete navigation path $asp_q = S\text{-}sp_1\text{-}sp_2\text{-}\cdots\text{-}D$.

- Decrypt $Edist_q$ to get the path distance $dist_q$ in $\mathbb{G}_1$: compute $Edist_q^{q_1} = (g_1^{dist_q} h_1)^{q_1} = (g_1^{q_1})^{dist_q}$ and compute the discrete log of $Edist_q^{q_1}$ base $\hat{g} = g_1^{q_1}$ using Pollard's lambda method [54] or compare with precomputed $\{\hat{g}\}^i$ where $i \in [1, distMax]$. Note that we do not publish $\hat{g}$, so the adversary cannot decrypt $dist_q$ by the brute-force attack.

F. When D and S are Coincidental

SCNP is TSP when $D = S$ and our group strategy does not apply at first glance. This is because the coincidence eliminates a **SD** that groups the remaining stops and sets the stage for G-B-OSCH/VNH/OSPH. To bridge the gap between the coincidental case and our strategies, we take the Initiative to hop from S to the nearest P and create a **PS** as an oxidizer to reignite grouping. In this way, the navigation path is $\mathsf{Path}(S, D) = S\text{-}P\text{-}\mathsf{Path}(P, D)$. We justify the five strategies of our divide-and-conquer approach in Appendix, available online.

VI. Security Analysis

We define two leakage functions $\mathcal{L}_1$ and $\mathcal{L}_2$ as below.

- $\mathcal{L}_1(\mathcal{RG})$: Given a road graph $\mathcal{RG}$, $\mathcal{L}_1(\mathcal{RG}) = (n, X, Y, sd_{max}, sp_{max})$ where X and Y are the coordinate bound in x axis and y axis, respectively; $sp_{max} = \max_{v_i \in \mathcal{V}}\{\max_{\mathcal{PD}_{v_i}^{v_j}} |sp_{v_i v_j}|\}$; $sd_{max} = \max_{v_i \in \mathcal{V}} \{\max_{\mathcal{PD}_{v_i}^{v_j}} sd_{v_i v_j}\}$.
- $\mathcal{L}_2(\mathcal{ERG}, q)$: Given an encrypted graph $\mathcal{ERG}$ and a navigation query q, $\mathcal{L}_2(\mathcal{ERG}, q) = (QP(\mathcal{ERG}, q), AP(\mathcal{ERG}, q))$, where $QP(\mathcal{EG}, q)$ is the query pattern and $AP(\mathcal{EG}, q)$ is the access pattern (similar to sketch pattern [11] and access pattern [13]).

Definition 2 (Query Pattern): The query pattern reveals whether the stops in the query have appeared before. For two queries q_i, q_j, define $\mathsf{sim}(q_i, q_j) = (S_i = S_j, D_i = D_j, P_{i1} = P_{j1}, P_{i2} = P_{j2}, \ldots, P_{ik} = P_{jk})$, i.e., whether each of the locations $(S_i, i_0, P_{i1}, P_{i2}, \ldots, P_{ik})$ matches each of the locations of $(S_j, D_j, P_{j1}, P_{j2}, \ldots, P_{jk})$. Let $\mathbf{q} = (q_1, q_2, \ldots, q_z)$ be a non-empty sequence of queries. $\mathcal{L}_1(\mathcal{RG}, q)$ outputs a $z \times z$ symmetric matrix that for $1 \le i, j \le z$, the element in the ith row and jth column equals $\mathsf{sim}(q_i, q_j)$. Namely, the query pattern reveals whether the stops in the query have appeared before.

Definition 3 (Access Pattern): For an encrypted road graph $\mathcal{ERG}$ and a navigation query q, access pattern is defined as $AP(\mathcal{ERG}, q) = \{id(\mathcal{EQR}_q)\}$, where $id(\mathcal{EQR})$ is the identifiers of stops in the encrypted query result $\mathcal{EQR}$.

Theorem 1: If P and F are pseudo-random, and Ω is CPA-secure, Hermes is adaptively $(\mathcal{L}_1, \mathcal{L}_2)$-semantically secure in the random oracle model, where $\mathcal{L}_1(\mathcal{RG}) = (n, X, Y, sd_{max}, sp_{max})$ and $\mathcal{L}_2(\mathcal{ERG}, q) = (QP(\mathcal{ERG}, q), AP(\mathcal{ERG}, q))$.

Now we construct a simulator $\mathcal{S}$ as below.

• Given $\mathcal{L}_1(\mathcal{RG})$, $\mathcal{S}$ samples $\delta_0 \xleftarrow{\$} \{0,1\}^{x+y+\lambda}$, where x, y, and λ correspond to the bit length of x coordinate, y coordinate, and security parameter. x and y are chosen from $[0, \log X]$ and $[0, \log Y]$, respectively. For all $1 \le i \le n$, $\mathcal{S}$ samples $\delta_i \xleftarrow{\$} \{0,1\}^{|h|+|F|+|\mathsf{Enc}|+\lambda}$. Here, $|h|$, $|F|$, and $|\mathsf{Enc}|$ correspond to the bit length of the hash function, path distance, ciphertext from the encryption scheme Ω, and stops on the shortest path (excluding S and D), respectively. r is chosen from $[0, sp_{max}]$ randomly and $r = 0$ refers to a direct and shortest path from S to D. The shortest distance is chosen from $[0, sd_{max}]$ randomly. $\mathcal{S}$ initializes an array Arr_1. $\mathcal{S}$ stores δ_0 in $Arr_1[0]$ and stores $\{\delta_i\}_{i=1}^n$ in the remaining of Arr_1 after a random permutation on $[1, n]$.

• $\mathcal{S}$ samples $\delta_1 \xleftarrow{\$} \{0,1\}^{x+y+\lambda}$. $\mathcal{S}$ selects one of the first $|h|$ bits of δ in the last step and concatenates with a random string sampled from $\{0,1\}^{|F|+|\mathsf{Enc}|+\lambda}$ to obtain δ_i. The same $\{0,1\}^{|h|}$ is selected only once.

• $\mathcal{S}$ repeats the step above $n-2$ times to obtain N arrays in total $\{Arr_1, Arr_2, \ldots, Arr_n\}$. For $1 \le i \le n$, $\mathcal{S}$ samples $dx_i \xleftarrow{\$} \{0,1\}^{\log n}$ without repetition and sets $DX^*[dx_i] \xleftarrow{\$} \{0,1\}^{\log n+\lambda}$. Finally, $\mathcal{S}$ outputs $\mathcal{EG}^* = (DX^*, Arr^*)$.

TABLE II
PARAMETER SETTINGS (BOLD: DEFAULT VALUES)

Parameter	Value
N	$2000i$, i = 2,3,4,5,6,7,8,9,10
k	1, 2, **3**, 4, 5, 6, 7, 8, 9
$\|sk\|$, $\|pk\|$, $\|k_i\|$, $\|T\|$	256, 520, 1024, log$n$
r	$[1000, 10000]$

• Given $\mathcal{L}_2(\mathcal{EG}, q)$, $\mathcal{S}$ checks whether q has been queried before. If so, $\mathcal{S}$ returns the previously token; otherwise, $\mathcal{S}$ samples an integer k in PPT as follows.

$\mathcal{S}$ sets $\mathcal{I}(S)[0] = dx_i$ for a previously unused dx_i. $\mathcal{S}$ generates a unused $\alpha \in [n]$, a key $k_4^* \xleftarrow{\$} \{0,1\}^\lambda$, and sets $\mathcal{I}(S)[1] = DX[dx_i] \oplus (\alpha || k_s)$. $\mathcal{S}$ generates a random value from $\{0,1\}^{|h|}$ as $\mathcal{I}(S)[2]$. $\mathcal{S}$ records the association $(k_s, AP(\mathcal{EG}, q), n)$. $\mathcal{S}$ does the same for D and k stops.

$\mathcal{S}$ simulates the H_2 as follows. Given (k, v), $\mathcal{S}$ checks whether k has been queried before and any entry in an Arr is (s, r) where s is a $(|h| + |F| + |\mathsf{Enc}|)$-bit string. If k is new and there is an appropriate entry (s, r) in Arr corresponding to α, $\mathcal{S}$ outputs $s \oplus (\eta, \{0,1\}^{\log|sp_{max}|}, [\![0]\!])$, where η is in a random location of Arr associated with k and $AP(\mathcal{EG}, q)$, $\{0,1\}^{\log|sp_{max}|}$ is a random string of a path, $[\![0]\!]$ is $\Omega.\mathsf{Enc}(0)$. If no appropriate entry exists, $\mathcal{S}$ returns a random value.

Therefore, the simulated view and the real view are indistinguishable by $\mathcal{A}$ such that Hermes is adaptively $(\mathcal{L}_1, \mathcal{L}_2)$-semantically secure in the random oracle model against an adaptive adversary, i.e., we deduce the correctness of Theorem 1 from the pseudo randomness of P and F and the CPA-security of Ω. □

VII. PERFORMANCE

A. Experimental Settings

Dataset: We downloaded the city map of Los Angeles from OpenStreetMap, which consists of 2200 intersections, 986 roads, and 6448 road segments. We preprocessed it to obtain 9 road graphs (see github.com/SecureNavigation/Hermes).

Parameters: We list the key experimental parameters in Table II, including number of road intersections (nodes in road graph) N, number of unordered stops k, length of the private/public key sk/pk, length of the secret keys k_1, k_2, k_3, k_4, and length of the PRP T. Note that r is the random number we use for secure comparison. Since the message space of Ω is $\{0, 1, \ldots, M\}$ with $M < q_2$, we set r as a positive integer large enough for decrypting $Edist$ successfully.

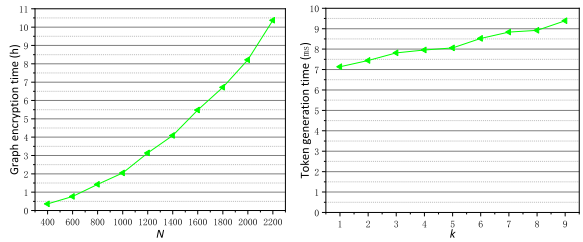
Metrics: For efficiency, we measure the computational cost of graph encryption, token generation, path search, and path recover, measure the communication overhead, and test how the cost and overhead scale with N and k. For accuracy, we compute distance deviation rate as $ddr = |dist - sd|/sd$ and path similarity $ps = \sum_{i=1}^{k} b_i/k$ where $b_i = 1$ if the ith stop in asp equals to the ith stop in sp and $b_i = 0$ otherwise.

Baselines: Since our navigation problem is different from the related work and the cryptographic primitives are not applicable to the TSP solutions [20], [30], [31], we design six baselines for performance comparison based on our navigation strategies. Specifically, they are carefully designed to highlight the role of grouping, bridging, splitting, and vertically nearest hopping. The baselines include 1) Always-the-Nearest (A), i.e., a basic greedy approach; 2) G+A (GA) groups and invokes A; 3) GB-A includes bridge in (2); 4) GB-OSCH-A-A, 5) GB-A-VNH-A, and 6) GB-A-A-OSPH are designed to test OSCH, VNH, and OSPH, respectively. We do not include Full Permutation (FP) because it incurs too much burden, e.g., its search time is *two hours* when $N = 1000, k = 7$. Besides, we also compare Hermes with TSP and Path TSP in the plaintext domain to evaluate the search efficiency and accuracy.

Setup: We instantiate NSP on a server (Windows Server 2021 R2 Datacenter, a 3.7-GHz Intel(R) Core(TM) i7-8770 K processor, and 32 GB RAM). We use JPBC library [55] to implement basic cryptographic primitives and jPBC [56] for encryption preprocessing. We instantiate Ω based on finite groups of composite order that support a bilinear map proposed by Boneh et al. [27]. We use SHA256 as the hash function H_1 and instantiate the random oracle H_2 with HMAC − SHA256. We use HMAC for the pseudo-random function F. The secure comparison circuit is built from FastGC [57].

TABLE III
TIME COMPLEXITY OF HERMES

Step	Pre-Process	Graph encryption	Token generation	Path query	Path recover
T.C.	$O(N^3)$	$O(N^2)$	$O(k)$	$O(k^2)$	$O(k)$



(a) Graph enc. varying N (b) Token generation varying k

Fig. 16. User's C.C. in graph encryption and token generation.

B. Efficiency

Computational Cost (C.C.): We list the time complexity (T.C.) in Table III. Preprocessing invokes the Floyd-Warshall algorithm. Its T.C. is $O(N^3)$ with time being 2.5s when $N = 1000$. *Graph Encryption*. It includes sd matrix encryption, node axis encryption, and DX (Arr) generation. The T.C. of the last one is $O(N)$ while the T.C. of the former two is $O(N^2)$. However, we optimize the matrix encryption by setting $Edist_{ij} = Edist_{ji}$, making the overall T.C. linear for smaller N. Since the sd matrix encryption dominates graph encryption, the overall T.C. tends to be $O(N^2)$ as N increases. The graph encryption time is 2 hours when $N = 1000$ (Fig. 16(a)). *Token Generation* costs user 7.81 ms when $N = 1000, k = 3$, which increases with k (Fig. 16(b)).

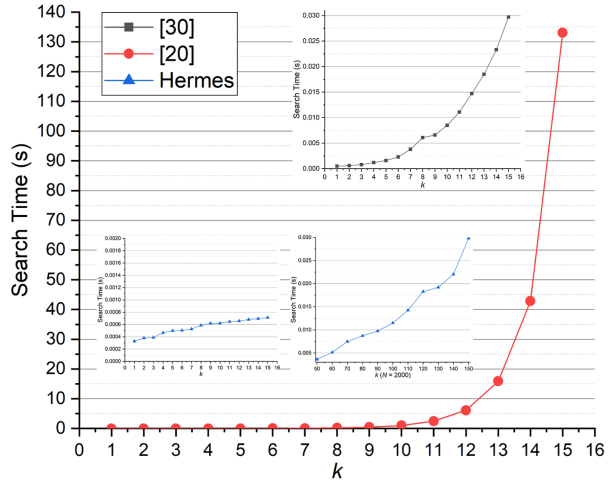


Fig. 17. User's C.C. in path search on plaintexts.

Path Search: The search complexity of existing schemes in k-stops shortest paths is $O(k^3)$ for TSP [30] and $O(2^N)$ for Path TSP [20]. We have conducted experiments to show the time costs of Hermes, TSP [30], and Path TSP [20] in the plaintext domain. We vary N from 3 to 17 and k from 1 to 15, following [20]. In Path TSP, N represents the total number of nodes, including k intermediate nodes, start node, and end node, i.e., $N = k + 2$. The results in Fig. 17 confirm the theoretical analysis. [20] shows exponential growth $(O(2^N))$ with path search time increasing rapidly from 0.0058s to 133.08s as k grows. We limit k to a maximum of 15 because the runtime of [20] exceeds 2 minutes when $k > 15$, making it impractical for a larger k. [30] shows cubic growth $O(k^3)$, with runtime rising from 0.0005 to 0.0297s. Hermes does not strictly follow the quadratic growth $O(k^2)$ when $k \in [1, 15]$ because the runtime is too small in the plaintext domain. But it exhibits its pattern as k increases, with runtime growing from 0.0036 to 0.0298s for $k \in [50, 150]$.

The SP recursively invokes the SN at most k times $(O(k))$. Assume there are k_1, k_2, and k_3 stops in $\mathcal{P}^{\text{Low}}$, $\mathcal{P}^{\text{Mid}}$, and $\mathcal{P}^{\text{High}}$, respectively. At first, the SP groups k stops $(O(k))$, computes two bridge points $(O(k_1 + k_2))$, and enters one subarea. For SA^{Low}, the SP orients k_1 stops $(O(k_1))$, splits one chosen semispace $(\leq O(k_1))$, scans $\leq k_1$ stops $(O(k))$, and hops $(O(1))$; for SA^{Mid}, the SP only hops $(O(1))$ because B_1 is already computed; for SA^{High}, the SP orients k_3 stops $(O(k_3))$, splits one chosen semispace $(\leq O(k_3))$, pulls $\leq k_3$ stops $(O(k_3))$, and hops $(O(1))$. As the number of unvisited stops decreases, the T.C. of path search is $< O(k * (k+k_1+k_2+\text{Max}(3k_1+1, 1, 3k_3+1))) = O(k^2)$. This is reflected in Fig. 18(a) where the line tends to be linear when $k \leq 7$ because the k is not large. Since the stop distribution (locate in which subarea) affects the search, we select some typical distributions and compute an average time. When $k = 5$, $N = 1000$, the search time is 7.58 s (Fig. 18(a)).

Search Optimization: We optimize searching by navigating in $\mathcal{P}^{\text{Low}}$ and in $\mathcal{P}^{\text{Mid}}$ simultaneously. The optimization effect is more apparent as k increases. The search time for $N = 1000, k = 9$ is 20.12 s, which is reduced to 18.48 s after parallel processing. The search time is not affected by N because (1) N only decides DX size, (2) search correlates DX in looking for sds and sps that are completed by using indexes only (Fig. 18(b)).

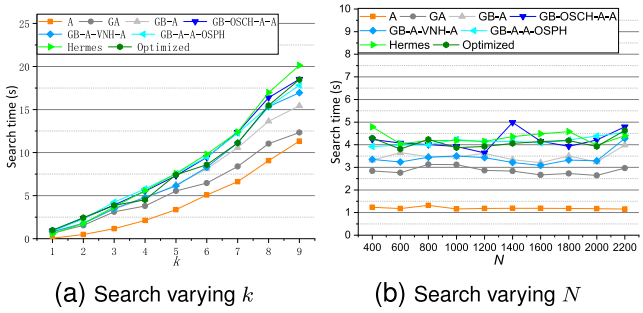


Fig. 18. C.C. of NSP and proxy.

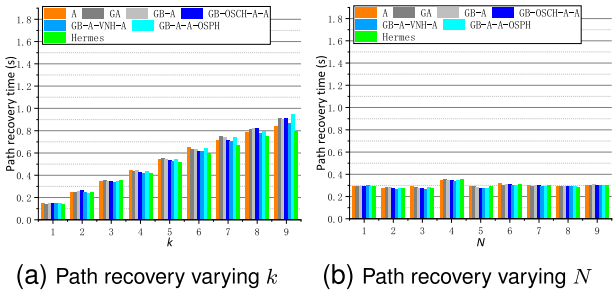


Fig. 19. C.C. of user in path recovery.

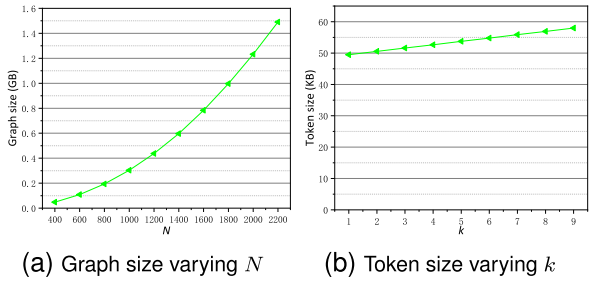
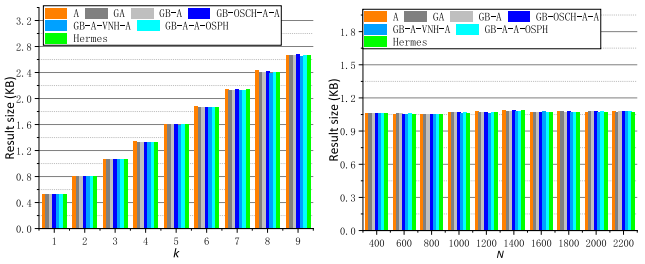


Fig. 20. C.O. of user.

Path Recovery: It costs user 0.54 s for path recovery, which increases with k (Fig. 19(a)). Path recovery is not directly related to N, but the decryption of Esd encrypted from different sd leads to the fluctuations in recovery time (around 0.55s in Fig. 19(b)).

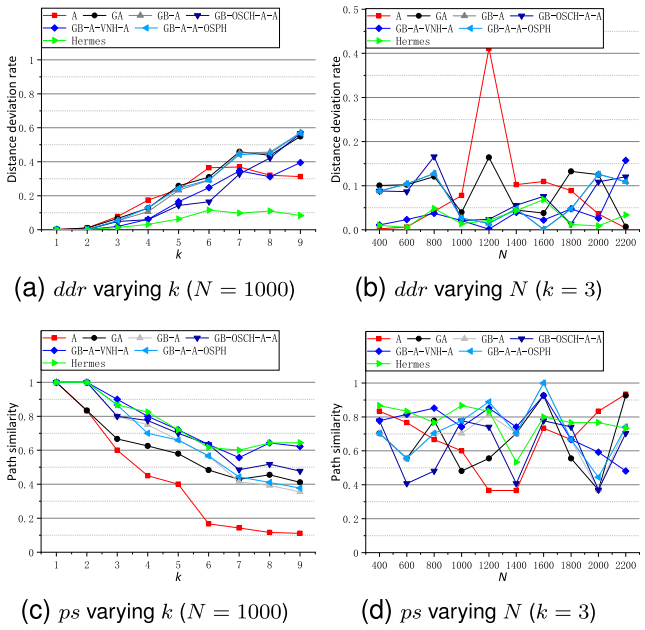
Communication Overhead (C.O.): The graph size only increases with N. When $N = 1000$, the graph size is 0.3GB (Fig. 20(a)). The token size only grows with k. When $k = 3$, the token size is 51.6 KB (Fig. 20(b)). The result size grows linearly with k. When $k = 3$, the result size is 1.08 KB (Fig. 21(a)). It stays the same when N varies (Fig. 21(b)). It is not affected by N because the length of $Easp$'s triads $\{\gamma\}$ and $Edist$ stay the same.

Scalability: The C.C. and C.O. increase with N and k (Table III). Although graph encryption consumes some time and storage, it is only one time operation. The key operation is path



(a) Result size varying k (b) Result size varying N

Fig. 21. C.O. of NSP.



(a) ddr varying k ($N = 1000$) (b) ddr varying N ($k = 3$)

(c) ps varying k ($N = 1000$) (d) ps varying N ($k = 3$)

Fig. 22. Navigation accuracy.

search of a T.C. $O(k^2)$. After optimization, we further reduce the search time. This level of T.C. roots in our navigation strategies and it is better than the one of FP ($O(k!)$). For most users, $k \in [1, 5]$ is already a popular choice for their own SCN (see ballot 2in Section VIII-A).

Accuracy: We first compute the ground truths based on euclidean distance by manually checking all $k!$ permutations of k stops in plaintext. For each k, we select its typical distributions, check at least 10 cases in each distribution.

As shown in Fig. 22(a), the ddr is less than 6.4% when $k \leq 5$ and its maximum is 11.5% when $k = 6$. We attribute this phenomenon to our distance assumption. We assume that the sp between two stops is a straight line. While in reality, it does not always hold, which leads to the problem of how the map layout affects the navigation accuracy. The more grid pattern the layout follows, the more accurate the navigation will be. Such layout can be found in cities like Seattle, Washington, and Toronto. When $k = 1$, ddr is not 0, but 0.05%, which is caused by the precision loss during coordinate transformation. As shown in Fig. 22(c), the sp is relatively high and it is 64.4% even when $k = 9$. When $k > 6$ ($k > 7$), the ddr (sp) does not show monotone increasing (decreasing) due to the specific stop distributions. The ddr and sp are not affected much by N as in Fig. 22(b) and (d). Some singular points appear in the results due to our distance assumption and specific stop distributions, which indicate a future research direction. Although we do not acquire the shortest path every time, the promising accuracy corroborates that our strategies have grasped the essence of solving the SCNP.

C. Comparison With Six Baselines

Since the graph encryption and token generation are the same for all schemes, we skip the two aspects in comparison. For Always-the-Nearest (A), we design a similar secure comparison circuit where the NSP sends $[\![dist_1 - dist_2 + r]\!]$ to the proxy to compare $dist_1$ and $dist_2$. For path recovery and result size with varying k and N, the results of all schemes are similar. For search time, A has a T.C. of $O(k^2)$, but it requires less computation. The other five baselines share the same T.C. being a simplified version of Hermes. A has the worst accuracy (Fig. 22) for being a pure greedy strategy. GA and GB-A show an improvement over A (Fig. 22(a)) that in turn validates G and B. The last three baselines also exhibit an advantage over GB-A that further verifies the effectiveness of three subarea strategies. Hermes exceeds six variants most cases by showing a lower distance deviation rate and a higher path similarity. Combined with the results from comparison with TSP and Path TSP in Section VII-B, we say that our approach improves efficiency and does not introduce unnecessary computational overhead.

We do not compare the efficiency of using homomorphic encryption and garbled circuits in existing schemes in k-stops shortest paths because either existing schemes do not address SCNP or the two cryptographic techniques are not applicable to their scheme, e.g., R-Tree [30] and spanning tree [20]. For instance, introducing them in the R-tree search process will hinder the comparison of the distance between the minimum bounding rectangle and the query point, making it impossible to prioritize the next query tree node.

VIII. Discussions

A. Two Assumptions of k Unordered Stops

We initiated a ballot on the Mechanic Turk: *Do you think it is a good feature or common need to add multiple unordered stops between the starting point and the destination on Google Maps?* From April 4, 2022 to May 31, 2022, we collected 2000 answers and results show that 1486 workers (74.3%) chose "Yes", indicating that most workers agree with our assumption. We initiated a second ballot *What is the maximum number of stops do you need between starting point and destination in a navigation service?* 888 workers (67.5%) out of 1316 workers choose from [1, 5].

B. Acceptable Response Time

We initiated a third ballot for *What is your acceptable response time (excluding network delay) when you use a navigation app?* with ten answers (from 1 s to 10 s). 1590 workers

(80%) out of 1991 workers accept the maximum response time in [3 s, 10s]. Ballot 2 and ballot 3 indicate that our search time is acceptable even under our delicate navigation strategies over encrypted road graphs.

C. Static versus Dynamic Data

We only consider static data in this work because we mainly focus on defining the SCNP and then proposing a scheme to achieve security, accuracy, and efficiency under SCNP. The dynamic setting is definitely a direction worth investigating. There are two cases where the data can be dynamic. The first one is dynamic edges, i.e., roads can be newly built, temporarily blocked, or removed. Such a case necessitates securely edge updating in the encrypted road graph. The second one is dynamic weights, i.e., traffic congestion on roads can vary over time. A potential solution is to leverage efficient homomorphic encryption that supports both ciphertext computation and ciphertext update. By doing so, we can dynamically adjust navigation routes by calculating the travel cost over encrypted traffic data while not sacrificing efficiency. Besides, we plan to research several variants of Hermes, e.g., SCNP with some ordered stops (e.g., visit stop 1 before stop 3, visit stop 5 after stop 6), SCNP with geographic constraints (e.g., a river lies between stop 7 and stop 8), and SCNP with time constraints (e.g., arrive at stop 2 in 15 minutes after departure).

IX. Some Justification

Justification of Group: Our navigation is a divide-and-conquer approach that groups k unordered stops into three subareas and navigates in them sequentially. In other words, we visit from SA^{Low}, to SA^{Mid}, and to SA^{High}. The correctness of Group is straightforward via proof by contradiction. If we break the order of visiting the three subareas, i.e., low-mid-high, we will end up with detours that come back and forth between subareas. It also indicates that the ideal number of navigation paths from one subarea to another is just one.

Justification of Bridge: After grouping, we connect three subareas by linking subpaths. It naturally requires to determine S and D for each subarea. Otherwise, we cannot compute a subpath in each subarea, which we introduce bridge points. For SA^{Low}, its S is the initial S. We compute the bridge point B_1 in SA^{Mid} to be SA^{Low}'s D, which is determined by the strategy VNH for SA^{Mid}. For SA^{Mid}, its S is B_1. We set its *target destination*, i.e., B_2, as the initial D. The target destination means we stop at the last visited stop in SA^{Mid} and do not hop to B_2 when $\mathcal{P}^{\text{High}} \neq \emptyset$. For SA^{High}, we set its S as the last unordered stop in SA^{Mid} and set its D as the initial D. Given the carefully chosen bridge points, we divide the navigation task into three subtasks, compute three subpaths from which we stitch a complete navigation path.

Justification of OSCH for SA^{Low}: The initial navigation in SA^{Low} looks like an irregular circle that starts from S, crosses an orthogonal line L_1, and lands at B_1 in SA^{Mid}. Intuitively, we want to avoid as many detours as possible. As illustrated in Fig. 8, the subpath in SA^{Low} is closely correlated with two factors: the orientation of B_1 and two semispaces in SA^{Low}. First, the three cases in Fig. 8 show that B_1 and two semispaces obviously determine the priority of one semispace over the other, which requires orienting B_1 and $\mathcal{P}^{\text{Low}}$. Second, it is optimal to traverse the "most right" stop (case 1&2) when B_1 is on **SD**'s left side, or the "most left" stop when B_1 is on **SD**'s right side. *The correctness of this greedy stop roots in drawing a convex/concave polygon that conforms to a specific trend.* After experimenting on multiple methods, including nearest, vertically/horizontally nearest, and radiation. Based on this, we propose to split the chosen semispace and scan toward $\mathbf{SB_1}$ clockwise/counter-clockwise to find the next-to-visit stop. We also considered three cases where multiple stops are on L_1, L_2, and B_1S.

Justification of VNH for SA^{Mid}: This strategy is invoked when computing the first-to-visit stop in SA^{Mid}. The initial navigation in SA^{Mid} is different from the one in SA^{Low} and SA^{High}, which looks like a zigzag. The pattern difference roots in the different stop distribution between S and D. Recall that L_1 and L_2 are two lines orthogonal to **SD** that cross S and D, respectively. After grouping, the stops in SA^{Low} (SA^{High}) are on or below L_1 (on or above L_2). However, all the stops in SA^{Mid} are strictly scattered between L_1 and L_2. Such a distribution calls for a different navigation strategy for SA^{Mid}. Given the observation in Fig. 12, we make one locally optimal move by hopping to the vertically nearest stop. We also consider the case where multiple stops are vertically nearest.

Justification of OSPH for SA^{High}: The navigation path in SA^{High} shares a similar pattern of starting from the last stop in SA^{Mid}, crossing an orthogonal line L_2, and ending at D. It resembles OSCH and we omit further elaboration.

X. Conclusion

In this paper, we are motivated by real-life navigation to propose SCN and its formal definition. We design a novel scheme named Hermes to achieve SCN while balancing accuracy, security, and efficiency. The navigation is based on a powerful insight: navigating through k unordered stops follows a divide-and-conquer pattern that is expressed by recursively invoking five well-designed navigation strategies considering accuracy and efficiency. The road graph and navigation queries are secured by seamlessly integrated cryptographic techniques targetedly chosen for efficiently serving the underlying navigation strategies. Both theoretical analysis and extensive experiments confirm its accuracy, security, and efficiency. We believe this work will spawn a line of research on SCN that can be fruitfully extended to related areas.

References

[1] T. W. Chim, S. Yiu, L. C. K. Hui, and V. O. K. Li, "VSPN: VANET-based secure and privacy-preserving navigation," *IEEE Trans. Comput.*, vol. 63, no. 2, pp. 510–524, Feb. 2014.

[2] M. Li, Y. Chen, S. Zheng, D. Hu, C. Lal, and M. Conti, "Privacy-preserving navigation supporting similar queries in vehicular networks," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 2, pp. 1133–1148, Mar./Apr. 2022.

[3] Google, *Google Maps*, 2022. [Online]. Available: https://www.google.com/maps

[4] L. Ceci, *Leading Mapping Apps in the United States in 2021, by Downloads*, 2022. [Online]. Available: https://www.statista.com/statistics/865413/most-popular-us-mapping-apps-ranked-by-audience

[5] M. Graham and J. Elias, *How Google's $150 Billion Advertising Business Works*, 2021. [Online]. Available: https://www.cnbc.com/2021/05/18/how-does-google-make-money-advertising-business-breakdown-.html

[6] K. C. K. Lee, W. Lee, H. V. Leong, and B. Zheng, "Navigational path privacy protection: Navigational path privacy protection," in *Proc. 18th ACM Conf. Inf. Knowl. Manage.*, D. W. Cheung, I. Song, W. W. Chu, X. Hu, and J. Lin, Eds., Hong Kong, China, 2009, pp. 691–700.

[7] M. E. Andrés, N. E. Bordenabe, K. Chatzikokolakis, and C. Palamidessi, "Geo-indistinguishability: Differential privacy for location-based systems," in *Proc. 2013 ACM SIGSAC Conf. Comput. Commun. Secur.*, A. Sadeghi, V. D. Gligor, and M. Yung, Eds., Berlin, Germany, 2013, pp. 901–914.

[8] Y. Cao, Y. Xiao, L. Xiong, L. Bai, and M. Yoshikawa, "Protecting spatiotemporal event privacy in continuous location-based services," *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 8, pp. 3141–3154, Aug. 2021.

[9] R. Shokri, G. Theodorakopoulos, J. L. Boudec, and J. Hubaux, "Quantifying location privacy," in *Proc. 32nd IEEE Symp. Secur. Privacy*, Berkeley, CA, USA, 2011, pp. 247–262.

[10] I. Bilogrevic, K. Huguenin, S. Mihaila, R. Shokri, and J. Hubaux, "Predicting users' motivations behind location check-ins and utility implications of privacy protection mechanisms," in *Proc. 22nd Annu. Netw. Distrib. Syst. Secur. Symp.*, San Diego, CA, USA, 2015, pp. 1–12.

[11] X. Meng, S. Kamara, K. Nissim, and G. Kollios, "GRECS: Graph encryption for approximate shortest distance queries," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, I. Ray, N. Li, and C. Kruegel, Eds., 2015, pp. 504–517.

[12] D. J. Wu, J. Zimmerman, J. Planul, and J. C. Mitchell, "Privacy-preserving shortest path computation," in *Proc. 23rd Annu. Netw. Distrib. Syst. Secur. Symp.*, San Diego, CA, USA, 2016, pp. 1–15.

[13] Q. Wang, K. Ren, M. Du, Q. Li, and A. Mohaisen, "SecGDB: Graph encryption for exact shortest distance queries with efficient updates," in *Proc. 21st Int. Conf. Financial Cryptogr. Data Secur.*, A. Kiayias, Ed., Springer, Sliema, Malta, 2017, pp. 79–97.

[14] M. Du, S. Wu, Q. Wang, D. Chen, P. Jiang, and A. Mohaisen, "GraphShield: Dynamic large graphs for secure queries with forward privacy," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 7, pp. 3295–3308, Jul. 2022.

[15] E. Ghosh, S. Kamara, and R. Tamassia, "Efficient graph encryption scheme for shortest path queries," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, J. Cao, M. H. Au, Z. Lin, and M. Yung, Eds., 2021, pp. 516–525.

[16] Google, *Google Maps Help-Add Multiple Destinations*, 2023. [Online]. Available: https://support.google.com/maps/answer/144339?co=GENIE.Platform%3DAndroid&hl=en

[17] Uber, *Uber-Extra Stops*, 2023. [Online]. Available: https://www.uber.com/us/en/ride/how-it-works/multiple-stops

[18] N. Staff, *How to Add a Stop in Current Directions in Google Maps App*, 2019. [Online]. Available: https://nerdschalk.com/how-to-add-stops-in-curent-route-directions-in-google-maps

[19] Turk, *Amazon Mechanical Turk*, 2023. [Online]. Available: https://www.mturk.com

[20] R. Zenklusen, "A 1.5-approximation for path TSP," in *Proc. 13th Annu. ACM-SIAM Symp. Discrete Algorithms*, 2019, pp. 1539–1549.

[21] S. Wang, X. Xiao, Y. Yang, and W. Lin, "Effective indexing for approximate constrained shortest path queries on large road networks," in *Proc. VLDB Endowment*, vol. 10, no. 2, pp. 61–72, 2016.

[22] Y. Yuan, X. Lian, G. Wang, Y. Ma, and Y. Wang, "Constrained shortest path query in a large time-dependent graph," in *Proc. VLDB Endowment*, vol. 12, no. 10, pp. 1058–1070, 2019.

[23] Q. Gong, H. Cao, and P. Nagarkar, "Skyline queries constrained by multi-cost transportation networks," in *Proc. 35th IEEE Int. Conf. Data Eng.*, 2019, pp. 926–937.

[24] S. Lu, B. He, Y. Li, and H. Fu, "Accelerating exact constrained shortest paths on GPUs," in *Proc. VLDB Endowment*, vol. 14, no. 4, pp. 547–559, 2020.

[25] Z. Liu, L. Li, M. Zhang, W. Hua, P. Chao, and X. Zhou, "Efficient constrained shortest path query answering with forest hop labeling," in *Proc. 37th IEEE Int. Conf. Data Eng.*, Chania, Greece, 2021, pp. 1763–1774.

[26] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[27] D. Boneh, E. Goh, and K. Nissim, "Evaluating 2-DNF formulas on ciphertexts," in *Proc. 2nd Theory Cryptogr. Conf.*, J. Kilian, Ed., Springer, Cambridge, MA, USA, 2005, pp. 325–341.

[28] A. Yao, "Protocols for secure computations (extended abstract)," in *Proc. 23rd Annu. Symp. Found. Comput. Sci.*, Chicago, IL, USA, 1982, pp. 160–164.

[29] A. Yao, "How to generate and exchange secrets (extended abstract)," in *Proc. 27th Annu. Symp. Found. Comput. Sci.*, Toronto, Canada, 1986, pp. 162–167.

[30] M. Terrovitis, S. Bakiras, D. Papadias, and K. Mouratidis, "Constrained shortest path computation," in *Proc. Int. Symp. Spatial Temporal Databases*, Springer, 2005, pp. 181–199.

[31] V. Traub, J. Vygen, and R. Zenklusen, "Reducing path TSP to TSP," in *Proc. 52nd Annu. ACM SIGACT Symp. Theory Comput.*, 2020, pp. 14–27.

[32] M. L. Fredman and R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *J. ACM*, vol. 34, no. 3, pp. 596–615, 1987.

[33] D. Boneh, C. Gentry, S. Halevi, F. Wang, and D. J. Wu, "Private database queries using somewhat homomorphic encryption," in *Proc. 11th Int. Conf. Appl. Cryptogr. Netw. Secur.*, M. J. J. Jr., M. E. Locasto, P. Mohassel, and R. Safavi-Naini, Eds., Springer, 2013, pp. 102–118.

[34] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft, "Privacy-preserving ridge regression on hundreds of millions of records," in *Proc. 2013 IEEE Symp. Secur. Privacy*, Berkeley, CA, USA, 2013, pp. 334–348.

[35] Y. Elmehdwi, B. K. Samanthula, and W. Jiang, "Secure k-nearest neighbor query over encrypted data in outsourced environments," in *Proc. IEEE 30th Int. Conf. Data Eng.*, I. F. Cruz, E. Ferrari, Y. Tao, E. Bertino, and G. Trajcevski, Eds., 2014, pp. 664–675.

[36] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *Proc. ACM Conf. Comput. Commun. Secur.*, T. Yu, G. Danezis, and V. D. Gligor, Eds., Raleigh, NC, USA, 2012, pp. 965–976.

[37] D. Cash, S. Jarecki, C. S. Jutla, H. Krawczyk, M. Rosu, and M. Steiner, "Highly-scalable searchable symmetric encryption with support for boolean queries," in *Proc. 33rd Annu. Cryptol. Conf.*, R. Canetti and J. A. Garay, Eds., Springer, Santa Barbara, CA, USA, 2013, pp. 353–373.

[38] R. Bost, "$\sum$o φ oς: Forward secure searchable encryption," in *Proc. 2016 ACM SIGSAC Conf. Comput. Commun. Secur.*, E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, Eds., Vienna, Austria, 2016, pp. 1143–1154.

[39] R. Bost, B. Minaud, and O. Ohrimenko, "Forward and backward private searchable encryption from constrained cryptographic primitives," in *Proc. 2017 ACM SIGSAC Conf. Comput. Commun. Secur.*, B. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds., Dallas, TX, USA, 2017, pp. 1465–1482.

[40] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," in *Proc. 13th ACM Conf. Comput. Commun. Secur.*, A. Juels, R. N. Wright, and S. D. C. di Vimercati, Eds., Alexandria, VA, USA, 2006, pp. 79–88.

[41] M. Chase and S. Kamara, "Structured encryption and controlled disclosure," in *Proc. 16th Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, M. Abe, Ed., Springer, 2010, pp. 577–594.

[42] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proc. Int. Conf. Theory Appl. Cryptographic Techn.*, J. Stern, Ed., Springer, Prague, Czech Republic, 1999, pp. 223–238.

[43] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*, 3rd ed. London, U.K./Boca Raton, FL, USA: Chapman & Hall/CRC Press, 2021.

[44] V. Kolesnikov, A. Sadeghi, and T. Schneider, "Improved garbled circuit building blocks and applications to auctions and computing minima," in *Proc. 8th Int. Conf. Cryptol. Netw. Secur.*, J. A. Garay, A. Miyaji, and A. Otsuka, Eds., Springer, 2009, pp. 1–20.

[45] Y. Huang, L. Malka, D. Evans, and J. Katz, "Efficient privacy-preserving biometric identification," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, San Diego, CA, USA, 2011, pp. 1–14.

[46] Y. Huang, D. Evans, J. Katz, and L. Malka, "Faster secure two-party computation using garbled circuits," in *Proc. 20th USENIX Secur. Symp.*, USENIX Assoc., San Francisco, CA, USA, 2011, Art. no. 35.

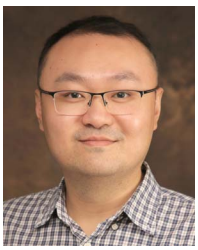
[47] M. Bellare, V. T. Hoang, and P. Rogaway, "Foundations of garbled circuits," in *Proc. ACM Conf. Comput. Commun. Secur.*, T. Yu, G. Danezis, and V. D. Gligor, Eds., Raleigh, NC, USA, 2012, pp. 784–796.

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

[49] Y. Luo, X. Jia, S. Fu, and M. Xu, "pRide: Privacy-preserving ride matching over road networks for online ride-hailing service," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 7, pp. 1791–1802, Jul. 2019.

[50] H. Yu, X. Jia, H. Zhang, X. Yu, and J. Shu, "PSRide: Privacy-preserving shared ride matching for online ride hailing systems," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 3, pp. 1425–1440, May/Jun. 2021.

[51] M. Naor and B. Pinkas, "Efficient oblivious transfer protocols," in *Proc. 12th Annu. Symp. Discrete Algorithms*, S. R. Kosaraju, Ed., Washington, DC, USA, 2001, pp. 448–457.
[52] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank, "Extending oblivious transfers efficiently," in *Proc. 23rd Annu. Int. Cryptol. Conf.*, D. Boneh, Ed., Springer, Santa Barbara, CA, USA, 2003, pp. 145–161.
[53] R. W. Floyd, "Algorithm 97: Shortest path," *Commun. ACM*, vol. 5, no. 6, pp. 344–348, 1962.
[54] A. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. Boca Raton, FL, USA: CRC Press, 1996.
[55] JPBC, *The Java Pairing Based Cryptography Library (JPBC)*, 2023. [Online]. Available: http://gas.dia.unisa.it/projects/jpbc
[56] A. D. Caro and V. Iovino, "JPBC: Java pairing based cryptography," in *Proc. 16th IEEE Symp. Comput. Commun.*, Kerkyra, Corfu, Greece, 2011, pp. 850–855.
[57] FastGC, *FastGC*, 2023. [Online]. Available: https://github.com/uvasrg/FastGC



Meng Li (Senior Member, IEEE) received the PhD degree in computer science and technology from the School of Computer Science and Technology, Beijing Institute of Technology (BIT), China, in 2019. He is an associate professor and personnel secretary with the School of Computer Science and Information Engineering, Hefei University of Technology (HFUT), China. He is also a post-doc researcher with the Department of Mathematics and HIT Center, University of Padua, Italy, where he is with the Security and Privacy Through Zeal (SPRITZ) Research Group led by Prof. Mauro Conti (IEEE Fellow). His research interests include security, privacy, applied cryptography, blockchain, TEE, and Internet of Vehicles. In this area, he has published 87 papers in topmost journals and conferences, including *IEEE Transactions on Information Forensics and Security*, *IEEE Transactions on Dependable and Secure Computing*, *IEEE/ACM Transactions on Networking*, *IEEE Transactions on Mobile Computing*, *IEEE Transactions on Knowledge and Data Engineering*, *ACM Transactions on Database Systems*, *IEEE Transactions on Services Computing*, *IEEE Communications Surveys and Tutorials*, ISSTA, and MobiCom. He is an associate editor of *IEEE Transactions on Information Forensics and Security*, *IEEE Transactions on Network and Service Management*, *IEEE Internet of Things Journal*, and *Computer Networks*. He is the recipient of 2024 IEEE HITC Award for Excellence (early career researcher).

Yifei Chen received the MS degree from the School of Computer Science and Information Engineering, Hefei University of Technology, in 2022. He is currently working toward the PhD degree with the School of Computer Science and Information Engineering, Hefei University of Technology. His research interests include security, privacy, applied cryptography, TEE, blockchain, and vehicular networks.

Jianbo Gao received the MS degree from the School of Computer Science and Information Engineering, Hefei University of Technology, in 2023. He is currently working toward the MS degree with the School of Cyberspace Science and Technology, Beijing Institute of Technology. His research interests include security, privacy, applied cryptography, and vehicular networks.

Jingyu Wu received the BS degree from the School of Computer Science and Information Engineering, Hefei University of Technology, in 2024. He is currently working toward the MS degree with the School of Computer Science and Technology, University of Science and Technology of China.

Zijian Zhang (Senior Member, IEEE) received the PhD degree from the School of Computer Science and Technology, Beijing Institute of Technology. He is now a professor with the School of Cyberspace Science and Technology, Beijing Institute of Technology. He was a visiting scholar with the Computer Science and Engineering Department, State University of New York at Buffalo in 2015. His research interests include design of authentication and key agreement protocol and analysis of entity behavior and preference.

Jialing He received the MS and PhD degrees from the Beijing Institute of Technology, Beijing, China, in 2018 and 2022, respectively, where she is currently an research assistant professor with the College of Computer Science, Chongqing University, Chongqing, China. Her current research interests include machining learning security, user behavior mining, and privacy preserving.

Liehuang Zhu (Senior Member, IEEE) received the MS degree in computer science from Wuhan University, Wuhan, China, in 2001, and the PhD degree in computer science from the Beijing Institute of Technology, Beijing, China, in 2004. He is a full professor with the School of Cyberspace Science and Technology, Beijing Institute of Technology, Beijing, China. His research interests include data security and privacy protection, blockchain applications, and AI security. He has authored more than 150 journal and conference papers in these areas. He is an associate editor of *IEEE Transactions on Vehicular Technology*, *IEEE Network*, and *IEEE Internet of Things Journal*. He was a guest editor of special issue of the *IEEE Wireless Communications* and *IEEE Transactions on Industrial Informatics*. He has served as program co-chair of MSN 2017, IWWS 2018, and INTRUST 2014. He received the Best Paper Award at IEEE/ACM IWQoS 2017, IEEE TrustCom 2018, and IEEE IPCCC 2014.

Mauro Conti (Fellow, IEEE) received the PhD degree from the Sapienza University of Rome, Italy, in 2009. He is a full professor with the University of Padua, Italy. He is also affiliated with TU Delft and University of Washington, Seattle. After his PhD, he was a post-doc researcher with Vrije Universiteit Amsterdam, The Netherlands. In 2011 he joined as an assistant professor with the University of Padua, where he became associate professor in 2015, and full professor in 2018. He has been a visiting researcher with GMU, UCLA, UCI, TU Darmstadt, UF, and FIU. He has been awarded with a Marie Curie Fellowship (2012) by the European Commission, and with a Fellowship by the German DAAD (2013). His research is also funded by companies, including Cisco, Intel, and Huawei. His main research interest include the area of security and privacy. In this area, he published more than 400 papers in topmost international peer-reviewed journals and conferences. He is the editor-in-chief of *IEEE Transactions on Information Forensics and Security*, area editor-in-chief of *IEEE Communications Surveys & Tutorials*, and has been an associate editor for several journals, including the *IEEE Communications Surveys & Tutorials*, *IEEE Transactions on Dependable and Secure Computing*, and *IEEE Transactions on Network and Service Management*. He was program chair for TRUST 2015, ICISS 2016, WiSec 2017, ACNS 2020, CANS 2021, and general chair for SecureComm 2012, SACMAT 2013, NSS 2021 and ACNS 2022. He is senior member of the ACM, and fellow of the Young Academy of Europe.

Xiaodong Lin (Fellow, IEEE) received the PhD degree in information engineering from the Beijing University of Posts and Telecommunications, China, and the PhD degree in electrical and computer engineering from the University of Waterloo, Canada. He is currently a professor with the School of Computer Science, University of Guelph, Canada. His research interests include computer and network security, applied cryptography, and computer forensics.