Decentralized and Privacy-Preserving Smart Parking With Secure Repetition and Full Verifiability

Meng Li, *Senior Member, IEEE*, Mingwei Zhang, Liehuang Zhu, *Senior Member, IEEE*, Zijian Zhang, *Senior Member, IEEE*, Mauro Conti, *Fellow, IEEE*, and Mamoun Alazab, *Senior Member, IEEE*

***Abstract*—Smart Parking Services (SPSs) enable cruising drivers to find the nearest parking lot with available spots, reducing the traveling time, gas, and traffic congestion. However, drivers risk the exposure of sensitive location data during parking query to an untrusted Smart Parking Service Provider (SPSP). Our motivation arises from *a repetitive query to an updated database, i.e., how a driver can be repetitively paired with a previously-matched-but-forgotten lot*. Meanwhile, we aim to achieve repetitive query in an oblivious and unlinkable manner. In this work, we present Mnemosyne^2: decentralized and privacy-preserving smart parking with secure repetition and full verifiability. Specifically, we design repetitive, oblivious, and unlinkable Secure k Nearest Neighbor (Sk NN) with basic verifiability (correctness and completeness) for encrypted-and-updated databases. We build a local Ethereum blockchain to perform driver-lot matching via smart contracts. To adapt to the lot count update, we resort to the immutable blockchain for advanced verifiability (truthfulness). Last, we utilize decentralized blacklistable anonymous credentials to guarantee identity privacy. Finally, we formally define and prove privacy and security. We conduct extensive experiments over a real-world dataset and compare Mnemosyne^2 with existing work. The results show that a query only needs 8 seconds (175 ms) on average for service waiting (verification) among 500 drivers.**

***Index Terms*—Smart Parking, repetitive query, privacy, security, SkNN, blockchain.**

Manuscript received 22 June 2023; revised 19 January 2024; accepted 3 May 2024. Date of publication 7 May 2024; date of current version 5 November 2024. This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant U23A20303 and Grant 62372149, in part by the Beijing Advanced Innovation Center for Future Blockchain and Privacy Computing, and in part by the EU LOCARD Project under Grant H2020-SU-SEC-2018-832735. The work of Mamoun Alazab was supported in part by the Ministry of Education of the Republic of Korea and in part by the National Research Foundation of Korea under Grant NRF-2021S1A5A2A03064391. Recommended for acceptance by O. Yagan. *(Corresponding author: Zijian Zhang.)*

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

Liehuang Zhu is with the School of Cyberspace Science and Technology, Beijing Institute of Technology, Beijing 100081, China (e-mail: liehuangz@bit.edu.cn).

Zijian Zhang is with the School of Cyberspace Science and Technology, Beijing Institute of Technology, Beijing 100081, China, and also with the Beijing Advanced Innovation Center for Future Blockchain and Privacy Computing, Beijing 100081, China (e-mail: zhangzijian@bit.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 Delft, The Netherlands (e-mail: mauro.conti@math.unipd.it).

Mamoun Alazab is with the College of Engineering, IT and Environment, Charles Darwin University, Casuarina, NT 0810, Australia (e-mail: alazab.m@ieee.org).

Digital Object Identifier 10.1109/TMC.2024.3397687

I. Introduction

THE surge of vehicles and the lack of effective navigation have lead to a parking headache in modern cities. It is reported that there are 289.5 million registered cars in the US in 2021 [1] and drivers from NYC, LA, and San Francisco spent up to 107 Hours a year looking for parking [2]. It is estimated that the global smart parking market will reach US$11.2 Billion by 2027 [3]. Driven by the advancement of cloud computing and smart devices, Smart Parking Services (SPSs) are proposed to alleviate the parking problem. SPSs guide cruising drivers to a nearby parking lot with the help of a Smart Parking Service Provider (SPSP) or Road-Side Units [4], [5], [6], [7]. Recently, Decentralized Smart Parking Services (DSPSs) attract plenty of attention from both academia [8], [9] and industrial community [10] for eliminating the deficiency of centralized solutions, i.e., single-point-of-failure and non-transparency. They have revealed a great potential to fundamentally alter transportation systems by reducing cruising time, saving gas/electricity cost, and easing traffic congestion.

While SPSs offer appealing advantages, drivers still encounter various privacy concerns [11], [12]. This is because drivers have to upload sensitive current locations to request a parking lot from an untrusted SPSP. Such location information is highly related to user activities, which can be analyzed to track and even profile drivers [13]. The privacy concerns have stimulated the emergence of Privacy-Preserving Smart Parking (PPSP) [4], [5], [6], [7], [8], [9] that focuses on privacy issues.

We observe that *there is a new parking requirement, i.e., repetitive parking. In particular, a driver may request a previously matched parking lot that she/he forgets its exact location.* We validate it by using an online ballot (see Section VIII). For example, as shown in Fig. 1, a driver requested a parking spot on day 1 and is matched with a parking lot. On day k, the driver in the same neighborhood needs to find the previously matched parking lot. However, the driver forgets the exact location and has erased the parking history locally for privacy concerns. Inspired by this new requirement, *we are motivated to achieve secure repetition*, i.e., drivers can request a previously-matched-but-forgotten lot in an oblivious and unlinkable manner. Informally, it prevents

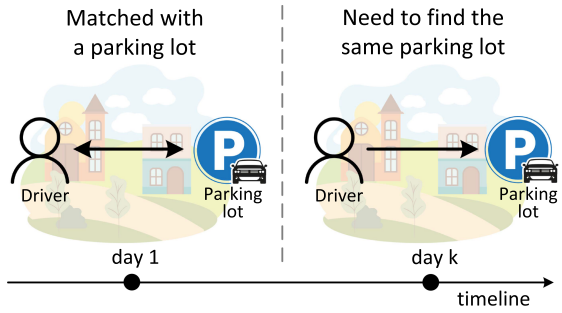


Fig. 1. An example of repetitive parking.

the SPSP from perceiving a driver's demand for repetitive query and linking the current request to her/his previous requests. At first glance, requesting the same parking lot more than once does not violate user privacy. However, it informs the SPSP of the driver's visit to the same neighbor, which leaks the driver's moving pattern and frequently visited locations. Therefore, the potential privacy leakage necessitates secure repetition.

Existing work on PPSP [4], [5], [6], [7], [8], [9] does not support repetitive parking, let alone secure repetition. Secure k Nearest Neighbor (S k NN) [14], [15] naturally fits in PPSP since it is a secure location-based service that performs user-item matching and protects location privacy. However, existing work does not consider searching on an updated index that may not return the previous data item (parking lot) and disable repetitive query (parking). For example, say a driver Bob drove to the Hair Salon on 64th Street in New York and he used a smart parking app to park in a parking lot "iPark" on the same street. A few days later, Bob drives to the club again but forgets where the iPark is. A problem arises when the SPSP has a new encrypted index, which is updated by the parking lot owner, and the previous iPark was ordered at the bottom of the recommendation list. If Bob submits a S2NN query, he will probably receive iPark on E 65th Street and iPark on E 67th Street, which is not what he expects. Moreover, most existing work on S k NN and PPSP only considers a semi-honest model for the service provider [4], [5], [6], [7], [8], [9], [14], [15]. In this study, we consider a stronger security model, where the SPSP is malicious and has a secret agreement with a Parking Lot Manager (PLM). Simply put, they can launch *an unfair ordering attack, a mismatching attack, an ignoring attack, and a miscounting attack*, which necessitates full verifiability. The drivers should be allowed to verify the correctness, completeness, and truthfulness of the query results. Therefore, smart parking with secure repetition and full verifiability exhibits a different structure that cannot be handled using existing techniques, which calls for new solutions. To achieve this goal, we need to address **two fundamental problems**:

- How to request a previously-matched data item from the SPSP with an updated secure index under a S k NN framework?
- How to guarantee verifiability of the query results when the SPSP may tamper with the matching process?

A simple way of finding a previously-matched parking lot in an oblivious and unlinkable manner is to query all the parking lots near the driver's current location. Apparently, this causes huge computational costs in query processing and large communication overhead of returned results. Assume that the driver needs the parking lot pl with a sequence number i. Intuitively, there are three approaches to finding pl_i in SPSP's index: 1) Find pl_i and continue to find the other $k-1$ data items. 2) End at pl_i and return the obtained data items. 3) Randomly choose r data items before pl_i and find $k-r-1$ data items after pl_i. The three approaches require special treatment on locating pl_i which makes it difficult for the SPSP not to notice this difference. This first one may traverse the whole index and the second one may return all the matched data items if pl_i is in the end. The last one is faced with an uncertain choice of r. To enable repetitive parking, we can introduce an identity to each parking lot. The data owner appends the identity to the location for each parking lot. When processing a repetitive parking query, the SPSP uses the identity as the extra query condition. While matching the previous parking lot precisely regardless of the index update, this approach, however, excludes other location-matching parking lots which may expose the driver's intention. To match other parking lots, we cannot use the identity directly. Therefore, **challenge I** lies in the contradiction between locating a preferred parking lot and matching other parking lots. In current S k NN schemes [14], [15], there is a count update problem that could potentially be the Achilles' heel of S k NN. As it is well-known that a place of business (park lot) frequently updates its service availability (count of available parking spots), thereby, updating the count of each data item is a normal operation for the SPSP. However, the index structures of current S k NN techniques can only lay a foundation for basic verifiability (correctness and completeness), but cannot control the data item count that varies during query matching (advanced verifiability). Therefore, **challenge II** lies in the contradiction between data owner's losing control over counts and SPSP's proving the truthfulness of counts.

To cope with the above challenges, *we propose a decentralized and privacy-preserving scheme named Mnemosyne*[2] *to achieve smart parking with secure repetition and full verifiability.* Specifically, we first divide the service map into a l-leveled pyramid and each level consists of a number of grids. For each level, we design an efficient space encoding technique to process locations. The parking lot manager encodes the parking lots' locations and obtains a set of leveled location codes. At the l-th level, we assign an identity to each parking lot. The parking lot manager computes a prefix family of the parking lot identity and integrates it with location codes at the l-th level. Next, the PLM inserts the integrated codes into an Indistinguishable Bloom Filter (IBF) [17] as a secure index. A driver user who is about to submit a repetitive parking query generates a customized identity range to compute a minimum set of prefixes while guaranteeing the range includes the identity of the previous parking lot and the length of the minimum set of prefixes is equal to the one of a normal query (non-repetitive query). Next, the driver integrates the prefixes with the location codes and computes a query token, hiding the intention of repetitive parking. By doing so, the SPSP can search the secure index by querying the token on it without violating the secure repetition, solving the first challenge.

TABLE I
COMPARISON AMONG PREVIOUS PPSP SCHEMES AND MNEMOSYNE^2

Scheme	Decentralized Model	Security Model	Identity Privacy	Location Privacy	Secure Repetition	Full Verifiability			Fine-Grained Blacklist
						Correctness	Completeness	Truthfulness	
P-SPAN [4]		Semi-honest	√						
SAVP [5]		Semi-honest	√	√					
PrivAV [6]		Semi-honest	√	√					
ASAP [7]		Semi-honest	√	Partial[1]					
PEPS [8]	√	Semi-honest	√	Partial[1]					
PriParkRec [9]	√	Semi-honest	√	Partial[1]					
Mnemosyne^2	√	Malicious	√	√	√	√	√	√	√

1: Location privacy is partially protected via a cloaking technique.

To bridge the gap between the loss of count control and the autarchy of count update, we build a consortium blockchain [18], [19] among several collaborating SPSPs to offer transparency by recording parking/departing queries, and responding to parking queries via smart contracts. In this way, the SPSPs have to prove the truthfulness of available spots by providing periodic proofs from the auditable blockchain, solving the second challenge.

Last, we utilize the Decentralized Blacklistable Anonymous Credentials (DBAC) [20] to adapt to the decentralized network while guaranteeing identity privacy.

We list the comparison among previous PPSP schemes and Mnemosyne^2 in Table I, and state our contributions as follows.

- To the best of our knowledge, we are the first to focus on the repetitive parking in SPSs. We propose a privacy-preserving smart parking system SPS with secure repetition based on a carefully crafted S k NN technique.
- We focus on the count update problem of existing S k NN and design a blockchain-based approach to provide full verifiability of query results.
- We integrate smart parking with S k NN by locating four security attacks that apply to both of them.
- We formally state and define privacy and security. We implement a prototype based on a server and conduct experiments over a real-world dataset. We conduct a thorough comparison with existing work regarding system model, security, privacy, and performance.

The remaining of this paper is organized as follows. We discuss related work in Section II. We elaborate on the system model, threat model, and design objectives in Section III. We revisit some preliminaries in Section IV. In Section V, we present the proposed Mnemosyne^2. We formally analyze the security and privacy in Section VI. We implement Mnemosyne^2 and analyze its performance in Section VII. Lastly, we give discussions in Section VIII and conclude in Section IX.

II. RELATED WORK

In this section, we review some related work and summarize how Mnemosyne^2 advances the state of the art.

A. PPSP

Ni et al. [4] proposed a privacy-preserving smart parking navigation scheme P-SPAN with efficient navigation result retrieval. They utilize the short randomizable signatures [21] to provide anonymous authentication. Each driver encrypts the basic query information, e.g., pickup location and destination, and sends the ciphertext to an SPSP via the relay of the nearby RSUs. The SPSP authenticates and decrypts the query and looks for an available parking lot. The SPSP encrypts a navigation result and returns it to the driver by using a counting Bloom filter [22].

Huang et al. [5] proposed a privacy-preserving reservation scheme for automated valet parking. They protect identity privacy and prevent the claimed "double-reservation attack" based on zero-knowledge proofs of knowledge and proxy re-signature [23]. To enhance location privacy, they adopt the Geo-indistinguishability mechanism [24] to protect from the location-based statistical analysis attack.

Ni et al. [6] proposed a secure and privacy-preserving automated valet parking scheme for self-driving vehicles. They extend anonymous authentication to support two-factor authentication with mutual traceability based on one-time password and secure mobile devices. They complete the querying process based on a cuckoo filter supporting item adding and removing [25].

Zhu et al. [7] proposed to leverage private parking spots to ease the public parking problem and presented a privacy-preserving smart-parking scheme supporting anonymous payment. The short randomizable signatures are used to provide identity privacy in a conditional way. Each driver sends a cloaked location to the SPSP which constructs a hashmap for quick matching. The anonymous payment process is achieved by using E-cash based on blond signatures [26].

Wang et al. [8] presented a privacy-preserving parking spot sharing scheme without trusted third parties. The use decentralized anonymous credentials [27] for identity privacy in a consortium blockchain maintained by fog servers. The parking spot queries and reports are encrypted via ElGamal encryption [28]. A variant Monero is used to guarantee the anonymity and confidentiality of payments.

Li et al. [9] presented a privacy-preserving parking-space recommendation scheme. A driver with a decentralized anonymous credential anonymously reveals to the SPSP (enabled by the blockchain) a set of necessary attributes to find a parking slot. The blockchain forwards the request to a parking space provider, which responds to the request and updates the status and decisions to the blockchain.

B. S k NN

Li et al. [44] presented the first range query processing protocol, which achieved index indistinguishability under the indistinguishability against chosen keyword attack (IND-CKA).

A data owner converts each data item dt_i by prefix encoding [29] and organizes each prefix family of encoded item $F(di_i)$ into a PBTree. Then the data owner makes the PBtree privacy-preserving by a keyed hash message authentication code HMAC and Bloom filters [30]. For each prefix pr_i, the data owner computes several hashes $\mathsf{HMAC}(K_j, pr_i)$, and inserts a randomized version $\mathsf{HMAC}(r, \mathsf{HMAC}(K_j, pr_i))$ into a Bloom filter. Each r corresponds to a node and each node relates to a prefix family, i.e., data item. Next, a data user converts a range into a minimum set of prefixes and computes several hashes $\mathsf{HMAC}(K_j, pr_i)$ for each pr_i as a trapdoor. The service provider searches the PBtree by using the trapdoor.

Lei et al. [15] presented a secure and efficient query processing protocol SecEQP. They leveraged several primitive projection functions to convert the neighbor regions of a given location. Given the codes of two converted locations, the service provider computes the proximity of the two locations by judging whether the two codes are the same. The two-dimensional location data is projected to high-dimensional data to expand the location space and make the converted location more secure. The data owner further embeds the codes into an IBF [17] to build a secure index. In an IBF, a pseudo-random hash function h determines a cell location $H(h_{k+1}(h_j(w_i)) \oplus r)$, i.e., which twin cell stores '1'. The data user computes similar trapdoors by using a keyed hash message authentication code. The query processing is conducted by searching the IBF tree to find a match by using the trapdoor.

Cui et al. [16] proposed a secure and verifiable k nearest neighbor query processing protocol SVkNN. They adopted Voronoi diagram to divide the whole area. This space encoding method is a replaceable unit that is not directly related to their core design. They design a new data structure, i.e., Verifiable and Secure Index (VSI) to support fast query and verification. Next, they propose several secure protocols and a compact verification method to facilitate the operation over VSI to support performing the search over the secure index. However, SVkNN uses two cloud servers that require extra communication and computation.

Li et al. [31] proposed a repetitive, oblivious, and unlinkable S k NN scheme ROU for location-based services. It designs a multi-level structure to process locations and integrates a data item identity into the framework of S k NN. Data owners and data users can realize secure query processing via encrypted indexes and encrypted tokens from a customized identity range. It is also the foundation of this work.

The promotion over existing work is that Mnemosyne ^{2} supports secure repetitive querying for Sk NN with an updated index and secure repetitive parking for SPSs, supports truthfulness of results for S k NN and full verifiability (correctness, completeness, and truthfulness) of query results for SPSs, resists unfair ordering attack, mismatching attack, ignoring attack, and miscounting attack from the malicious SPSP. Specifically, we extend our previous work (ICICS'22) [31] from the following aspects: We observe the problem of repetitive query in smart parking services (SPSs) and emphasize how it violates drivers' location privacy. We formally define repetitive querying in SPSs. Following this problem, we propose the requirement of secure repetition in RHS. We introduce and formally define four new attacks, i.e., unfair ordering attack, mismatching attack, ignoring attack, miscounting attack from the Smart Parking Service Provider (SPSP) and drivers. Under these attacks, the matching fairness of is easily violated. We formally define privacy (identity privacy and location privacy), secure repetition (obliviousness and unlinkability) and full verifiability (correctness, completeness, and truthfulness) in SPSs. We propose a new ride matching scheme Mnemosyne ^{2} based on a carefully crafted Sk NN and blockchain to achieve the privacy and security goals. We update the related work with the state-of-the-art. We conduct a thorough comparison with P-SPAN (2018), SAVP (2018), PrivAV (2018), ASAP (2020), PEPS (2020), PriParkRec (2021) regarding system model, computational costs, and communication overhead. We include a new section (Section III Problem Statement). This section gives the system model, security model, and design objectives of our smart parking scheme. We formally analyze the security and privacy of our proposed scheme. We instantiate Mnemosyne^2 using a PC server as crypto server and an Android smartphone as a driver, two Android virtual machines as testing devices. We conduct extensive experiments over a real-world dataset to evaluate Mnemosyne^2's performance, including computational cost, communication overhead, and scalability. According to the previous points, Title, Abstract, Introduction, and Conclusions have been completely revised. In general, we re-structured the paper in a more reader-friendly manner.

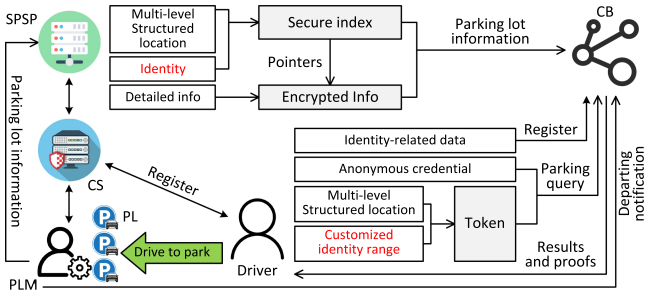


Fig. 2. System model and information flows of mnemosyne^2.

III. Problem Statement

In this section, we formalize the system model (Section III-A) and the formal security model (Section III-B), and summarize the formal design objectives (Section III-C). We formally define the four new attacks mentioned in Section I.

A. System Model

We portray the system model in Fig. 2. We list the key notations in Table II. The system model of Mnemosyne^2 consists of five types of entities: driver, parking lot (PL), parking lot manager (PLM), smart parking service provider (SPSP), Crypto Server (CS), and consortium blockchain (CB).

Driver is a user who is equipped with an on-board unit or a smartphone and searching a nearby parking lot near her/his current location cl. The driver registers to an SPSP, the CB, and the CS. The driver downloads the latest requirement of an SPSP from the CB, sends a parking query to an SPSP, and awaits a

TABLE II
KEY NOTATIONS OF MNEMOSYNE[2]

Notation	Definition
SPS, DSPS	Smart parking service, decentralized SPS
SPSP, PPSP	SPS provider, privacy-preserving smart parking
PL, PLM	Parking lot, PL manager
CS, CB	Crypto server, consortium blockchain
SkNN	Secure k Nearest Neighbor
DBAC	Decentralized blacklistable anonymous credentials
pl, ps	Parking lot, number of parking spots
cl, pq, $\mathcal{R}$	Current location, parking query, query result
$\mathrm{Tx}^{\mathrm{Reg}}$, $\mathrm{Tx}^{\mathrm{Que}}$	Parking registration/querying transaction
$\mathrm{Tx}^{\mathrm{Sta}}$, $\mathrm{Tx}^{\mathrm{Dep}}$	Parking status transaction, departing transaction
DLMSC	Driver-lot matching smart contract
$\mathrm{Att}^{\mathrm{UO}}$, $\mathrm{Att}^{\mathrm{MM}}$	Unfair ordering attack, mismatching attack
$\mathrm{Att}^{\mathrm{IG}}$, $\mathrm{Att}^{\mathrm{MC}}$	Ignoring attack, miscounting attack
ind, qt, k	Secure index, query token, query parameter
IBF, sk	Indistinguishable Bloom filter, secret key
m, t	Number of cell twins, number of h
h, H	Pseudo-random hash function, hash function
SPK	Signature of knowledge
n_1, n_2, n_3, n_4	Number of parking lots/drivers/PLMs/SPSPs
gn, lc, mc	Grid number, location code, mixed code
ct, HV, RH, $\mathcal{TR}$	Ciphertext, hash value, root hash value, index tree
$\mathcal{S}$, $\mathcal{M}_1$, $\mathcal{M}_2$; $\mathcal{TK}$	Minimum set of prefixes; token
$\mathcal{A}$, $\mathcal{A}'$	Adversary

query result from the CB. The driver decrypts the information of a set of parking lot and chooses one to establish a secure channel to communicate and park. When the parking is complete, the driver sends a feedback to the CB. Only registered drivers can enjoy the SPS.

PL is a parking lot with multiple parking spots. It monitors the status of each parking spot, waits for drivers to park, and charges a parking fee for the drivers through IoT devices (e.g. sensors and cameras).

PLM owns at least one parking lot pl. It monitors each PL and uploads their real-time status (e.g., parking fee standard and number of available parking spots) to an SPSP.

SPSP is a cloud server that collects parking lot information from several PLMs. It provides on-demand parking services to cruising drivers by uploading a parking status transaction $\mathrm{Tx}^{\mathrm{Sta}}$ with parking lots, proofs, and requirements to the CB. It receives and verifies parking queries from drivers. The SPSP accepts a driver's query if and only if the driver meets its requirement. An SPSP collects data from the CB automatically and puts its requirement, including a candidate driver set and a blacklist, to the CB regularly. We assume that there are more than one SPSP and they all associate with at least one PLM.

CB is a permissioned blockchain co-maintained by SPSPs that processes all the parking transactions. It is accessible to all participants, e.g., drivers and SPSPs. It receives registration queries from drivers, parking status from SPSPs, and responds to drivers' parking queries by using a driver-lot matching smart contract (DLMSC).

CS is an entity that initializes the Mnemosyne [2] system by generating public parameters and cryptographic keys for the drivers and SPSPs. For the public parameters of the DBAC, the CS is only used in the system initialization phase. For the parameters of the Sk NN, the CS is only used during registration. The CS can be the Department of Transportation in a real-world implementation. It divides the parking service area into a multi-level structure with SPSPs.

Definition 1 (Repetitive Query): Given a driver D_j and his/her parking query Q, we there is a repetitive query Q' from D_j when the requested PL identity is the same as the query result of Q.

B. Security Model

The blockchain is trusted for correctness and availability while not for user privacy [32]. We assume that the CS generates the public parameters honestly. We do not consider physical attack, such as tracking a driver by taking photos. The threats mainly come from internal adversaries [19], [33], [34], [35], [36], [37] and we adopt the semi-honest security model for the drivers. The drivers are assumed to be semi-honest. We do not exclude the possibility that a driver to repudiate the parking fee. Such a misbehavior can be detected at the parking lot exit by conditionally triggered video surveillance or camera, which generates a proof of flee for accountability. Moreover, we can ask the drivers to put down a deposit before parking, which in turn calls for studying its feasibility. The security assumption of PLM and PL is semi-honest. They do not collude with the SPSP since they care about their reputation in the long run and it is possible for the SPSP to know which PL the current driver is heading toward under such a collusion attack.

Especially, we consider unfair ordering attack, mismatching attack, ignoring attack, miscounting attack from the SPSP. • The unfair ordering attack sorts a parking lot in a specific location when building a secure index over a set of parking spots, which is similar to unfair ranking where some search engines treat websites unfairly [38].

- The mismatching attack recommends to a driver a parking lot that does not match the current location of the driver.
- The ignoring attack does not recommend to a driver a parking lot that matches the current location of the driver when the number of matched parking lots is less than k.
- The miscounting attack claims that a location-matching parking lot does not have enough parking spots. The miscounting attack is from the malicious SPSP since it receives all parking lot information from the PLMs and can easily misclaim the number of parking spots. The PLM is considered as the owner of several PLs that does not have enough incentive to miscount when maintaining its good reputation.

Now we give their formal definitions.

Definition 2 (Unfair Ordering Attack Att^{UO}): Given a set of parking lots $\mathcal{PL} = \{pl_1, \ldots, pl_{n_1}\}$, we define a normal ordering function, i.e., a pseudo-random permutation $F_1 : \mathcal{PL}^{n_1} \rightarrow \pi(\mathcal{PL}^{n_1})$, which outputs a sequences of parking lots drawn from $\mathcal{PL}$ after a pseudo-random permutation π. Given $\mathcal{PL}$, a parking lot pl_i, and a location j where pl_i and j are chosen by a malicious SPSP $\mathcal{A}$, the unfair ordering attack $\mathrm{Att}^{\mathrm{UO}}$ is defined as a function

$$F_1^{\mathrm{UO}} : \mathcal{PL}^{n_1} \times [n_1] \times [n_1] \rightarrow \{\ldots, \underbrace{pl_i}_{j\text{th}}, \ldots\}.$$

$\mathcal{PL}^{n_1}$ denotes to a sequence of n_1 distinctive parking plots and Att^{UO} can be extended to put any pl at any intended location.

Definition 3 (Mismatching Attack Att^{MM}): Given a set of secure indexes $\mathcal{IND} = \{ind_1, ind_2, \ldots, ind_{n_1}\}$ and a query token qt including a query parameter k, we define a normal matching function $F_2 : \mathcal{IND}^{n_1} \times \{qt\} \rightarrow \{ind_i^{qt} | \mathsf{Match}(ind_i^{qt}, qt) = 1, 1 \leq i \leq k, ind_i^{qt} \in \mathcal{IND}\}$, which outputs the first k secure indexes that matches qt. $\mathsf{Match}(\cdot)$ is a membership checking function that checks whether a qt matches an ind. Given $\mathcal{IND}$, a secure index ind_j chosen by $\mathcal{A}$, the mismatching attack Att^{MM} is defined as a function

$$F_2^{\text{MM}} : \mathcal{IND}^{n_1} \times \{qt\} \times [n_1] \rightarrow \{ind_i^{qt} | \mathsf{Match}(ind_i^{qt}, qt) = 1, 0 \leq i \leq j-1, j \leq k, \mathsf{Match}(ind_j^{qt}, qt) = 0\}.$$

Definition 4 (Ignoring Attack Att^{IG}): Given $\mathcal{IND}$, a secure index ind_j chosen by $\mathcal{A}$, the ignoring attack Att^{IG} is defined as a function

$$F_3^{\text{IG}} : \mathcal{IND}^{n_1} \times \{qt\} \times [n_1] \rightarrow \{ind_i^{qt} | \mathsf{Match}(ind_i^{qt}, qt) = 1, 0 \leq i \leq k, ind_i^{qt} \in \mathcal{IND} \setminus \{ind_j\}, ind_j \in F_2(\mathcal{IND}, qt)\}.$$

Definition 5 (Miscounting Attack Att^{MC}): A secure index ind_i corresponds to a parking lot pl_i with a number of available parking spot ps_i, we define a normal counting function $F_4 : \{ind\} \rightarrow \{0, 1, \ldots, cnt\}$, which outputs the number of available parking spots. Given an ind_i chosen by $\mathcal{A}$, the miscounting attack Att^{MC} is defined as a function

$$F_4^{\text{MC}} : \{ind_i\} \rightarrow \{0, 1, \ldots, ps_i - 1, ps_i + 1, \ldots, cnt\}.$$

We separated privacy from security to achieve a reasonable level of rigor and clarity. The privacy is about location, identity, and secure repetition (obliviousness, unlinkability), while security only refers to defending three attacks.

C. Design Objectives

1) Privacy: (1.1) Location privacy. The current location of drivers should be protected from the SPSP. The location privacy refers to index privacy and token privacy. (1.2) Identity Privacy. The identity of drivers should be protected from the SPSP and PLM. We give its formal definition as follows. A Probabilistic Polynomial-Time (PPT) adversary $\mathcal{A}$ attacking the smart parking scheme Π is SPSP. Now we design an experiment $\mathsf{PrivK}^{\mathsf{ind}}_{\mathcal{A},\Pi}(\lambda)$ based on $\mathcal{A}$ and Π as follows.

Identity indistinguishability experiment $PrivK^{ind}_{\mathcal{A},\Pi}(\lambda)$:

1. Setup: a pair of public key and private key (sk, pk) is generated. Tickets $\{tk_i\}$ of sk are computed by $\mathsf{TicketGen}(sk)$ [20]. $\mathcal{A}$ can access an oracle $\mathcal{O}$.
2. Oracle: $\mathcal{A}$ makes a number of queries to $\mathcal{O}$ and receives newly computed tickets from $\mathsf{TicketGen}(sk)$.
3. Challenge: $\mathcal{C}$ chooses a uniform bit $b \in \{0, 1\}$. If $b = 0$, $\mathcal{C}$ returns $\{tk_i\}$ to $\mathcal{A}$; else, returns tickets sampled uniformly at random from the range of the $\mathsf{TicketGen}(\cdot)$.
4. Guess: $\mathcal{A}$ outputs $b' \in \{0, 1\}$.
5. The output of the experiment is defined to be 1 if $b' = b$, and 0 otherwise. We write $\mathsf{PrivK}^{\mathsf{ind}}_{\mathcal{A},\Pi}(\lambda) = 1$ if the output of the experiment is 1, i.e., $\mathcal{A}$ succeeds.

Definition 6 (Identity Privacy): Mnemosyne^2 achieves identity privacy if $\Pr[\mathsf{PrivK}^{\mathsf{idn}}_{\mathcal{A},\Pi}(\lambda) = 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$, where the probability is taken over all randomness used in $\mathsf{PrivK}^{\mathsf{idn}}_{\mathcal{A},\Pi}(\lambda)$, idn stands for identity, and $\mathsf{negl}()$ is a negligible function.

2) Secure Repetition:

(2.1) Obliviousness. $\mathcal{A}$ cannot know whether any driver requests a previous parking lot, i.e., cannot distinguish a normal query from a normal query (non-repetitive query). We design an experiment $\mathsf{PrivK}^{\mathsf{obl}}_{\mathcal{A},\Pi}(\lambda)$ as follows.

Obliviousness experiment $\mathsf{PrivK}^{\mathsf{obl}}_{\mathcal{A},\Pi}(\lambda)$:

1. Setup: n_1 parking lots and n_2 drivers are generated. Secret keys are generated by using Setup.
2. Execution: Π is executed with n_1 parking lots and n_2 drivers. $\mathcal{A}$ observes the index, token, and matching.
3. Prepare: $\mathcal{A}$ generates a pair of queries $q_0 = (cl_0, pl'_0)$ and $q_1 = (cl_1, pl'_1)$ satisfying $cl_0 = cl_1, pl'_0 \in \{pl_1, pl_{n_1}\}$, $pl'_1 = 0$, and $|\mathsf{S}(\mathsf{Range}(pl'_0))| = |\mathsf{S}(pl_1, pl_{n_1})|$. $\mathcal{A}$ sends (q_0, q_1) to $\mathcal{C}$. $\mathsf{Range}()$ is a designed function to hide the PL's identity, which we will define in Section V-D.
4. Challenge: A uniform bit $b \in \{0, 1\}$ is chosen and a challenge query token $\mathcal{TK}_b \leftarrow \mathsf{Token}(q_b)$ is computed and given to $\mathcal{A}$.
5. Guess: $\mathcal{A}$ outputs a bit b'.
6. The output of the experiment is defined to be 1 if $b' = b$, and 0 otherwise. We write $\mathsf{PrivK}^{\mathsf{ind}}_{\mathcal{A},\Pi}(\lambda) = 1$ if the output of the experiment is 1, i.e., $\mathcal{A}$ succeeds.

Definition 7 (Obliviousness): Mnemosyne^2 achieves obliviousness if $\Pr[\mathsf{PrivK}^{\mathsf{obl}}_{\mathcal{A},\Pi}(\lambda) = 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$, where the probability is taken over all randomness used in the experiment.

(2.2) Unlinkablity. $\mathcal{A}$ cannot link any driver's request to her/his previous requests. We design an experiment $\mathsf{PrivK}^{\mathsf{unl}}_{\mathcal{A},\Pi}(\lambda)$ as follows.

Unlinkability experiment $\mathsf{PrivK}^{\mathsf{unl}}_{\mathcal{A},\Pi}(\lambda)$:

1. Setup: n_1 parking lots and n_2 drivers are generated. Secret keys are generated by using Setup.
2. Execution: Π is executed with n_1 parking lots and n_2 drivers. $\mathcal{A}$ observes the index, token, and matching.
3. Challenge: A uniform bit $i \in [1, n_2]$ is chosen and a new query token q_{i1} is computed and given to $\mathcal{A}$. We require that previous query q_{i0} and q_{i1} satisfy $\mathsf{Exp}(cl_{i0}) \neq \mathsf{Exp}(cl_{i1}) \vee id_{i0} \neq id_{i1}$. $\mathsf{Exp}()$ is a function that expands the input area to a bigger area.
4. Guess: $\mathcal{A}$ outputs a bit i'.
5. The output of the experiment is defined to be 1 if $i' = i$, and 0 otherwise. We write $\mathsf{PrivK}^{\mathsf{unl}}_{\mathcal{A},\Pi}(\lambda) = 1$ if the output of the experiment is 1, i.e., $\mathcal{A}$ succeeds.

Definition 8 (Unlinkablity): Mnemosyne^2 achieves unlinkablity if $\Pr[\mathsf{PrivK}^{\mathsf{unl}}_{\mathcal{A},\Pi}(\lambda) = 1] \leq \frac{1}{n_2} + \mathsf{negl}(\lambda)$, where the probability is taken over all randomness used in the experiment.

3) Full Verifiability:

(3.1) Correctness. Given a park query pq, the SPSP returns the result set $\mathcal{R} = \{R_1, R_2, \ldots, R_u\}$. If for each result R_i $(i \leq u)$, the decrypted parking lot pl_i of R_i matches the location of pq, then $\mathcal{R}$ is correct.

(3.2) Completeness. Given a parking query pq, the SPSP returns the result set $\mathcal{R} = \{R_1, R_2, \ldots, R_u\}$. If each of the first k records $pl_i \in \mathcal{PL}$ such that pl_i matches Q, the encrypted data R_i must in $\mathcal{R}$, then $\mathcal{R}$ is complete.

(3.3) Truthfulness. Given a parking query pq, the SPSP returns the result set $\mathcal{R} = \{R_1, \ldots, R_u\}$. If each R_i $(i \leq u)$ has enough available parking spots, then $\mathcal{R}$ is truthful.

Efficiency: The proposed scheme Mnemosyne^2 should be efficient, i.e., computational costs and communication overhead of Mnemosyne^2 is acceptable. Especially, it has good scalability regarding number of drivers and parking spots.

IV. Preliminaries

In this section, we revisit some preliminaries, namely IBF (Section IV-A), two cryptographic assumptions (Section IV-B), zero-knowledge proof of knowledge (Section IV-C), and blockchain and smart contract (Section IV-D).

A. IBF

An Indistinguishable Bloom Filter (IBF) contains an array IBF of m cell twins, t pseudo-random hash functions $h_1, h_2, \ldots, h_{t+1}$, and a hash function H. Each cell twin has two cells and each cell stores either '0' or '1'. In the beginning, the all cells are set to '0'. An item tm is hashed to t twin cells $IBF[h_1(tm)], IBF[h_2(tm)], \ldots, IBF[h_t(tm)]$. h_{t+1} and a random number r determine which cell stores '1': $IBF_i[h_i(tm)][H(h_{t+1}(h_i(tm)) \oplus r)] = 1, 1 \leq i \leq t$.

B. Two Cryptographic Assumptions

Link Decisional (LD)-RSA assumption [39]. Let $N = (2p + 1)(2q + 1)$ be a product of two large safe primes and g is a generator of QR_N. Let p_0, q_0, p_1, q_1 be four sufficiently large and distinct primes, and $n_0 = p_0 q_0$, $n_1 = p_1 q_1$. We have $(N; g; n_0; n_1; g^{p_0+q_0}) \overset{c}{\approx} (N; g; n_0; n_1; g^{p_1+q_1})$ where $A \overset{c}{\approx} B$ to denote that A and B are computationally indistinguishable

Decisional Diffie-Hellman (DDH)-II assumption [40]. The original DDH-II assumption works in a prime order sub-group of a group $\mathbb{Z}_p^*$ for prime p, it can be extended to the quadratic residue group QR_N, where N is the product of two safe primes. Let g be a generator of QR_N, then for any distribution X with a super-logarithmic min-entropy, we have $(g, g^x, g^y, g^{xy}) \approx (g, g^x, g^y, g^z)$, where $x \leftarrow \mathcal{X}$ and $y, z \overset{\$}{\leftarrow} \mathbb{Z}_N$.

C. Zero-Knowledge Proof of Knowledge

In a zero-knowledge proof of knowledge (ZKPK) protocol [41], a prover proves a statement to a verifier without revealing anything about the statement other than that it is true. Normally, the prover P and the verifier V have to conduct several interactions for the proving process to complete. But this process can be converted into non-interactive proofs by applying the Fiat-Shamir heuristic [42]. For instance, we denote NIZKPK$\{(a) : A = g^a\}$ as a non-interactive zero-knowledge proof of knowledge of the value a satisfying $A = g^a$. The values in () are the knowledge that P needs to prove, and the other values are known to V. The transformed non-interaction proof protocol admits a message m as input. It is named Signature Proof of Knowledge (SPK), denoted as SPK$\{(w) : S\}[m]$ where w is a witness and S is a statement.

D. Blockchain and Smart Contract

Blockchain is a public, decentralized, and tamper-proof ledger that was initially used as an underlying technique to solve the double-spending problem in cryptocurrencies [43]. It integrates cryptography, economic modeling, peer-to-peer networking and decentralized consensus to achieve distributed database synchronization. Blockchain is classified into three types: public, private, and consortium. We rely on the last one to build our parking system where the blockchain stakeholders have some trust in each other and they collaborate with each other to achieve a common goal. Smart contact is a piece of codes deployed on the blockchain with a unique address and state variables. It autonomously performs the functions triggered by specific transactions in a predetermined manner. Executing functions in smart contracts raises some execution fees to incentivize peers and mitigate denial of service attacks.

V. The Proposed Scheme Mnemosyne^2

A. Overview

At a high level, Mnemosyne ^{2} consists of five phases: system initialization, parking sharing, parking querying and responding, and parking completion. In system initialization, the CS generates all system parameters for anonymous authentication and Sk NN. The SPSPs initialize the CB and DLMSC. The drivers register to one of the SPSPs. The drivers register to CB by sending a parking registration transaction Tx$^{\text{Reg}}$. The drivers and PLMs register to the CS to obtain keys and functions. In parking sharing, each SPSP collects parking lot information from their PLMs and uploads a parking status transaction Tx$^{\text{Sta}}$ to CB. In parking querying and responding, a driver sends a parking querying transaction Tx$^{\text{Que}}$ to an SPSP. They interact with each other to complete anonymous authentication. If the driver is authenticated, the SPSP forwards it to the CB to be processed by the DLMSC. If the query is matched to a parking lot, the DLMSC generates a query result $\mathcal{R}$, i.e., a receipt of Tx$^{\text{Que}}$, waiting to be packed in a new block for the driver to retrieve. In parking completion, the driver pays a parking fee to the parking lot and PLM sends a departing transaction Tx$^{\text{Dep}}$ to CB.

B. System Initialization

We propose a space encoding technique to process the location of parking lots into a multi-level structure and prepare for efficient matching.

As shown in the upper half part of Fig. 3 (PLM view), there are $l = 3$ levels in the pyramid-like structure, i.e., l_1, l_2, and l_3. All levels cover the whole service area with different granularity. From the second level l_2, the area is divided into more than one grid. Each level encodes its grids from '1' prefixed with the level number such that each grid has a unique number. In the upper

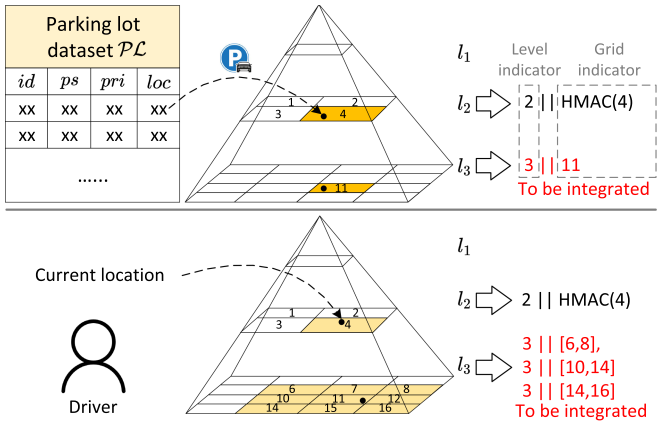


Fig. 3. Space encoding technique.

half part of Fig. 3 (driver view), the current location on the lowest level is expanded into several grids. The CS generates a secret key sk.

The n_4 SPSPs create and maintain a CB. The SPSPs all have a blockchain account by possessing a pair of public key and private key (pk, sk). A DLMSC is defined and deployed on the CB with a unique address. The drivers register to one of the SPSPs to become a candidate driver.

Each SPSP publishes on the CB a requirement $\mathcal{RE}$ including a candidate driver set $\mathcal{CD}$, a policy $\mathcal{PO}$, and a rating records list $\mathcal{LI}$. A driver D_j is assumed to satisfy $\mathcal{RE}$ if $D_j \in \mathcal{CD}$ and the scores of D_j calculated according to $\mathcal{LI}$ and $\mathcal{PO}$ satisfy $\mathcal{PO}$. We refer readers to [20] for details. The anonymous authentication is based on RSA and works in a quadratic residue group QR_N with a generator g, where N is the product of two big safe prime numbers.

Given a security parameter 1^λ, a driver runs a key generation algorithm $\mathsf{Gen}(1^\lambda)$ to generate two safe primes p, q and a prime $n = 2pq + 1$, and outputs a secret key $sk = (p, q)$ and a public key $pk = n$. To check the validity of (sk, pk), the driver checks whether p and q are safe primes with identical lengths, $n = 2pq + 1$, and n is a prime. Next, the driver computes $\pi \leftarrow \mathsf{SPK}\{(sk) : \mathsf{Ver}_1(pk, sk) = 1\}[aux, att]$, where $\mathsf{Ver}_1(\cdot)$ is a polynomial-time algorithm, s.t. $\mathsf{Ver}_1(pk, sk) = 1$ iff (pk, sk) is a valid key pair. The driver generates a pseudo identity pid and stores (pid, pk, π, att, aux) on the CB by sending a parking registration transaction:

$$\mathrm{Tx}^{\mathrm{Reg}} = [\text{“Register”}, pid, pk, \pi, att, aux, pk_{Eth}]_{sk}. \quad (1)$$

For the S k NN part, the CS generates $t+2$ (regularly updated) secret keys $\mathcal{K} = \{sk_0, sk_1, \ldots, sk_{t+1}\}$, t pseudo-random hash functions $h_1, h_2, \ldots, h_t$ where $h_i(.) = \mathsf{HMAC}_{sk_i}(.)\%m$ $(1 \le i \le t)$, a pseudo-random hash function $h_{t+1}(.) = \mathsf{HMAC}_{t+1}(.)$, and a hash function $H(.) = \mathsf{SHA256}(.)\%2$. The drivers and PLMs register to the CS to obtain $\mathcal{K}$ and $(h_1, h_2, \ldots, h_{t+1}, H)$. We define $\mathsf{Check}_1(IBF, r, \mathcal{TK})$ as checking whether a keyword w of a token $\mathcal{TK}$ exists in an IBF. For example, $\mathcal{TK} = \{110, 11*, 1**\}$ has three keywords.

C. Parking Sharing

To provide an SPS, an SPSP collects real-time parking lot information from its PLMs. Specifically, a PLM is monitoring a set of parking lots $\mathcal{PL} = \{pl_i\}$. A parking lot $pl_i = (id_i, ps_i, pri_i, loc_i)$ where id_i, ps_i, pri_i, and loc_i are the identity, parking spot number, price standard, and location of pl_i, respectively. For each pl_i, the PLM converts loc_i into a set of grid numbers $\{gn_{i2}, \ldots, gn_{il}\}$ and encodes them into a set of $l-1$ leveled location codes:

$$\begin{aligned}\mathcal{LC}_i &= \{lc_{i2}, lc_{i3}, \ldots, lc_{il-1}, lc_{il}\} \\ &= \{2||\mathsf{HMAC}_{sk}(gn_{i2}), 3||\mathsf{HMAC}_{sk}(gn_{i3}), \\ &\ldots, l-1||\mathsf{HMAC}_{sk}(gn_{il-1}), gn_{il}\}. \end{aligned} \quad (2)$$

The PLM processes $\{lc_{i2}, lc_{i3}, \ldots, lc_{il-1}\}$ as follows.

- Create an empty IBF IBF_i.
- Embed each location code lc_{ij} and a randomly chosen number r_i into IBF_i by setting for $j \in [2, l-1]$, $u \in [1, t]$:

$$IBF_i[h_u(lc_{ij})][H(h_{t+1}(h_u(lc_{ij})) \oplus r_i)] = 1, \quad (3)$$

$$IBF_i[h_u(lc_{ij})][1 - H(h_{t+1}(h_u(lc_{ij})) \oplus r_i)] = 0. \quad (4)$$

The PLM processes lc_{il} as follows.

- Compute a prefix family $\mathcal{PF}_{i1}$ of gn_{il} by using prefix encoding [44] and a prefix family $\mathcal{PF}_{i2}$ of id_i.
- Mix $\mathcal{PF}_{i1}$ with $\mathcal{PF}_{i2}$ by concatenating their prefixes to obtain a mixed code set $\mathcal{MC}_i$.
- Prefix each mixed code with the level number. In this way, we lay a foundation for the repetitive query. • Insert each mix code mc_{ij} in $\mathcal{MC}_i$ into IBF_i by setting for all $j \in [1, |\mathcal{MC}_i|]$, $u \in [1, t]$:

$$IBF_i[h_u(mc_{ij})][H(h_{t+1}(h_u(mc_{ij})) \oplus r_i)] = 1, \quad (5)$$

$$IBF_i[h_u(mc_{ij})][1 - H(h_{t+1}(h_u(mc_{ij})) \oplus r_i)] = 0. \quad (6)$$

After processing all parking lots, the PLM obtains a set of IBFs and compute a hash value HV_i for each IBF_i. The PLM builds an index tree $\mathcal{TR}$ from the bottom to up as follows. Assume that IBF is the father IBF of two IBFs: IBF^{le} (left child) and IBF^{ri} (right child), then for each $i \in [1, m]$, the value of IBF's i-th twin is the logical OR of IBF^{le}'s i-th twin and IBF^{ri}'s i-th twin

$$\begin{aligned}&IBF[H(h_{t+1}(i) \oplus r_1)][i] \\ &= IBF^{le}[i][H(h_{t+1}(i) \oplus r_2)] \vee IBF^{ri}[i][H(h_{t+1}(i) \oplus r_3)]. \end{aligned} \quad (7)$$

The hash value HV_i of each intermediate node is computed as the hash value of its left child and right child: $HV_i = \mathsf{hash}(HV_i^{le} + HV_i^{ri})$. The PLM encrypts each pl_i by using AES encryption Enc and sk_0 to obtain a ciphertext ct_i. Finally, the PLM submits to the SPSP id_{sp} the number of available parking spots ps_i of each pl_i, the $\mathcal{TR}$ including IBFs and random numbers, the ciphertexts $\mathcal{CT}$, and a hash value set $\mathcal{HV}$ including a root hash value RH. When there are multiple PLMs, the SPSP will merge their trees to form a complete tree. For each PLM,

the SPSP uploads a parking status transaction to the CB:

$$\mathrm{Tx}^{\mathrm{Sta}} = [\text{“Status”}, id_{sp}, \{(pl_i, ps_i^{ts}, ps_i^{ts-1}, \mathcal{TR}, \mathcal{CT}, \mathcal{HV}, \mathcal{B}_{i1}, \mathcal{B}_{i2})\}, pk]_{sk}, \quad (8)$$

where ps_i^{ts} (ps_i^{ts-1}) is the ps_i of pl_i in current period ts (last period $ts-1$) and $\mathcal{B}_{i1}$ ($\mathcal{B}_{i2}$) is a set of block identifiers and corresponding blocks include the parking querying transactions (departing transactions) related to pl_i during the last update period. Note that searching all the blocks for each query takes much time, we alleviate this effect by only checking the new blocks generated from the pl's last update Tx. The pl_i is to be included in a future block for generating a PoT.

D. Parking Querying

A driver D_j is cruising at a current location cl_j with an aim for pl_i and converts cl_j into a set of leveled location codes:

$$\mathcal{LC} = \{lc_{j2}, \ldots, lc_{jl}\} = \{2||\mathsf{HMAC}_{sk}(g_{j2}), 3||\mathsf{HMAC}_{sk}(g_{j3}), \ldots, l-1||\mathsf{HMAC}_{sk}(g_{jl-1}), \mathsf{Exp}(g_{jl})\}, \quad (9)$$

where $\mathsf{Exp}(g_{il})$ expands from g_{il} to a bigger area, e.g., the nearest nine grids.

D_j processes $\{lc_{j2}, lc_{j3}, \ldots, lc_{jl-1}\}$ as follows.

- Compute a location $h_v(lc_{ju}), 2 \le u \le l-1, 1 \le v \le t$.
- Compute a hash $h_{t+1}(h_v(lc_{ju})), 2 \le u \le l-1, 1 \le v \le t$.

The sub-token of a location code lc_{ju} is a t-pair of twin locations and hashes: $\{(h_1(lc_{ju})), h_{t+1}(h_1(lc_{ju})), \ldots, (h_t(lc_{ju}), h_{t+1}(h_t(lc_{ju})))\}$. Given the $l-2$ location codes, D_j now obtains a $((l-2)\times t)$-pair of twin locations and hashes. We denote $\mathcal{TK}_{j1}$ as the first part of the token $\mathcal{TK}_j$.

For the l-th location code, D_j computes a minimum set S of prefixes $\mathcal{M}_1$ for $\mathsf{Exp}(g_{jl})$ and a minimum set of prefixes $\mathcal{M}_2$ for $\mathsf{Range}_i(id_i)$. We require that $\mathsf{Range}_i(id_i) =$

$$\begin{cases} [id_i, id_i+1] \vee [id_i+2, id_i+3] \vee \ldots \vee [id_i+2|\mathcal{S}(1,n)|-2, id_i+2|\mathcal{S}(1,n)|-1], \text{if } id_i\%2 = 0 \\ [id_i-1, id_i] \vee [id_i+1, id_i+2] \vee \ldots \vee [id_i+2|\mathcal{S}(1,n)|-3, id_i+2|\mathcal{S}(1,n)|-2], \text{if } id_i\%2 = 1 \end{cases}$$

By doing so, we have $|\mathcal{M}_2| = |\mathsf{S}(pl_1, pl_{n_1})|$, i.e., the number of prefixes in $\mathcal{M}_2$ is equal to the one of $\mathsf{S}(pl_1, pl_{n_1})$. D_j mixes $\mathcal{M}_1$ with $\mathcal{M}_2$ by concatenating their prefixes to obtain a mixed code set $\mathcal{MC}$. Further, U prefixes each mixed code with the level number. We denote the set by $\mathcal{TK}_{j2}$, i.e., the second part of the $\mathcal{TK}_j$. Now D_j has the token $\mathcal{TK}_j = (\mathcal{TK}_{j1}, \mathcal{TK}_{j2})$.

Before sending a parking query, D_j needs to interact with an SPSP id_{sp} to complete anonymous authentication. D_j downloads the requirement $\mathcal{RE} = (\mathcal{CD}, \mathcal{PO}, \mathcal{LI})$ for accessing id_{sp} from the CB and checks whether she/he satisfies the $\mathcal{RE}$. If so, D_j sends a request to an SPSP id_{sp} and receives a challenge $m||id'_{sp}$ back, where m is a randomly chosen message. If $id'_{sp} = id_{sp}$, D_j generates a ticket $\mathcal{TI}_j$ and a proof π'_j. Specifically, we assume that D_j chooses the first SPSP when registering. D_j computes a ticket $\mathcal{TI}_j = \{\tau_{j1}, \tau_{j2}, \ldots, \tau_{jn_3}\}$:

$$r \leftarrow \mathbb{Z}_n,$$
$$\tau_{j1} = (t_{j1}, t_{j2}) = (g^r \bmod N, t_{j1}^{p+q} \bmod N),$$
$$\tau_{ji} \leftarrow \mathsf{TicketGen}^{(i)}(\cdot),\ i \in [2, n_3].$$

D_j computes a proof $\pi'_j =:$

$$\mathrm{SPK}\left\{(sk_j, pk_j) : \begin{array}{l} \vee_{i=1}^{n_3}(\mathsf{Ver}_1^{(i)}(pk_j, sk_j) = 1 \\ \wedge pk_j \in \mathcal{CD}_i \\ \wedge \mathsf{Ver}_2^{(i)}(sk_j, \tau_{ji}) = 1 \\ \wedge \mathsf{Ver}_3^{(i)}(\mathcal{PO}, \mathcal{LI}^{(i)}, sk_j) = 1) \end{array}\right\}[m||id_{sp}],$$

where the statement contains a proof of possession of sk_j, a proof of validity of pk_j, a proof of the validity of τ_{j1}, a proof of fulfillment of a policy. $\mathcal{CD}_i$ consists of all public keys of drivers registered to SPSP_i, and $\mathcal{LI}^{(i)}$ consists of all rating records in $\mathcal{LI}$ but for each record the ticket $\mathcal{TI}' = (\tau'_1, \tau'_2, \ldots, \tau'_{n_3})$ is replaced with τ'_i [46]. Next, D_j sends a parking querying transaction to id_{sp}:

$$\mathrm{Tx}^{\mathrm{Que}} = [\text{“Querying”}, \mathcal{TI}_j, \pi'_j, \mathcal{TK}_j, pk']_{sk'}.$$

E. Parking Responding

Upon receiving the query, id_{sp} verifies π'_j and sends $\mathrm{Tx}^{\mathrm{Que}}$ to the CB iff π'_j is valid. At the end of each blockchain period, the selected (winning) node executes the DLMSC (with pseudocode in Algorithm 1) to match $\mathcal{TK}_j$ and a previous $\mathcal{TR}$ as follows:

- Search $\mathcal{TR}$ from up to the bottom to find leaf nodes that match $\mathcal{TK}_j$. 1) Compute $\mathsf{Check}_1(IBF, \mathcal{TK}_{j1})$ for matching leaf nodes. If this match continues until the leaf level, it means there is at least one parking lot matches the query on the first $l-2$ levels. 2) At a matching leaf node with IBF, compute $\mathsf{Check}_1(IBF, \mathcal{TK}_{j2})$. If there is a match, continue to search other matched leaf nodes until k matching leaf nodes are found. If the search on current index tree is done with less than k matching leaf nodes, search other index trees.
- Proof of Correctness (PoC1). Pack the IBF, r, and HV of each matched leaf node (pl with available parking spots) into the PoC1, proving that pl matches $\mathcal{TK}_j$ regarding location.
- Proof of Completeness (PoC2). There are two parts of a token: $\mathcal{TK}_1$ and $\mathcal{TK}_2$.
  - If an index tree does not match $\mathcal{TK}_1$, the PoC2 is the IBF and random number of the root.
  - Otherwise, the search reaches the leaf level by using $\mathcal{TK}_2$. The principle is to pack the IBF and random number of (as less as possible) corresponding nodes. For example, if no leaf node matches $\mathcal{TK}_2$, then the PoC2 is also the IBF and random number of the root. if one leaf node (parking lot) matches $\mathcal{TK}_2$, but its brother leaf does not, then pack the IBF and random number of the brother node into PoC2; if two brother leaf nodes do not match, then their father node is a potential node to be packed. Finally, the PoC2 should prove that all leaf nodes in $\mathcal{TR}$ have been searched. Note that if there are already k matching leaf nodes before the

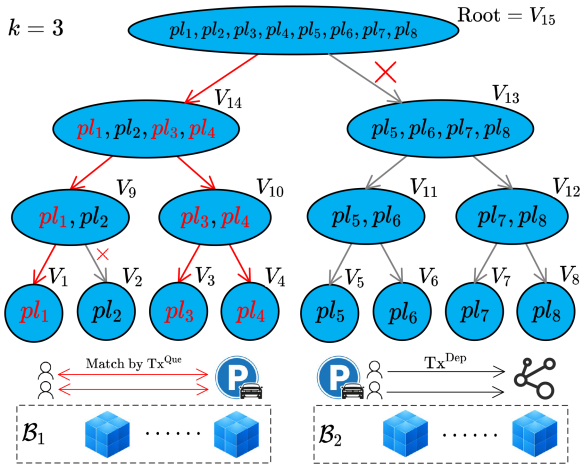


Fig. 4. An sketch of generating PoC1, PoC2, and PoT.

search is over, then pack the hash value of the highest node on the brother path as in the Merkle tree.

• Proof of Truthfulness (PoT). Pack the $\{\mathcal{B}_1, \mathcal{B}_2\}$ of each matched leaf node (pl with available parking spots) during the last update period into the PoT, proving that the claimed number of parking spots is truthfully updated. We do not include this process in the DLMSC because the Ethereum smart contracts we use in our implementation cannot query the block information. We realize it by SP's local processing.

• If $\mathcal{TK}$ is matched to a pl, generate a query result $\mathcal{R}$, i.e., a receipt of Tx^{Que}, to be packed in a new block for D_j to retrieve.

We give an example of the three proofs in Fig. 4. There are three matching parking lots pl_1, pl_3, pl_4 when $k = 3$ and three valid search paths are marked in red.

- The PoC1 is $IBF_{V_1}, r_{V_1}, HV_{V_1}, IBF_{V_3}, r_{V_3}, HV_{V_3}$, and $IBF_{V_4}, r_{V_4}, HV_{V_4}$.
- The PoC2 is HV_{V_2} and $HV_{V_{13}}$.
- The PoT is $\mathcal{B}_1$ and $\mathcal{B}_2$. The former contains the numbers of all the blocks that include Tx^{Que} matched to pl_1, pl_3, and pl_4. The latter contains the numbers of all the blocks that include Tx^{Dep} related to pl_1, pl_3, and pl_4 during the last period.

F. Parking Completion

D_j retrieves the parking result $\mathcal{R}$ in the latest block and verifies the $\mathcal{R}$ by verifying the given proofs: match $\mathcal{TK}_j$ with each IBF in PoC1, recompute a RH' from $\mathcal{HV}$ in PoC2 and compare it with RH from the CB; (if necessary) verify ps by using $\mathcal{B}_1$ and $\mathcal{B}_2$. If they all pass, D_j decrypts and chooses one parking lot to park and the corresponding PLM sends a parking transaction to the CB after D_j arrives:

$$\text{Tx}^{\text{Par}} = [\text{“Parking”}, ts, pl, pk^*]_{sk^*}.$$

Upon the completion of parking in pl, D_j remotely pays a parking fee based on anonymous electronic cash or Bitcoin [6].

Algorithm 1: Pseudocode of DLMSC.

```
create Que, k; //Parking query
create Par{}; //Parking lots
 //Map of matching results:
mapping (bytes32 => uint256) R{};
 //Brother nodes in Merkle tree:
mapping (bytes32 => uint256) Bro{};
 //Proof of correctness:
mapping (bytes32 => uint256) PoC1{};
 //Proof of completeness:
mapping (bytes32 => uint256) PoC2{};
function Query(k, TK)
  Que ← TK;
  call Match;
function Share(TR)
  Par{} ← TR;
function Match(Que, Par{})
  Initialize a stack s;
  p ← root of TR;
  while s is not empty || p is not null
    while p is not null
      if p is not leaf node
        if Check_1(p.IBF, p.r, Que[0]) = 1
          s.push(p);
          p ← p.lchild;
        else PoC2{} ← (P.IBF, P.r); break;
      if p is leaf node
        if Check_1(P.IBF, P.r, Que[1]) = 1 &&
          pl.ps >0 && |R{}| < k
          R{} ← (P.pl);
          PoC1{} ← (P.IBF, P.r, P.HV);
          s.push(p);
          p ← p.lchild;
    if s is not empty
      p ← s.pop();
      p ← p.rchild;
  Compute Bro{} of valid search paths;
  if |R{}| < k
    PoC2{} ← (IBF, r) from Bro{};
  else PoC2{} ← HV from Bro{};
  clear Que, k;
  return (R{}, PoC1{}, PoC2{});
function Parking(pl)
  pl.ps ← pl.ps - 1; //Update ps
function Departing(pl)
  pl.ps ← pl.ps + 1; //Update ps
```

Finally, the PLM sends a departing transaction to the CB:

$$\text{Tx}^{\text{Dep}} = [\text{“Departing”}, ts, pl, pk^*]_{sk^*}.$$

G. Managing a Blacklist

Recall that a driver initially retrieves the SPSP's latest requirements from the blockchain to check whether he satisfies it. The

requirement for drivers to access services of an SPSP includes three parts, namely the candidate driver set $\mathcal{CD}$, the policy $\mathcal{PO}$ and the rating records list $\mathcal{LI}$. The list $\mathcal{LI}$ encompasses rating records utilized for driver evaluation. Specifically, each rating record primarily consists of a tuple $(sid, tid, score)$, where sid is the unique string identifying the SPSP submitting the rating, tid is the unique string for the rated authentication event, and $score$ is the rating for tid. In detail, each element in $score$ is a tuple (c, ς) where c is a category and ς is a score for c.

In the setup phase, each SPSP would initialize an empty list $\mathcal{LI}$. Based on ς, the rating record list is divided into two parts: the meritlist, comprising terms with $\varsigma \geq 0$, and the blacklist, encompassing terms with $\varsigma < 0$. After the driver drivers away from the PL, the SPSP submits a rating transaction and adds a new rating record in the $\mathcal{LI}$. When an SPSP revokes a rating record, it puts a revoke transcation to the blockchain, and deletes the rating record in the $\mathcal{LI}$.

VI. Privacy and Security Analysis

In this section, we formally prove the privacy and security of Mnemosyne^2 with respect to the design objectives.

A. Privacy

1) Location Privacy: We adopt the adaptive indistinguishability under chosen-keyword attack (IND-CKA) secure model [47] and prove that Mnemosyne^2 is adaptive IND-CKA $(\mathcal{L}_1, \mathcal{L}_2)$-secure in the random oracle model. We assume that PLM uses a CPA-secure encryption scheme [48] Σ to encrypt the data items. We define two leakage functions as follows. 1) $\mathcal{L}_1(\mathcal{TR}, \mathcal{PL})$: Given $\mathcal{TR}$ and $\mathcal{PL}$, $\mathcal{L}_1$ returns $m, n_1, (pl_1, pl_2, \ldots, pl_{n_1})$, and ciphertext length $|ct|$. 2) $\mathcal{L}_2(\mathcal{TR}, \mathcal{PL}, \mathcal{TK})$: Given $\mathcal{TR}$, $\mathcal{PL}$, and $\mathcal{TK}$, $\mathcal{L}_2$ returns search pattern, i.e., whether $\mathcal{TK}$ was searched, and access pattern, i.e., which data item matches $\mathcal{TK}$.

Theorem 1: Mnemosyne^2 is adaptive IND-CKA $(\mathcal{L}_1, \mathcal{L}_2)$-secure in the random oracle model, achieving location privacy (data privacy, index privacy, and token privacy).

Proof: We build a simulator $\mathbb{S}$ that can simulate a view $V^* = (\mathcal{TR}^*, \mathcal{TK}^*, \mathcal{CT}^*)$ with the information acquired from the $\mathcal{L}_1$ and the $\mathcal{L}_2$. Then, we prove that a PPT adversary $\mathcal{A}$ cannot distinguish the SV from the real view $V = (\mathcal{TR}, \mathcal{TK}, \mathcal{CT})$.

Data privacy: To simulate the ciphertexts $\mathcal{CT}$ of $\mathcal{PL}$, the $\mathbb{S}$ acquires n and $|ct|$ from the $\mathcal{L}_1$. The $\mathbb{S}$ creates a simulated ciphertext $\mathcal{CT}^*$ with a randomly chosen plaintext and the Σ. The $\mathbb{S}$ has to make sure that the length of the $\mathcal{CT}^*$ is the same as the length of the real ciphertext. Hence, the $\mathcal{A}$ cannot distinguish the $\mathcal{CT}^*$ from the C since the Σ provides ciphertext indistinguishability.

Index privacy: To simulate the $\mathcal{TR}^*$, the $\mathbb{S}$ builds an $\mathcal{TR}*$ with the same structure. The $\mathbb{S}$ builds an IBF for each node z in $\mathcal{TR}$ while satisfying that the IBF size is the same as the length of the IBF in the $\mathcal{TR}$. In the j-th cell twin of IBF_z, the $\mathbb{S}$ either sets $IBF_z[j][0] = 0$ or $IBF_z[j][0] = 1$ which is determined by tossing a coin. Next, the $\mathbb{S}$ chooses a random number r to randomize each node. Lastly, the $\mathbb{S}$ returns IBF^* and r as the $\mathcal{TR}^*$. The $\mathcal{TR}^*$ is the same as the real index $\mathcal{TR}$. The '0's and '1's are IBF^* equally distributed in the cell twins of the IBF^* and 1-cell. Therefore, the $\mathcal{A}$ cannot distinguish the $\mathcal{TR}^*$ from the I.

Token privacy: To simulate $\mathcal{TK}^*$, the $\mathbb{S}$ knows if a received $\mathcal{TK}$ has been submitted from the $\mathcal{L}_2$. If so, the $\mathbb{S}$ returns the previous query token to the $\mathcal{A}$. Otherwise, the $\mathbb{S}$ creates a new query token $\mathcal{TK}^*$ which is a set of location hashes and locations. Specifically, the $\mathbb{S}$ uses the H to choose p-pair of hashes and locations while make sure that the chosen ones match the $\mathcal{TK}$ for the matched leaf nodes. The $\mathbb{S}$ uses the H to ensure that the p-pair of hashes and locations do not match the $\mathcal{TK}$ for the unmatched leaf nodes. Then the $\mathbb{S}$ returns the p-pair of hashes and locations as the $\mathcal{TK}^*$. Since $\mathcal{TK}^*$ is generated by random hash functions, the $\mathcal{A}$ cannot distinguish the $\mathcal{TK}^*$ from the $\mathcal{TK}$.

In conclusion, the $V*$ and the V are indistinguishable by $\mathcal{A}$. Therefore, Mnemosyne^2 is adaptive IND-CKA $(\mathcal{L}_1, \mathcal{L}_2)$-secure in the random oracle model.□

2) Identity Privacy:

Theorem 2: Mnemosyne^2 achieves identity privacy if the LD-RSA assumption [39] and the DDH-II assumption [40] hold in the quadratic residue group QR_N.

Proof: To prove this Theorem, we define the following four hybrids [49], [50], between whom the indistinguishabilities are reduced to the two assumptions. **Hybrid** H_1: This is exactly the identity indistinguishability experiment $\mathsf{PrivK}^{\mathsf{ind}}_{\mathcal{A},\Pi}(\lambda)$ and the oracle $\mathcal{O}$ is answered honestly. Specifically, when $b = 0$, the tickets $\{tk_i\}$ are returned; when $b = 1$, new tickets sampled uniformly at random from the range of the $\mathsf{TicketGen}(\cdot)$ are returned. **Hybrid** H_2: This is identical to H_1 except that when $b = 0$, a secret key sk' is generated and tickets of sk' are returned. Indistinguishability between H_1 and H_2 comes from the LD-RSA assumption. **Hybrid** H_3: This is identical to H_2 except that when $b = 0$, h is sampled freshly and uniformly from QR_N, a random number is sampled $r \xleftarrow{\$} \mathbb{Z}_N$, and (h, h^r) is returned. Indistinguishability between H_2 and H_3 comes from the DDH-II assumption. **Hybrid** H_4: This is identical to H_3 except that when $b = 0$, h_1, h_2 are sampled freshly and uniformly from QR_N. (h_1, h_2) is returned. Indistinguishability between H_3 and H_4 comes from the standard DDH assumption, which can be implied by the DDH-II assumption. Note that in H_4, the $\mathcal{O}$ is answered identically for both the case $b = 0$ and the case $b = 1$. Therefore, $\Pr[\mathsf{PrivK}^{\mathsf{idn}}_{\mathcal{A},\Pi}(\lambda) = 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$, completing the proof.□

B. Secure Repetition

1) Obliviousness:

Theorem 3: Mnemosyne^2 achieves obliviousness if the underlying HMAC is a pseudo-random function [51].

Proof: We assume that $\mathcal{A}$ attacks Π with an upper bound ub on the number of Π's executions. We briefly describe an indistinguishability experiment $D^{\mathsf{HMAC},f}(\lambda)$: a PPT adversary $\mathcal{A}'$, which generates a message m and has an oracle access to $\mathsf{HMAC}(\cdot)$, is given an output of either the HMAC or a random function f on m and distinguishes whether it is HMAC or f. We say that HMAC is a pseudo-random function if there is a negligible function negl such that $|\Pr[D^{\mathsf{HMAC}(\cdot)}(\lambda) = 1] -$

$|\Pr[D^{f(\cdot)}(\lambda)=1]| \leq \mathsf{negl}(\lambda)$. Now we construct the efficient adversary $\mathcal{A}'$ that runs $\mathcal{A}$ to attacks Π', i.e., HMAC.

Algorithm $\mathcal{A}'$:

The algorithm is given all public parameters of Π.

1) Generate messages $\mathcal{M}=\{m_1, m_2, \ldots, m_{ub}\}$.

2) Query $\mathcal{M}$ to $\mathsf{HMAC}(\cdot)$ and obtains ub outputs.

3) Generate a message $m_c \notin \mathcal{M}$ and send it to the challenger.

4) Receive an output $\mathsf{HMAC}(m_c)$.

5) Choose a random value r in the range of the $\mathsf{HMAC}(\cdot)$ and return $(\mathsf{HMAC}(m_c), r)$ to $\mathcal{A}$.

6) Output what $\mathcal{A}$ outputs.

The view of $\mathcal{A}$ when run as a subroutine by $\mathcal{A}'$ in experiment $D^{\mathsf{HMAC},f}(\lambda)$ is identical to the view of $\mathcal{A}$ in experiment $\mathsf{PrivK}^{\mathsf{obl}}_{\mathcal{A},\Pi}(\lambda)$. Therefore, we have $\Pr[D^{\mathsf{HMAC},f}(\lambda)=1] = \Pr[\mathsf{PrivK}^{\mathsf{obl}}_{\mathcal{A},\Pi}(\lambda)=1]$. If $\mathcal{A}$ successfully distinguishes a repetitive query and a normal query, then Π is breakable, i.e., do not guarantee obliviousness, where $\mathcal{A}'$ can invoke it to break the pseudo-randomness of HMAC. However, since Π' is secure, then $\Pr[D^{\mathsf{HMAC},f}(\lambda)=1]$ is negligible and $\Pr[D^{\mathsf{HMAC}(\cdot)}(\lambda)=1] \leq \frac{1}{2}+\mathsf{negl}(\lambda)$, it immediately implies that $\Pr[\mathsf{PrivK}^{\mathsf{obl}}_{\mathcal{A},\Pi}(\lambda)=1] \leq \frac{1}{2}+\mathsf{negl}(\lambda)$. We complete the proof. □

2) Unlinkability: Mnemosyne^2 achieves unlinkability for three reasons. First, there is no identifiable information used in token generation. Second we require that for a driver's two queries q_0 and q_1, it satisfies that $\mathsf{Exp}(cl_0) \neq \mathsf{Exp}(cl_1) \vee id_0 \neq id_1$, which leads to two different mix code sets. The former case indicates two different current locations, i.e., different grid numbers. The latter means that the driver explicitly asks for two different parking lots. Last, we require that the secret keys $\mathcal{K}=\{sk_0, sk_1, sk_2, \ldots, sk_{t+1}\}$ are regularly updated. Therefore, the adversary $\mathcal{A}$ can correctly guess i by 1) randomly guessing i and 2) correctly guessing the secret keys and the mixed code set, which leads to $\Pr[\mathsf{PrivK}^{\mathsf{unl}}_{\mathcal{A},\Pi}(\lambda)=1] = \Pr[\text{Case 1}] + \Pr[\text{Case 2}]$. In case 2, we need to compute the minimum number of prefixes in a mixed code set $\mathcal{MC}$. Given a range $[a,b]$, where a and b are two numbers of w bits, the number of prefixes in $S([a,b])$ is at least $2w-2$ [52]. The minimum number of prefixes in $\mathsf{S}(pl_1, pl_{n_1})$ is $2\log_2(pl_{n_1}+1)-2$. The number of prefixes in $\mathcal{M}_1$ is 6. Hence, the minimum size of $\mathcal{MC}$ is $12\log_2(pl_{n1}+1)-12$. Now we have $\Pr[\mathsf{PrivK}^{\mathsf{unl}}_{\mathcal{A},\Pi}(\lambda)=1] \leq \frac{1}{n_2} + \frac{1}{2^{(t+1)\lambda} * 2^{12\log_2(pl_{n1}+1)-12}}$, which is not bigger than $\frac{1}{n_2}+\mathsf{negl}(\lambda)$. □

C. Full Verifiability

The full verifiability refers to correctness, completeness, and truthfulness. 1) Correctness. For each returned leaf node (parking lot), the DLMSC attaches its IBF and random number for the driver to re-search the token on the IBF, thus guaranteeing the correctness of the returned parking lot. 2) Completeness. If there are less than k matching parking lots, the DLMSC packs the IBFs and random numbers of corresponding nodes in the index tree such that the driver can verify the mismatching nodes. It also packs the hash value of the unsearched nodes when there are already k matching leaf nodes for the driver to re-compute the hash value of the root, thus guaranteeing the completeness of the search on the index tree. 3) Truthfulness. It is a new definition that we propose for smart parking. Truthfulness includes (a) PLMs truthfully report the number of available parking spots and (b) the returned result has enough available parking spots. The first one is guaranteed by the semi-honest assumption of PLM. The second one is secured by the PoT. For each parking lot, when there are matching queries in each time period, the DLMSC retrieves the number of previous blocks that include the parking querying transactions (IN) and departing transactions (OUT) pertinent to this parking lot. The drivers who receive a query result can verify the number of parking spots in the parking lot by counting the IN and OUT activities. Given that the number of parking spots are truthfully reported and recorded on the blockchain, consecutively proving that the claimed number of parking spots is truthfully updated in the last period will form a chain of continuity and integrity of such a number. Therefore, truthfulness holds.

TABLE III
DATASET

City	pl	Max ps	Min ps	Ave ps	price ($/h)
Los Angeles	62	1256	14	237	[1, 46.2]

D. Discussions of Sharing Secret Keys

In the design of query processing [15], [17], [44], the data owner shares the same secret keys with data users. In the parking scenario, we follow this setting by mapping PLM to data owner, and mapping driver to data user. At first glance, it may seem incompatible since the PLM and drivers are not acquainted and do not share information beforehand. However, the PLM and drivers are not malicious, and they participate in the parking system to gain profits and request parking services. The unique background and tacit understanding smoothly transform into a relation that resembles the one between owner and user in query processing.

For security, even though the PLM is semi-honest, it cannot acquire the plaintext of drivers' location or the requested PL identity since the location is projected into a grid and the PL identity is hidden in a customized range. Further, to protect the keys from the PLM, we can remotely attest codes in a Trusted Execution Environment (TEE), e.g., Intel SGX enclave [53], [54], [55], store the secret keys in it, and then run index building in the secure enclave with confidentiality and integrity.

VII. Performance Analysis

In this section, we show how to build a prototype of Mnemosyne^2 and analyze its performance regarding computational cost, communication overhead, and scalability. We also compare Mnemosyne^2 with existing PPSP schemes.

A. Experiment Settings

Dataset: We have collected a city map as our SPS areas including the information of parking lots. Each parking lot has an identity, two location coordinates, the number of parking spots and price standard. We list the detailed information of the collected dataset in Table III.

TABLE IV
EXPERIMENTAL PARAMETERS

Parameter	Value	Parameter	Value
k	5	n_3	4
t	5	$\|sk\|$	256
m	9140	$\|p\|$	1024
n_1	62	$\|q\|$	1024
n_2	[100, 500]	$\|sk'\|$	256
pr	1%	$\|r\|$	256

Parameters: We list the key experimental paraments in Table IV, including the value of query parameter k, the number of pseudo-random hash functions t, the false positive rate pr, the *IBF* size m, the number of parking lots n_1, the number of drivers n_2, the number of SPSPS n_3, the lengths of secret keys sk, the random number r, two safe primes p, q, and the AES symmetric key sk'.

Metrics: We evaluate computational costs and communication overhead for Driver, PLM, CS and SP. For computational costs, we measured the running time of registration, PLM and SP's sharing, Driver's querying, SP's responding, and Driver and PLM's completion. For communication overhead, we measured the size of the transmitted messages during the five main phase, i.e., registration, sharing, querying, responding and completion.

Setup: We instantiate Mnemosyne[2] on a PC server running Win11 Home with a 10th Gen Intel (R) Core (TM) i5-10210 U CPU @ 1.60 GHz processor, and 8 GB RAM. We use $\mathsf{HMAC-SHA256}$ as the pseudo-random function to implement the hash functions of *IBF*. We use AES as the symmetric encryption. We write the smart contract with an online integrated development environment remix (remix.ethereum.org) and deploy it via metamask (metamask.io) which is a light-weighted browser plugin. We use puppeth (github.com/ethereum/go-ethereum/tree/master/cmd/puppeth) to create the genesis block with Proof-of-Authority consensus mechanism (Clique) and the block creation time is set as 1 s. It can be adjusted according to the application. We use Java (JDK8) to implement cryptographic primitives. The work mode of AES in this work is Cipher-Block Chaining (CBC). We have uploaded all source codes of Mnemosyne[2] on Github: https://github.com/UbiPLab/DecPark.

B. Computational Cost

We use different notations to stand for cryptographic operations: mu_i, ex_i, di_i denote multiplication, exponentiation, and division in $\mathbb{G}_i$ of prime order p' $(i = 1, 2, 3)$, respectively. g_i is the generator of $\mathbb{G}_i$ $(i = 1, 2, 3)$. h/H and bp denote hash function and bilinear pairing. Gen, Ver, Sig, Ecd denote generation, verification, signing, and encoding. Ins, Che, Del denote inserting, checking, and deleting in an IBF, a Bloom filter [4], a cuckoo filter [6], a hashmap [7], or an exchange pool [8]. Enc and Dec denote AES encryption and decryption. Enc' and Dec denote the ElGamal encryption and decryption. CSC denotes creating a smart contract [9]. Specifically, in P-SPAN [4], PriAV [6], ASAP [7], and PEPS [8], the bilinear pairing is e: $\mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_3$; in SAVP [5], the bilinear pairing is e: $\mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$. We record the theoretical results in Table V.

The CS in registration processes the registration requests from drivers and PLMs, namely generating hash functions, i.e., $n(h + H)$, it costs $1.85n$ ms.

A driver's primary computational cost rests in registration, querying and completion. Registration consists of generating a key pair (pk, sk), computing a signature of knowledge π, signing the registration transaction, i.e., $\mathsf{Gen} + \mathsf{Gen}_\pi + \mathsf{Sig}$, it costs the driver $134 + 122 + 10 = 266$ ms. Querying consists of computing a ticket tk, computing a signature of knowledge π', computing a query token qt, signing the querying transaction, i.e., $\mathsf{Gen}_{tk} + \mathsf{Gen}_{\pi'} + \mathsf{Gen}_{qt} + \mathsf{Sig}$, it costs $5 + 631 + 4 + 15 = 655$ ms. Completion consists of decrypting parking result $\mathcal{R}$, verifying the given proofs PoC1, PoC2 and PoT, i.e., $\mathsf{Dec} + \mathsf{Ver}_1 + \mathsf{Ver}_2 + \mathsf{Ver}_3$, it costs $3 + 3 + 2 + 170 = 178$ ms.

For a PLM, the most computational cost lies in registration, sharing and completion. Registration consists of generating a key pair (pk, sk), computing a signature of knowledge π, signing the registration transaction, i.e., $\mathsf{Gen} + \mathsf{Gen}_\pi + \mathsf{Sig}$, it costs the PLM $123 + 116 + 10 = 249$ ms. Sharing consists of encoding and encrypting parking lots $\mathcal{PL}$, inserting location codes and prefixes into *IBFs*, computing the hash value set HV, i.e., $\mathsf{Ecd} + \mathsf{Enc} + \mathsf{ins} + h$, it costs $5 + 728 + 3045 + 127 = 3905$ ms. Completion consists of signing parking transaction Tx^{Par} and departing transaction transaction Tx^{Dep}, i.e., $2\mathsf{Sig}$, it costs $2 * 9 = 18$ ms.

The major computational cost of a SP rests in registration, sharing and responding. Registration consists of verifying PLMs and drivers' signature and proof π, i.e., $n(\mathsf{Ver}_\sigma + \mathsf{Ver}_\pi)$, it costs it costs the SP $n(2 + 122) = 124n$ ms. Sharing consists of signing status transaction Tx^{Sta}, computing the number of querying transactions B_1 and departing transactions B_2, updating parking spots ps, i.e., $\mathsf{Sig} + \mathsf{Gen}_{\mathcal{B}_1} + \mathsf{Gen}_{\mathcal{B}_2} + \mathsf{Upd}$, it costs $17 + 125 + 2 = 144$ ms. Responding consists of verifying signature of drivers' querying transaction and proof π', searching index tree $\mathcal{TR}$, generating proof of correctness PoC1, proof of completeness PoC2 and proof of Truthfulness PoT, i.e., $\mathsf{Ver}_\sigma + \mathsf{Ver}_{\pi'} + \mathsf{Mat} + \mathsf{Gen}_1 + \mathsf{Gen}_2 + \mathsf{Gen}_3$, it costs $2 + 536 + 30 + 125 = 693$ ms.

C. Communication Overhead

The CS in registration returns secret keys and hash functions for all the registered drivers and PLMs, i.e., $n(t|sk| + |h| + |H|) = n * (t * 256 + 8 * 6 + 8 * 6) = (256t + 96)n$ bits. A driver sends a registration transaction Tx^{Reg}, i.e., $|\text{“}Register\text{”}| + |pid| + |pk| + |att| + |aux| + |\sigma| + |\pi| + |pk_{Eth}| = 8 * 8 + 16 + 2048 + 16 + 16 + 520 + (2048 + 2048 + 2048) * 5 + 256 + 256 = 33912$ bits $= 4.14$ Kbytes. The driver also sends a registration request to CS, it consists of identity of driver, i.e., $|id| = 16$ bits. At the phase of querying, the driver sends a querying transaction Tx^{Que}, i.e., $|\text{“}Querying\text{”}| + |\sigma| + |pk| + |tk| + |qt| + |\pi'| = 8 * 8 + 520 + 256 + 2048 * 3 + (14 + 256) * 5 * 2 + (14 + 256) * 60 * 5 + (6 + 8 + 14 + 8 + 18 + 16) * 2048 + 3 * 256 = 234812$ bits $= 28.66$ Kbytes. The PLM submits a registration request to CS and SP, i.e., $|id| = 8$ bits, $|\text{“}Register\text{”}| +$

TABLE V
COMPARISON OF COMPUTATIONAL COSTS

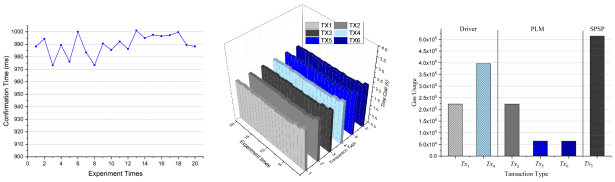
Scheme	Setup and Registration				Sharing		Querying	Responding	Completion	
	Driver	PLM	CS	SP	PLM	SP	Driver	SP	PLM	Driver
[4]	$5ex_1+3ex_2+4mu_2+\mathsf{Gen}_{\pi_1,\pi_2}^{1a}$	$ex_1+n_2(ex_1+di_1+4ex_2+3mu_2+3bp)^{1b}$	$n_2(4ex_1+mu_1+2ex_2+ex_2+di_2+\mathsf{Ver}_{\pi_1}+3bp)$	$(4+2n_2)ex_1+n_2\mathsf{Ver}_{\pi_2}$	n/a	n/a	$13ex_1+3mu_1+4ex_2+mu_2+2ex_3+\mathsf{Gen}_{\pi_3}+7H+\mathsf{Enc}+\mathsf{Dec}$	$n_2(2ex_1+mu_1+6ex_2+4mu_3+5H+6bp+\mathsf{Ins}+\mathsf{Che}+\mathsf{Del})^{1c}+n_2(5ex_1+mu_1+5ex_2+2mu_3+4H+3bp+\mathsf{Enc}+\mathsf{Dec})^{1d}$	n/a	n/a
[5]	$3ex_1+2mu_1+bp+\mathsf{Gen}_{\pi_1}$	$2ex_1$	n/a	$4ex_1+n_2(\mathsf{Ver}_{\pi_1}+ex_1+mu_1)$	n/a	n/a	$ex_1+2H+2bp+\mathsf{Gen}_{\pi_2}+\mathsf{Gen}_{\pi_3}$	$(n_2(2H+4bp))^{2a}+(n_2(\mathsf{Ver}_{\pi_2,\pi_3}+H))^{2b}$	1	1
[6]	$(13ex_1+8mu_1+3ex_2+mu_2+2H+2bp+\mathsf{Gen}_{\pi_1})^{3a}$	ex_1	ex_2	$ex_2+5bp+n_2(\mathsf{Ver}_{\pi_1}+4ex_1+4mu_1)$	n/a	n/a	$(12ex_1+6mu_1+3ex_2+4ex_3+4mu_3+7H+3bp+\mathsf{Enc}+3\mathsf{Enc}'+\mathsf{Dec}+\mathsf{Dec}'+\mathsf{Gen}_{\pi_2,\pi_3})^{3a}$	$n_2(2ex_1+mu_2+3ex_2+2H+2\mathsf{Ver}_{\pi_2,2\pi_3}+3\mathsf{Dec}'+\mathsf{Ins}+2\mathsf{Che}+\mathsf{Del})^{3b}$	1	1
[7]	$2ex_1+ex_2+\mathsf{Gen}_{\pi}$	n/a	$2ex_2+n(5ex_1+mu_1+2bp+\mathsf{Gen}_{\pi})^{4a}$	n/a	$3ex_2+mu_2+2ex_1+ex_3+H^{4b}$	$ex_2+mu_2+3ex_3+2mu_3+H$	$2ex_1+3ex_2+mu_2+ex_3+H$	$n_2(ex_2+mu_2+3ex_3+2mu_3+H+\mathsf{Che})$	n/a	n/a^{4c}
[8]	$(4+m')ex_1+(2+m')mu_1+\mathsf{Gen}_{\pi_1}^{5a}$	Gen_{π_1}	n/a	$ex_1+(n_2+n_1)\mathsf{Ver}_{\pi_1}^{5b}$	$6ex_1+2mu_1+H+\mathsf{Gen}_{\pi_2,\pi_3}$	$n_1(ex_1+mu_1+H+\mathsf{Ver}_{\pi_2,\pi_3}+\mathsf{Ins})$	$2ex_1+mu_1+H+\mathsf{Gen}_{\pi_3}$	$n_2(\mathsf{Ver}_{\pi_3}+\mathsf{Che})$	n/a	n/a
[9]	$2ex_1+mu_1+H+\mathsf{Gen}_{\pi_1}+\mathsf{Sig}+\mathsf{OPRF}_l^{6a}$	$H+\mathsf{Sig}$	$n_2\mathsf{Gen}_{Cre}+n_1\mathsf{Gen}_{Cer}^{6b}$	$n_2\mathsf{OPRF}_r$	$2H+\mathsf{Gen}_{Tx}+\mathsf{Sig}$	CSC^{6c}	Gen_{π_2}	$n_2(\mathsf{Ver}_{\pi_2}+\mathsf{Mat})$	1	1
Ours	$\mathsf{Gen}+\mathsf{Gen}_{\pi}+\mathsf{Sig}$	$\mathsf{Gen}+\mathsf{Gen}_{\pi}+\mathsf{Sig}$	$(n_2+n_3)(h+H)$	$(n_2+n_3)(\mathsf{Ver}_{\sigma}+\mathsf{Ver}_{\pi})$	$\mathsf{Ecd}+\mathsf{Ins}+h+\mathsf{Enc}$	$n_3(\mathsf{Sig}+\mathsf{Gen}_{\mathcal{B}_1,\mathcal{B}_2}+\mathsf{Upd})$	$\mathsf{Sig}+\mathsf{Gen}_{tk}+\mathsf{Gen}_{qt}+\mathsf{Gen}_{\pi'}$	$n_1(\mathsf{Ver}_{\sigma}+\mathsf{Ver}_{\pi'}+\mathsf{Mat}+\mathsf{Gen}_{1,2,3})$	$2n_2\mathsf{Sig}$	$\mathsf{Dec}+\mathsf{Ver}_{1,2,3}$

$1a$: π_1 and π_2 are zero-knowledge proofs; $1b$: P-SPAN hash no PLMs and employs RSUs to construct two Bloom filters to help drivers navigate to a parking lot;
$1d$: RSU's operations; $1e$: SP's operations. $2a$: PLM's operations; $2b$: SP's operations. $3a$: Operations of driver and autonomous vehicle.
$3b$: The operations of PLM and SP. $4a$: π is signature of knowledge of sk; n is number of drivers and suppliers registering to CS; $4b$: sharing entity is a supplier;
$4c$: ASAP has an anonymous payment after parking and we do not compare with it because this step is not our focus. $5a$: m' is the number of attributes;
$5b$: Fog servers undertake SP's tasks. $6a$: Driver interacts with SP to obtain OPRF hashing value; $6b$: Cre is anonymous credential and Cer is public key certificate.

TABLE VI
COMPARISON OF COMMUNICATION OVERHEAD

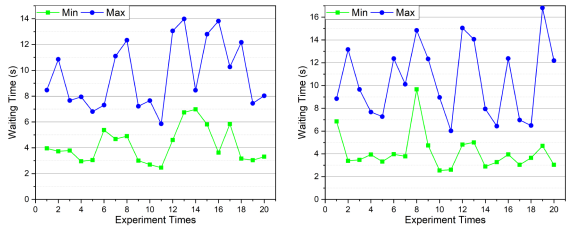
Scheme	Registration				Sharing		Querying	Responding	Completion	
	Driver	PLM	CS	SP	PLM	SP	Driver	SP	PLM	Driver
[4]	$2\|id\|+3\|g_1\|+3\|g_2\|+\pi_1+\pi_2$	0	$n_2(\|id\|+2\|g_1\|+\|g_3\|)$	$2\|g_1\|$	n/a	n/a	$3\|r\|+2\|ts\|+7\|g_1\|+3\|g_2\|+\|\mathsf{Enc}\|+\|\pi_3\|+2\|H\|$	$(n_2n_3'(\|ts\|+3\|g_1\|+2\|r\|)^{1a}+n_2(3\|id\|+\|ts\|+5\|g_1\|+4\|r\|+\|str\|)^{1b}$	n/a	n/a
[5]	$\|id\|+\|g_1\|+\|H\|+\|\pi_1\|$	$\|id\|+\|g_1\|$	n/a	$\|g_1\|+\|r\|$	n/a	n/a	$\|r\|+\|str\|+3\|loc'\|+\|ts\|+\|H\|+\|g_1\|+\|\pi_2\|+\|\pi_3\|^{2b}$	$(\|r\|)^{2a}+(\|str\|+2\|r\|+\|ts\|+2\|H\|)^{2c}$	1	1
[6]	$(3\|r\|+2\|ts\|+4\|g_1\|+2\|g_2\|+\|\pi_1\|)^{3a}$	0	0	$3\|r\|+\|g_1\|$	n/a	n/a	$(4\|str\|+4\|r\|+2\|g_1\|+3\|g_2\|+2\|g_3\|+2\|\mathsf{Enc}\|+3\|\mathsf{Enc}'\|+\|\pi_2\|+\|\pi_3\|)^{3a}$	$(\|str\|+4\|r\|+4\|g_1\|+5\|g_2\|+\|g_3\|+4\|\mathsf{Enc}\|+\|\mathsf{Enc}'\|+\|\pi_2\|)^{3b}$	1	1
[7]	$\|id\|+3\|g_1\|+\|g_2\|+\|\pi\|$	n/a	$2\|g_1\|+\|g_3\|+\|p'\|$	1	$\|id\|+2(\|ts\|+\|g_1\|+\|g_2\|)+\|H\|+\|p'\|$	n/a	$\|id\|+2(\|ts\|+\|g_1\|+\|g_2\|)+\|H\|+\|p'\|$	$\|id\|+2\|g_1\|+\|Supp\|+\|K\|+\|pr\|^4$	n/a	1
[8]	$2\|g_1\|+\|\pi_1\|+m\|r\|+\|str\|$	$2\|g_1\|+\|\pi_1\|+m\|r\|+\|str\|$	n/a	0	$\|pid\|+2\|ts\|+2\|g_1\|+\|g_2\|+\|\pi_2\|+\|\pi_3\|$	$n_1\|r\|$	$\|pid\|+2\|ts\|+2\|g_1\|+\|g_2\|+\|\pi_3\|$	$n_2(3\|r\|+\|pid\|)$	n/a	n/a
[9]	$\|id\|+\|H\|+\|\mathsf{OPRF}_l\|$	$\|id\|+\|H\|$	$n_2\|Cre\|+n_1\|Cer\|$	$n_2\|\mathsf{OPRF}_r\|$	$\|Cre\|+\|str\|+6\|loc\|+\|ts\|+\|pk\|+\|H\|+\|\sigma\|^6$	0	$\|ts\|+\|\pi_2\|$	$n_2(2\|loc\|)$	1	1
Ours	$\|id\|+\|pid\|+\|str\|+\|pk\|+\|att\|+\|aux\|+\|\sigma\|+\|\pi\|+\|pk_{Eth}\|$	$\|id\|+\|pid\|+\|str\|+\|pk\|+\|att\|+\|aux\|+\|\sigma\|+\|\pi\|+pk_{Eth}$	$(n_2+n_3)(t\|sk\|+\|h\|+\|H\|)$	0	$\|\mathcal{TR}\|+\|ct\|+\|HV\|+\|ps\|$	$n_3(\|\sigma\|+\|id\|+\|str\|+\|\mathcal{TR}\|+\|ct\|+\|HV\|+\|\mathrm{PoT}\|+\|ps\|)$	$\|\sigma\|+\|str\|+\|pk\|+\|tk\|+\|qt\|+\|\pi'\|$	$n_2(\|\mathcal{R}\|+\|\mathrm{PoC1}\|+\|\mathrm{PoC2}\|+\|\mathrm{PoT}\|)$	$2n_2(\|\sigma\|+\|str\|+\|pl\|+\|ts\|+\|pk\|)$	0

$1a$: Messages of an RSU, str is the parking query sent by a driver; $1b$: SP's messages, n_3' is the number of RSU. $2a$: PLM's messages;
$2b$: loc' is a location coordinate under a geo-indistinguishable mechanism; $2c$: SP's operations. $3a$: This part includes messages of driver and autonomous vehicle;
$3b$: Messages of PLM and SP. 4: $Supp$: cloaked parking location, K: public key, pr: price. 6: loc includes a pair of location coordinates.

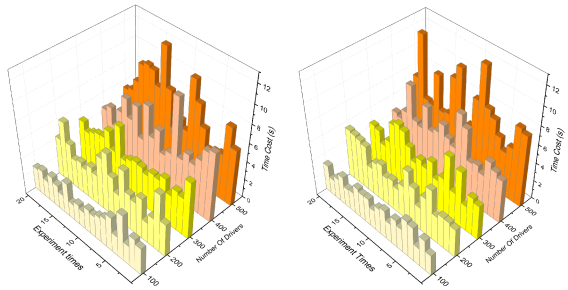


(a) Blockchain Consensus time (b) Tx Confirmation Time (c) Gas Cost

Fig. 5. Basic blockchain performance.



(a) n_2=500, D_j queries fixed pl (b) n_2=500, D_j queries random pl



(c) Varied n_2, D_j queries fixed pl (d) Varied n_2, D_j queries random pl

Fig. 6. Waiting time of one driver.

$|pid| + |pk| + |att| + |aux| + |\sigma| + |\pi| + |pk_{Eth}| = 4.14$ Kbytes. At the phase of sharing, the PLM also sends the index tree, the ciphertexts, a hash value set and the number of available parking spots of each parking lot to SP, i.e., $|\mathcal{TR}| + |ct| + |HV| + |ps| = 68.06$ Kbytes. When a driver chooses one parking lot to park, the PLM sends a parking transaction Tx^{Par}, i.e., $|\text{“}Parking\text{”}| + |\sigma| + |pl| + |ts| + |pk| = 0.10$ Kbytes, after the completion of parking, the PLM sends a departing transaction Tx^{Dep}, i.e., $|\text{“}Departing\text{”}| + |\sigma| + |pl| + |ts| + |pk| = 0.11$ Kbytes. For each PLM, the SP uploads a parking status transaction Tx^{Sta}, i.e., $|\sigma| + |id| + |str| + |\mathcal{TR}| + |ct| + |HV| + |\text{PoT}| + |ps| = 68.60$ Kbytes. For a query request, the SP sends the query result to the related driver, i.e., $|\mathcal{R}| + |\text{PoC1}| + |\text{PoC2}| + |\text{PoT}| = 11.65$ Kbytes. We record the results in Table VI.

D. Basic Blockchain Performance

Now we analyze the basic blockchain performance, i.e., blockchain consensus time, transaction confirmation time, and gas cost.

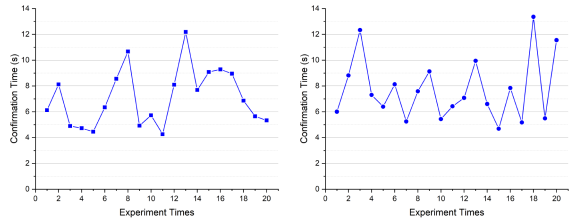
We set the block period to be 1 s. From the Fig. 5(a), the real consensus time fluctuates around 1 s because of the hardware interference. Next, we use $ethGetTransactionReceipt()$ to compute the confirmation time of the six types of transactions and use $System.currentTimeMillis()$ to record the elapsed time. The results are illustrated in Fig. 5(b). The interval time between two transactions of six types is 1 s. Due to network delay and consensus mechanism, the confirmation time of transactions is approximately 2 seconds. The transaction confirmation time can be reduced by decreasing the consensus time in the blockchain network according to application's requirements. We measure the gas cost of six types of transactions. Specifically, they are divided into three sets: Tx_1 and Tx_4 for driver; Tx_2, Tx_5 and Tx_6 for PLM; and Tx_3 for SPSP. Fig. 5(c) indicates that the average gas usage for Tx_4 is about 396093. We set the gas price to be 1 Gwei (0.000000001 Ether) in the Mnemosyne^2 blockchain, at the time of writing (August 9, 2022), the exchange rate is \$1,774.47 USD per Ether. Each Tx_4 costs about 0.0004 Ether (0.7 USD). The gas varies because the transaction items are different. Even when two transactions are of the same type, the query token and the proof vary for different driver in Tx_4.

We build our own blockchain test network based on Ethereum to adjust experimental parameters, such as blockchain creation time and gas price. We set the blockchain creation time as 1 s to reduce response time for parking queries. We set the gas price as 1 Gwei to reduce the transaction fee drivers. Such a capability of regulating parameters enable the parking system to have more flexibility in different parking scenarios. Layer 2 (L2) is a collective term to describe a specific set of Ethereum scaling solutions. An L2 is a separate blockchain that extends Ethereum and inherits Ethereum's security guarantees. L2 testnets are usually coupled to public Ethereum testnets [56]. For gas price, the L2 gas price on an Arbitrum chain has a set floor, which can be queried via ArbGasInfo's getMinimumGasPrice method (currently 0.1 gwei on Arbitrum One and 0.01 gwei on Nova) [57]. However, there is no regulatory party and anyone can freely join, which is not applicable to smart parking. L2 systems have limited scalability potential, while approaches offering better performance, sacrifice security and result in an increase in centralization. L2 projects also impose a load on the main-chain to constitute a severe bottleneck, preventing them from reaching their alleged maximum throughput levels [58]. Many L2 projects are still in an early stage and require users to trust some operators to be honest. L2 projects also contain additional risks compared to holding funds and transacting directly on Ethereum Mainnet. For instance, sequencers may go down, leading you to have to wait to access funds [59]. In this work, we realize a prototype system to evaluate the performance. The current test network suffices to do the job.

E. Scalability

Now we analyze the scalability of Mnemosyne^2, i.e., the maximum (minimum, average) waiting time of one driver and the average waiting time of multiple drivers when there are fixed/varied number of drivers that have different parking choices. The rationale is to see how n_2 and the choices of parking lots affect the waiting time.

In Fig. 6(a), the number of drivers as $n_2 = 500$ and each driver requests a fixed parking lot. The maximum (minimum) waiting



(a) All drivers query a fixed pl (b) 25% drivers query a fixed pl

(c) 50% drivers query a fixed pl (d) 75% drivers query a fixed pl

Fig. 7. Average waiting time of multiple drivers.

time of one driver is only 2.46 (13.98) seconds. In Fig. 6(b), we randomize the parking choices of drivers to find that the maximum (minimum) waiting time of one driver is only 2.55 (16.81) seconds, showing no obvious impacts from the parking choices.

In Fig. 6(c), n_2 varies from 100 to 500 and the drivers request a fixed parking lot. We conduct 20 sets of experiments for each choice of n_2. The average waiting time of one driver ranges from 1.75 seconds to 12.19 seconds. In Fig. 6(d), we randomize the parking choices of drivers to find that the average waiting time of one driver ranges from 1.87 seconds to 12.21 seconds.

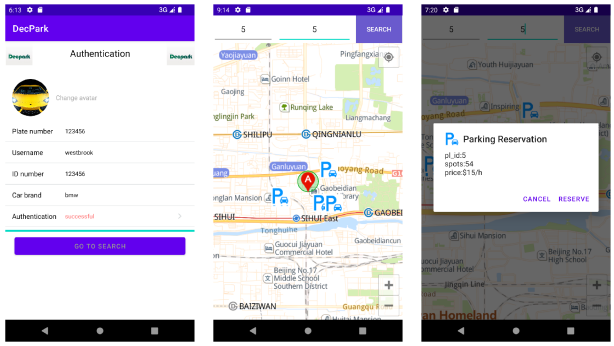
From Fig. 7(a), (b), (c), and (d), we test the average time of all drivers under different settings where a part of drivers request a fixed parking lot. The results show that the average waiting time ranges between 4 seconds and 14 seconds, indicating no obvious impacts from the parking choices. After the drivers receive query results, it takes 175 ms on average for verification (on PoC1, PoC2, and PoT).

F. Managing a Blacklist

There are three steps involved in managing a blacklist. An SPSP 1) puts its requirement to the blockchain by sending a transaction, i.e., $|\text{“Upload”}| + |\sigma| + |sid| + |\mathcal{CD}| + |\mathcal{PO}| + |\mathcal{LI}| + |pk| = 564.39 + 0.25 * |\mathcal{CD}| + 1.25 * |\mathcal{LI}|$ Kbytes, 2) submits a rating for the anonymous user in an authentication event, i.e., $|\text{“}Rating| + |\sigma| + |rid| + |sid| + |tid| + |score| + |pk| = 1.35$ Kbytes. Rating consists of signing the rating transaction and adding a new rating record, it costs 11 ms, and 3) sends a revoke transaction to revoke a rating record, i.e., $|\text{“Revoke”}| + |\sigma| + |rid| + |sid| + |pk| = 0.85$ Kbytes. Revoke consists of signing the revoke transaction and deleting a rating record, it costs 10 ms.

G. Comparison

Now we compare Mnemosyne^2 with related work regarding computational costs and communication overhead. Both



(a) Authentication (b) Querying (c) Reservation

Fig. 8. Screenshots of app mnemosyne^2.

P-SPAN [4] and ASAP [7] utilize the short randomizable signatures [21] to achieve anonymous authentication. P-SPAN [4] suffers from huge computation costs and communication overhead in querying responding because 1) the drivers have to not only send a parking query, but also retrieves a result from an RSU when entering its coverage area; 2) the RSUs have to verify and upload queries to the cloud, and update a local Bloom filter; 3) the cloud has to verify queries, and then generate and forward results to each RSU. Both SAVP [5] and PrivAV [6] focus on valet parking in autonomous driving era, which makes them a little different from our work. SAVP [5] has a similar system model without blockchain. It designs three zero-knowledge proofs of knowledge to defend against the double-reservation attack. PrivAV [6] utilized non-interactive zero-knowledge-proofs for identity privacy. All the computations of generating proofs and verifying proofs incur extra computational costs and communication overhead. ASAP [7] adopted a hashmap to store and look for parking spots, but it did not consider the Sybil attack. PEPS [8] leveraged the decentralized anonymous credentials to achieve anonymous authentication in a blockchain network without a trusted entity. However, this method cannot track malicious users. PriParkRec [9] is also a decentralized parking system, which incurs too much computational burden for using zero-knowledge proof, re-randomized public-key encryption, oblivious pseudo-random function, and private set intersection.

H. Implementation on Smartphone and Server

We instantiate Mnemosyne^2 using a PC server as CS and an Android smartphone as a driver, two Android virtual machines as testing devices. Specially, we use Geth (Go-Ethereum Client) as the primary tool for Ethereum network environment establishing. We instantiate four SPSPs on the PC server using VMware Virtual Machine and Geth, each node has a signer account with authority to seal a block. We use Spring Boot projects at IDEA for CS. We create an application on an Android smartphone and use Gradle to introduce the Web3J library to interact with the consortium blockchain. We use the javax.crypto library to implement the cryptographic primitives. We store the data generated during communication in a database using SQLite. The screenshots of the application are shown in Fig. 8. A driver registers to the CS and CB. After logging into the

parking system, the driver interacts with one of the SPSPS to complete anonymous authentication (Fig. 8(a)). By inputting the parking lot identity and query parameter, the authenticated driver can query the previously matched parking lot. The locations of returned results are marked on the map (Fig. 8(b)) and one matched parking lot is reserved (Fig. 8(c)).

VIII. Discussions

A. Assumption of Secure Repetitive Parking

To validate our assumption of the secure repetitive parking, we initiated an online ballot on the RapidWorkers [60]: "Do you think it is necessary to protect repetitive query from the service provider when using a smart parking app? Repetitive query refers to a cruising driver's need for the same parking lot that was assigned to them before. We assume that the driver forgot the exact location of the parking lot and did not record the parking history locally for privacy concerns." We upload the detailed information on https://github.com/UbiPLab/DecPark. We provided \$0.03 for answering the three questions as an incentive. From May 18, 2022 to August 21, 2022, we collected 861 answers, among which 625 (72.6%) chose "Yes", indicating that most workers agree with our assumption.

B. CS in Blockchain

The CS may look contradictory to our decentralized framework. However, it is responsible for generating parameters and keys for S k NN. Once the SPS is running, the CS stays offline until a new driver comes to register. There is also a centralized or trusted third server in some blockchain-based vehicular schemes, such as certificate authority in EBCPA [61], location prover in B-Ride [62], and task administrator and container administrator in BloCkEd [63].

C. Parking Lot Identity

We assume that the drivers have the identity id_i of the target parking lot for two reasons. First, the integer-based identity, e.g., a three-digit number, is much easier to remember than a specific location either by memory or jot down on a sun visor. Second, it is perfectly compatible with the S k NN range query processing.

Meanwhile, directly querying the PLM about the location of loc_i is not the best solution in our scenario. The reason is threefold. First, we cannot ask the drivers to send id_i in plaintext to PLM. Second, even if id_i is simply encrypted via S k NN without using the range technique we proposed, the calculated token will be the same when id_i is queried multiple times. Third, asking the PLM to respond to drivers' queries cannot guarantee correctness, completeness, or truthfulness since the PLM holds all the parking lot information private. This is why we use the blockchain and smart contract to match parking queries and parking lots. Although Private Information Retrieval (PIR) is a powerful technique for oblivious queries, still it incurs both prohibitively high computational cost and communications overhead compared to S k NN.

D. Parking Lot Availability

With a low probability, if the parking lot is not working for the current period but worked normally in the last period (e.g., the parking lot is not open because of the flood or fire), then the PoT would be unconvincing to drivers. However, the availability of parking lots is beyond the scope of this paper. Still we provide some solutions to such an issue. First, it is better to take preventive measures to protect parking lots from flood and fire [64]. Some safety precaution includes incorporating a drainage system and adding curbing [65], and adopting a smoke & fire early warning system [66]. Second, to reduce negative impacts of a similar incident, we can deploy a wireless sensor network or video surveillance to detect fire or flood. If some predefined conditions are satisfied (e.g., 122 degrees Fahrenheit in winter, water level reaches 20 centimeters), the PL will inform its PLM of "num=0" to indicate a status of "out of service". The PLM will send a special transaction to update the index tree and set the number of PL to 0. Afterwards, the SPSP will not dispatch this PL to future drivers.

E. Driver's Key Pair

Many existing work resorts to a trusted server to register users and distribute keys or credentials [6], [7], [33], [34]. Most of our previous works follow this assumption as well. However, storing all the sensitive keys and credentials on a server makes it a single point of failure and a target for adversaries, let alone the maintenance costs. On the other hand, it is possible to eliminate the need for a trusted server and shift the burden from the server to distributed users [20], [27], [67]. We also assume that the drivers are semi-honest. Given the above conditions, we have come to allow the drivers to generate a key pair and then anonymously authenticate the possession of a valid key pair to the SPSP.

F. Smart Contract

Smart contract can request some information of blocks by using some built-in global variables and functions. For instance, requesting the block number of a block via block.number and requesting the hash value of a block via blockhash(unit blockNumber). However, smart contract cannot access the information of transactions in a block. Requesting such information usually requires leveraging specific tools, such as Web3j and Web3.js [68].

IX. Conclusion

In this work, we first identify the problem of secure repetition and full verifiability in smart parking scenarios. To solve the problems, we have designed Mnemosyne ^{2}, i.e., a repetitive, oblivious, and unlinkable Sk NN scheme based on blockchain. With Mnemosyne^2, drivers can securely requests a previously matched parking lot and verify the matching results. We have formally defined and analyzed the security and privacy of Mnemosyne^2, including the capability to withstand four security attacks. We implement a prototype based on Ethereum.

Extensive experimental results show that Mnemosyne[2] exhibits good practicability and an improvement over existing schemes.

REFERENCES

[1] "US VIO vehicle registration statistics - hedges & company," 2024. [Online]. Available: https://hedgescompany.com/automotive-market-research-statistics/auto-mailing-lists-and-marketing

[2] "NYC, LA, San Francisco drivers spend up to 107 hours a year looking for parking," 2018. [Online]. Available: https://www.thedrive.com/news/25557/nyc-la-san-francisco-drivers-spend-up-to-107-hours-a-year-looking-for-parking

[3] "Global smart parking market report 2022: Market will reach 11.2 billion by 2027 from $4.1 Billion in 2021, Growing at a CAGR of $18.2%," PR Newswire, 2022. [Online]. Available: https://www.prnewswire.com/news-releases/global-smart-parking-market-report-2022-market-will-reach-11-2-billion-by-2027-from-4-1-billion-in-2021--growing-at-a-cagr-of-18-2-301540552.html

[4] J. Ni, K. Zhang, Y. Yu, X. Lin, and X. Shen, "Privacy-preserving smart parking navigation supporting efficient driving guidance retrieval," *IEEE Trans. Veh. Technol.*, vol. 67, no. 7, pp. 6504–6517, Jul. 2018.

[5] C. Huang, R. Lu, X. Lin, and X. Shen, "Secure automated valet parking: A privacy-preserving reservation scheme for autonomous vehicles," *IEEE Trans. Veh. Technol.*, vol. 67, no. 11, pp. 11169–11180, Nov. 2018.

[6] J. Ni, X. Lin, and X. Shen, "Toward privacy-preserving valet parking in autonomous driving era," *IEEE Trans. Veh. Technol.*, vol. 68, no. 3, pp. 2893–2905, Mar. 2019.

[7] L. Zhu, M. Li, Z. Zhang, and Z. Qin, "ASAP: An anonymous smart-parking and payment scheme in vehicular networks," *IEEE Trans. Dependable Secure Comput.*, vol. 17, no. 4, pp. 703–715, Jul./Aug. 2020.

[8] L. Wang, X. Lin, E. Zima, and C. Ma, "Towards Airbnb-like privacy-enhanced private parking spot sharing based on blockchain," *IEEE Trans. Veh. Technol.*, vol. 69, no. 3, pp. 2411–2423, Mar. 2020.

[9] Z. Li, M. Alazab, S. Garg, and M. S. Hossain, "PriParkRec: Privacy-preserving decentralized parking recommendation service," *IEEE Trans. Veh. Technol.*, vol. 70, no. 5, pp. 4037–4050, May 2021.

[10] "Blockchain's role in revolutionalizing city parking," 2019. [Online]. Available: https://www.allerin.com/blog/blockchains-role-in-revolutionalizing-city-parking

[11] M. Zhou et al., "PPTA: A location privacy-preserving and flexible task assignment service for spatial crowdsourcing," *Comput. Netw.*, vol. 224, 2023, Art. no. 109600.

[12] Y. Zheng et al., "SecDR: Enabling secure, efficient, and accurate data recovery for mobile crowdsensing," *IEEE Trans. Dependable Secure Comput.*, vol. 21, no. 2, pp. 789–803, Mar./Apr. 2024.

[13] D. Frassinelli, S. Park, and S. Nürnberger, "I know where you parked last summer : Automated reverse engineering and privacy analysis of modern cars," in *Proc. IEEE 41st Symp. Secur. Privacy*, San Francisco, USA, 2020, pp. 1401–1415.

[14] 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.*, Chicago, USA, 2014, pp. 664–675.

[15] X. Lei, A. X. Liu, R. Li, and G.-H. Tu, "SecEQP: A secure and efficient scheme for SkNN query problem over encrypted geodata on cloud," in *Proc. 35th IEEE Int. Conf. Data Eng.*, Macao, China, 2019, pp. 662–673.

[16] N. Cui, X. Yang, B. Wang, J. Li, and G. Wang, "SVkNN: Efficient secure and verifiable k-nearest neighbor query on the cloud platform," in *Proc. IEEE 36th Int. Conf. Data Eng.*, Dallas, USA, 2020, pp. 253–264.

[17] R. Li and A. X. Liu, "Adaptively secure conjunctive query processing over encrypted data for cloud computing," in *Proc. IEEE 33rd Int. Conf. Data Eng.*, San Diego, USA, 2017, pp. 697–708.

[18] V. Buterin, "On public and private blockchains," 2015. [Online]. Avaialble: https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains

[19] M. Li, Y. Chen, C. Lal, M. Conti, M. Alazab, and D. Hu, "Eunomia: Anonymous and secure vehicular digital forensics based on blockchain," *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 1, pp. 225–241, Jan./Feb. 2023.

[20] R. Yang, M. H. Au, Q. Xu, and Z. Yu, "Decentralized blacklistable anonymous credentials with reputation," in *Proc. 23rd Australas. Conf. Inf. Secur. Privacy*, Wollongong, Australia, 2018, pp. 720–738.

[21] D. Pointcheval and O. Sanders, "Short randomizable signatures," in *Proc. Cryptographers' Track RSA Conf.*, San Francisco, USA, 2016, pp. 111–126.

[22] C. Dong, L. Chen, and Z. Wen, "When private set intersection meets Big Data: An efficient and scalable protocol," in *Proc. 20th ACM Conf. Comput. Commun. Secur.*, Berlin, Germany, 2013, pp. 789–800.

[23] B. Libert and D. Vergnaud, "Multi-use unidirectional proxy re-signatures," in *Proc. 15th ACM Conf. Comput. Commun. Secur.*, Alexandria, USA, 2008, pp. 511–520.

[24] M. E. Andrés, N. E. Bordenabe, K. Chatzikokolakis, and C. Palamidessi, "Geo-indistinguishability: Differential privacy for location-based systems," in *Proc. 20th ACM Conf. Comput. Commun. Secur.*, Berlin, Germany, 2013, pp. 901–914.

[25] B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher, "Cuckoo filter: Practically better than bloom," in *Proc. 10th ACM Int. Conf. Emerg. Netw. Experiments Technol.*, Sydney, Australia, 2014, pp. 75–88.

[26] D. Chaum, "Blind signatures for untraceable payments," in *Proc. 3rd Adv. Cryptology*, Santa Barbara, USA, 1983, pp. 199–203.

[27] C. Garman, M. Green, and I. Miers, "Decentralized anonymous credentials," in *Proc. 21st Netw. Distrib. System Secur. Symp.*, San Diego, USA, 2014, pp. 1–15.

[28] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Trans. Inf. Theory*, vol. 31, no. 4, pp. 469–472, Jul. 1985.

[29] A. X. Liu and F. Chen, "Collaborative enforcement of firewall policies in virtual private networks," in *Proc. 27th ACM Symp. Princ. Distrib. Comput.*, Toronto, Canada, 2008, pp. 95–104.

[30] B. H. Bloom, "Space/time tradeoffs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[31] M. Li, M. Zhang, J. Gao, C. Lal, M. Conti, and M. Alazab, "Repetitive, oblivious, and unlinkable SkNN over encrypted-and-updated data on cloud," in *Proc. 24th Int. Conf. Inf. Commun. Secur.*, Canterbury, UK, 2022, pp. 261–280.

[32] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *Proc. IEEE 37th Symp. Secur. Privacy*, San Jose, USA, 2016, pp. 839–858.

[33] 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.

[34] M. Li, L. Zhu, Z. Zhang, C. Lal, M. Conti, and M. Alazab, "User-defined privacy-preserving traffic monitoring against n-by-1 jamming attack," *IEEE/ACM Trans. Netw.*, vol. 30, no. 5, pp. 2060–2073, Oct. 2022.

[35] D. He, J. Chen, and R. Zhang, "An efficient identity-based blind signature scheme without bilinear pairings," *Comput. Elect. Eng.*, vol. 37, no. 4, pp. 444–450, 2011.

[36] D. He, Y. Zhang, and J. Chen, "Cryptanalysis and improvement of an anonymous authentication protocol for wireless access networks," *Wirel. Pers. Commun.*, vol. 74, pp. 229–243, 2014.

[37] D. He, N. Kumar, K. -K. R. Choo, and W. Wu, "Efficient hierarchical identity-based signature with batch verification for automatic dependent surveillance-broadcast system," *IEEE Trans. Inf. Forensics Secur.*, vol. 12, no. 2, pp. 454–464, Feb. 2017.

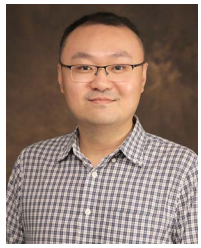
[38] F. Poutinsev, "Unfair search engine ranking results," Honest Pros and Cons (HPC), 2021. [Online]. Available: https://honestproscons.com/unfair-search-engine-ranking-results

[39] P. P. Tsang and V. K. Wei, "Short linkable ring signatures for e-voting, e-cash and attestation," in *Proc. 1st Int. Conf. Inf. Secur. Pract. Experience*, Singapore, 2005, pp. 48–60.

[40] R. Canetti, "Towards realizing random oracles: Hash functions that hide all partial information," in *Proc. 17th Annu. Int. Cryptol. Conf.*, Santa Barbara, USA, 1997, pp. 455–469.

[41] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," *SIAM J. Comput.*, vol. 18, no. 1, pp. 186–208, 1989.

[42] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Proc. 3th Annu. Int. Cryptol. Conf.*, 1986, pp. 186–194.
[43] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2009. [Online]. Available: https://bitcoin.org/bitcoin.pdf
[44] R. Li, A. Liu, A. L. Wang, and B. Bruhadeshwar, "Fast range query processing with strong privacy protection for cloud computing," in *Proc. 40th Int. Conf. Very Large Data Bases*, Hangzhou, China, 2014, pp. 1953–1964.
[45] M. Szydlo, "Merkle tree traversal in log space and time," in *Proc. 10th Int. Conf. Theory Appl. Cryptographic Techn.*, Interlaken, Switzerland, 2004, pp. 541–554.
[46] R. Cramer, I. Damgård, and B. Schoenmakers, "Proofs of partial knowledge and simplified design of witness hiding protocols," in *Proc. 14th Annu. Int. Cryptol. Conf.*, 1994, pp. 174–187.
[47] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," in *Proc. 13th ACM Comput. Commun. Secur. Conf.*, Alexandria, USA, 2006, pp. 79–88.
[48] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*, 3rd Ed. Boca Raton, FL, USA: CRC, 2021.
[49] R. Chatterjee et al., "Compact ring signatures from learning with errors," in *Proc. 41st Annu. Int. Cryptol. Conf.*, Virtual, 2021, pp. 282–312.
[50] F. Garillot, Y. Kondi, P. Mohassel, and V. Nikolaenko, "Threshold Schnorr with stateless deterministic signing from standard assumptions," in *Proc. 41st Annu. Int. Cryptol. Conf.*, 2021, pp. 127–156.
[51] L. Beringer, A. Petcher, K. Q. Ye, and A. W. Appel, "Verified correctness and security of OpenSSL HMAC," in *Proc. 24th USENIX Secur. Symp.*, Washington, DC, USA, 2015, pp. 207–221.
[52] P. Gupta and N. McKeown, "Algorithms for packet classification," *IEEE Netw.*, vol. 15, no. 2, pp. 24–32, Mar./Apr. 2001.
[53] F. McKeen et al., "Innovative instructions and software model for isolated execution," in *Proc. 2nd Int. Workshop Hardware Architectural Support Secur. Privacy*, 2013.
[54] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, "Innovative technology for CPU based attestation and sealing," in *Proc. 2nd Int. Workshop Hardware Architectural Support Secur. Privacy*, 2013, pp. 1–7.
[55] Intel, "Intel Software Guard Extensions," 2024. [Online]. Available: https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/get-started.html
[56] Ethereum, "Layer 2 testnets," 2024. [Online]. Available: https://ethereum.org/en/developers/docs/networks
[57] Arbitrum Docs, "Gas price floor," 2024. [Online]. Available: https://docs.arbitrum.io/arbos/gas#gas-price-floor
[58] R. Neiheiser, G. Inácio, L. Rech, C. Montez, M. Matos, and L. Rodrigues, "Practical limitations of ethereum's layer-2," *IEEE Access*, vol. 11, pp. 8651–8662, 2023.
[59] Etretrum, "Layer 2," 2024. [Online]. Available: https://ethereum.org/en/layer-2
[60] RapidWorkers. 2024. [Online]. Available: https://www.rapidworkers.com
[61] C. Lin, X. Huang, and D. He, "EBCPA: Efficient blockchain-based conditional privacy-preserving authentication for VANETs," *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 3, pp. 1818–1832, May/Jun. 2023.
[62] M. Baza, N. Lasla, M. Mahmoud, G. Srivastava, and M. Abdallah, "B-Ride: Ride sharing with privacy-preservation, trust and fair payment atop public blockchain," *IEEE Trans. Netw. Sci. Eng.*, vol. 8, no. 2, pp. 1214–1229, Second Quarter 2021.
[63] G. S. Aujla, A. Singh, M. Singh, S. Sharma, N. Kumar, and K.-K. R. Choo, "BloCkEd: Blockchain-based secure data processing framework in edge envisioned V2X environment," *IEEE Trans. Veh. Technol.*, vol. 69, no. 6, pp. 5850–5863, Jun. 2020.
[64] Dryzone, "6 tips to prevent fire and flood in your garage," 2022. [Online]. Available: https://www.dryzoneinc.net/blog/2022/december/6-tips-to-prevent-fire-and-flood-in-your-garage
[65] J. Topa, "How to prevent & fix your property's flooding parking lot," 2019. [Online]. Available: https://www.heliconusa.com/blog-1/2019/02/21/how-to-prevent-fix-your-propertys-flooding-parking-lot
[66] Parking Network, "Reducing the risk of car park fires: Introducing S.A.F.E. by highlight parking systems Ltd.," 2023. [Online]. Available: https://www.parking.net/parking-news/highlight-parking-systems-ltd/reducing-the-risk-of-car-park-fires
[67] M. H. Au, A. Kapadia, and W. Susilo, "BLACR: TTP-free blacklistable anonymous credentials with reputation," in *Proc. 19th Annu. Netw. Distrib. System Secur. Symp.*, San Diego, USA, 2012, pp. 1–17.
[68] Ethereum, "Block and transaction properties," 2024. [Online]. Available: https://docs.soliditylang.org/en/v0.8.23/units-and-global-variables.html#block-and-transaction-properties

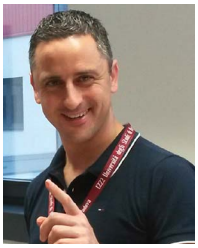


Meng Li (Senior Member, IEEE) received the PhD degree in computer science and technology from the School of Computer Science and Technology, Beijing Institute of Technology (BIT), China, in 2019. He is an associate professor and dean assistant with the School of Computer Science and Information Engineering, Hefei University of Technology (HFUT), China. He is also a postdoc 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). He was sponsored by ERCIM 'Alain Bensoussan' Fellowship Programme (from October 1, 2020 to March 31, 2021) to conduct postdoc research supervised by Prof. Fabio Martinelli at CNR, Italy. He was sponsored by China Scholarship Council (CSC) (from September 1, 2017 to August 31, 2018) for joint Ph.D. study supervised by Prof. Xiaodong Lin (IEEE fellow) with the Broadband Communications Research (BBCR) Lab, University of Waterloo and Wilfrid Laurier University, Canada. His research interests include data security, privacy preservation, applied cryptography, blockchain, TEE, and Internet of Vehicles. In this area, he has published 76 papers in international peer-reviewed journals and conferences, including *IEEE Transactions on Information Forensics and Security*, *IEEE Transactions on Dependable and Secure Computing*, *IEEE Transactions on Mobile Computing*, *IEEE/ACM Transactions on Networking*, *IEEE Transactions on Knowledge and Data Engineering*, *ACM Transactions on Database Systems*, *IEEE Transactions on Services Computing*, *IEEE Transactions on Smart Grid*, *IEEE Transactions on Industrial Informatics*, *IEEE Transactions on Vehicular Technology*, *IEEE Transactions on Network and Service Management*, *IEEE Transactions on Network Science and Engineering*, *IEEE Transactions on Green Communications and Networking*, COMST, ISSTA, MobiCom, ICICS, SecureComm, TrustCom, ICC, and IPCCC. He is a Senior Member of CIE, CIC, and CCF. He is an associate editor for *IEEE Transactions on Information Forensics and Security*, *IEEE Transactions on Network and Service Management*, and *IEEE Internet of Things Journal*.

Mingwei Zhang received the BE degree from the Jiangsu University of Science and Technology, in 2021. He is currently working toward the MS degree with the School of Computer Science and Information Engineering, Hefei University of Technology. His research interests include data security, privacy preservation, applied cryptography, blockchain, TEE, and Internet of Vehicles.

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 500 journal and conference papers in these areas.

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.



Mauro Conti (Fellow, IEEE) received the PhD degree from the Sapienza University of Rome, Italy, in 2009. He is full professor with the University of Padua, Italy. He is also affiliated with TU Delft and University of Washington, Seattle. After his Ph.D., he was a postdoc researcher with Vrije Universiteit Amsterdam, The Netherlands. In 2011 he joined as assistant professor with the University of Padua, where he became associate professor in 2015, and full professor in 2018. He has been 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 includes 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 editor-in-chief for *IEEE Transactions on Information Forensics and Security*, area editor-in-chief for *IEEE Communications Surveys & Tutorials*, and has been associate editor for several journals, including *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 a senior member of the ACM, and fellow of the Young Academy of Europe.

Mamoun Alazab (Senior Member, IEEE) received the PhD degree in computer science from the Federation University of Australia, School of Science, Information Technology and Engineering. He is an associate professor with the College of Engineering, IT and Environment, Charles Darwin University, Australia. He is a cybersecurity researcher and practitioner with industry and academic experience. His research is multidisciplinary that focuses on cybersecurity and digital forensics of computer systems with a focus on cybercrime detection and prevention including cyber terrorism and cyber warfare. He has more than 150 research papers. He delivered many invited and keynote speeches, 24 events in 2019 alone. He convened and chaired more than 50 conferences and workshops. He works closely with government and industry on many projects, including Northern Territory (NT) Department of Information and Corporate Services, IBM, Trend Micro, the Australian Federal Police (AFP), the Australian Communications and Media Authority (ACMA), Westpac, United Nations Office on Drugs and Crime (UNODC), and the Attorney General's Department. He is the founding chair of the IEEE Northern Territory (NT) Subsection.