Why Privacy-Preserving Protocols Are Sometimes Not Enough: A Case Study of the Brisbane Toll Collection Infrastructure

Amirhossein Adavoudi Jolfaei
University of Luxembourg
Luxembourg
amirhossein.adavoudi@uni.lu

Andy Rupp
University of Luxembourg and KASTEL SRL
Luxembourg and Germany
andy.rupp@uni.lu

Stefan Schiffner
Berufliche Hochschule Hamburg (BHH)
Germany
Stefan@inf-bhh.de

Thomas Engel
University of Luxembourg
Luxembourg
thomas.engel@uni.lu

ABSTRACT

The use of Electronic Toll Collection (ETC) systems is on the rise, as these systems have a significant impact on reducing operational costs. Toll service providers (TSPs) access various information, including drivers' IDs and monthly toll fees, to bill drivers. While this is legitimate, such information could be misused for other purposes violating drivers' privacy, most prominent, to infer drivers' movement patterns. To this end, privacy-preserving ETC (PPETC) schemes have been designed to minimize the amount of information leaked while still allowing drivers to be charged.

We demonstrate that merely applying such PPETC schemes to current ETC infrastructures may not ensure privacy. This is due to the (inevitable) minimal information leakage, such as monthly toll fees, which can potentially result in a privacy breach when combined with additional background information, such as road maps and statistical data. To show this, we provide a counterexample using the case study of Brisbane's ETC system. We present two attacks: the first, being a variant of the presence disclosure attack, tries to disclose the toll stations visited by a driver during a billing period as well as the frequency of visits. The second, being a stronger attack, aims to discover cycles of toll stations (e.g., the ones passed during a commute from home to work and back) and their frequencies.

We evaluate the success rates of our attacks using real parameters and statistics from Brisbane's ETC system. In one scenario, the success rate of our toll station disclosure attack can be as high as 94%. This scenario affects about 61% of drivers. In the same scenario, our cycle disclosure attack can achieve a success rate of 51%. It is remarkable that these high success rates can be achieved by only using minimal information as input, which is, e.g., available to a driver's payment service provider or bank, and by following very simple attack strategies without exploiting optimizations. As a further contribution, we analyze how the choice of various parameters, such as the set of toll rates, the number of toll stations, and the billing period length, impact a driver's privacy level regarding our attacks.

Proceedings on Privacy Enhancing Technologies 2024(1), 232–257

https://doi.org/10.56553/popets-2024-0014

KEYWORDS

Electronic Toll Collection, Privacy, Subset Sum Problem

1 INTRODUCTION

For the time period 2019 till 2030, it is predicted that the global ETC market will grow at a compound annual rate of 8.3 percent, reaching about 18.5 billion U.S. dollars by 2030 [6]. Also, in 2019, the EU issued a directive to make ETC systems in Europe fully interoperable [14]. Thus, a careful analysis of privacy concerns arising from such an important technology is crucial.

Road-infrastructure vs. autonomous-device based ETC. We can distinguish two important (post-payment) ETC technologies. In *road-infrastructure based ETC*, on-board units (OBUs) mounted inside a vehicle interact with road-side units (RSUs) to determine tolls. In deployed systems, typically, OBUs simply send over encrypted user IDs when passing an RSU, which are then used by the toll service provider (TSP) to update the user's balance. In (academic) proposals for privacy-preserving ETC systems, more complex cryptographic protocols are executed by OBUs and RSUs. Based on the user's final balance at the end of a billing period, the TSP issues an invoice to the user. In *autonomous-device based ETC*, road-infrastructure devices such as RSUs are not needed. Instead, the OBU determines the toll for road segments autonomously using location and time information from GPS. The OBU and the TSP interact in order to compute the final balance of a user. In this paper, we primarily focus on (privacy-preserving) road-infrastructure based ETC. However, as our attacks are fairly generic and only require minimal information as input (see later), we expect that they are also applicable to disclose a driver's (cycles of) road segments in an autonomous-device based ETC.

Privacy concerns. TSPs, often private companies, store sensitive information to charge drivers. This data typically includes drivers' identities and home addresses, wallet balances being the total toll drivers owe to the TSP by the end of a billing period, and the locations and times drivers pass toll stations. Naturally, privacy concerns arise due to the fact that such sensitive user data is in the hands of (private) TSPs. The work [32] provides an overview of the resulting privacy issues. For instance, a TSP could misuse the data to infer drivers' movement patterns and places of interest. Also, third parties, including law enforcement agencies and insurance companies, may try to persuade the TSP to get access to the data

for prosecution or commercial purposes. This demonstrates that the collected data could be misused for purposes beyond billing drivers.

PPETC leaking minimal information. To address these issues, several privacy-preserving ETC (PPETC) schemes have been proposed so far, e.g., [2, 10, 15, 31, 34] (cf. Section 10 for an overview). Such schemes aim to disclose only the minimal information to the TSP necessary to charge drivers but protect the anonymity and unlinkability of a driver's transactions. Typically, this includes the drivers' identities, home addresses, and final wallet balances, which are needed to issue invoices at the end of a billing period, as well as a database of anonymous and unlinkable transaction records (each containing at least location and time information[1]) which are provided by the RSUs.

Research question: real-world privacy provided by PPETC. As privacy protection in current PPETC schemes does not mean that there is no information leakage at all but certain minimal information (required to ensure the core functionality) is still leaked, it is interesting to see if this information is already sufficient to violate privacy in a practical scenario. Besides the information leaked by the protocols, the deployment of a PPETC scheme in a real-world scenario also fixes certain parameters and provides additional background information which is all relevant to a driver's privacy: the pricing scheme, number of toll stations, road infrastructure information, statistical data about driver behavior, etc. To the best of our knowledge, the impact of such information on the "real-world privacy" of a PPETC scheme has not been thoroughly analyzed yet. A driver's privacy certainly depends on the complexity of the subset sum problem (SSP) [28] which, in our case, is concerned with finding toll prices that add up to a driver's wallet balance (monthly total toll). By solving the SSP, an adversary can learn the toll stations a driver passed during a billing period [2, 15, 34]. Although the general SSP is NP-complete, there are variants that can be solved in polynomial time [33]. Moreover, we can expect that in a real-world scenario, the parameters relevant for its time complexity (number of toll prices) will usually be small values.

Our Contribution. In this paper, we provide evidence that a driver's privacy may not be preserved in general when a PPETC system is used to replace an existing ETC in practice. We do so by focusing on Brisbane's ETC infrastructure as a case study, using its real parameter settings (set of toll rates, number of toll stations), real statistical data (distribution of wallet balances, maximal number of visited toll stations), and real background information (road infrastructure).[2]

Our attacker is weaker than the one typically considered by PPETC schemes, the latter being the TSP colluding with all RSUs, as it exploits less information: it does not require the transaction records provided by RSUs as input. In fact, the minimal information used by our adversary is, e.g., available to the payment service provider (e.g., Apple Pay [1]) or bank of a TSP's customer. This makes our attacks generic and broadly applicable.

[1]If a transaction between an OBU and an RSU takes place, the RSU is at least aware of its own location and the current time.

[2]Note that we did not conduct a privacy analysis dedicated to Brisbane's current ETC protocols. However, our generic attacks also apply to their current system. Stronger and more efficient attacks, taking their actual protocols into account, might be possible.

We present two generic attacks. The first one, we call "toll station disclosure attack" (TSD), is a variant of the presence disclosure attack [36]. It tries to discover the toll stations visited by a driver during a billing period as well as the number of visits. The second attack, called "cycle disclosure attack" (CD), aims to identify the cycles made by a driver during a billing period as well as their frequencies. A cycle starts and ends at the driver's home location and passes through one or more toll stations. In comparison to the first attack, the adversary discloses more information (e.g., the order in which toll stations are visited) since the cycles already include the visited toll stations. The first attack is based on solving a variant of the SSP. The second attack exploits the output of the first one.

We evaluate our attacks based on information and statistics from the Brisbane ETC setting. This evaluation shows that our attacks achieve considerably high success rates. More precisely, the first attack achieves a success rate of 94% and the second one a success rate of 51% for 61% of the drivers. Although these success rates are already pretty high and answer our research question, we explore certain heuristics to further improve them by assessing the plausibility of solutions to the underlying SSP or removing implausible solutions. Note that exploiting transaction records, in addition, may (only) lead to better success rates or stronger attacks (but also to a stronger attacker model), which we leave as future work. As a further contribution, we study how certain parameter choices (toll rates, number of toll stations, length of billing period) affect the success rate of our attacks and give some recommendations to transportation system engineers based on our findings.

2 BACKGROUND

Here, we define various terms needed for our study.

Transaction: The transaction e is a tuple consisting of a toll location (loc), toll price (τ) and time (t), i.e., $e = \langle loc, \tau, t \rangle$. The set $T = \{e_1, e_2, \ldots, e_o\}$ consists of all transactions made by all drivers within a billing period. All the transactions are *anonymous*.

Billing period: The billing period is a system parameter fixed by the TSP and denoted by θ. It consists of a starting time and an end time. All transactions in this interval belong to the set of transactions associated with this billing period.

Toll station's identities: We define the set of identities of all toll stations/points as $S = \{s_1, s_2, \ldots, s_l\}$.

Toll price: This is a parameter fixed by the TSP. We define the set of toll prices as $P = \{\tau_1, \tau_2, \ldots, \tau_l\}$, where $\tau_j \in \mathbb{D}$ is a fixed toll fee that is assigned to toll station s_j. This study focuses on static toll pricing schemes with fixed prices, in contrast to dynamic toll pricing schemes where toll fees vary based on factors like the day of the week or time of day.

Graph: The map of a city, including toll stations and toll roads, is represented by a directed graph as $G = (V, E)$. In graph G, V represents the toll stations, i.e., $V = S$. Each edge represents a toll road. The graph may include other information available on the city's map, such as the distance between two toll stations.

Set of frequencies: The frequency f_j is the number of times a driver visits the toll station s_j in a billing period. We define the set $F = \{f_1, f_2, \ldots, f_l\}$, where $f_j \in \mathbb{N}_0$ is the frequency corresponding

to toll station s_j. Note that $\mathbb{N}_0$ is a set of non-negative integers, and $f_j = 0$ means the driver has not visited the toll station s_j.

Drivers' identity: The set $ID = \{i_1, i_2, \ldots, i_n\}$ denotes a subset of unique identities (e.g., passport number) of TSP customers. This information is needed to associate a monthly toll fee (wallet) with its corresponding driver's identity and home address in order to issue an invoice at the end of a billing period. For simplicity, we assume that each customer coincides with the driver who uses the TSP's services. Hence, each identity $i \in ID$ can be linked with both a driver and a customer. Assuming that each driver only uses one fixed vehicle, each i can be associated with the driver's vehicle. This assumption is valid if an OBU inside a vehicle is fixed and cannot be attached to another vehicle. Hence, there is a one-to-one relationship between the identity i and a driver/customer and its corresponding vehicle.

Drivers' home address: A subset of home addresses, associated with ID, is denoted as $H = \{(i_1, h_1), (i_2, h_2), \ldots, (i_n, h_n)\}$, where $i_j \in ID$ and h_j in each tuple corresponds to a subscribed driver's identity and his home address, respectively. A driver provides the TSP and payment provider with their home address to which invoices are sent.

Cycle: In this study, we consider a cycle in graph G as a circuit, which is a non-empty trail in which the first and last vertices are equal [42]. A cycle includes one or more toll roads on which at least one toll station is located. The cycle may also encompass one or more roads without any toll stations.

Subset sum problem: The SSP is a well-known NP-complete problem, which is defined as follows [28]. We consider the set $A = \{a_j : 1 \leq j \leq k, a_j \in \mathbb{N}_0\}$ and the value $M \in \mathbb{N}_0$, i.e., a non-negative integer. The aim is to find x_is such that $a_1 \cdot x_1 + a_2 \cdot x_2 + \cdots + a_k \cdot x_k = M, x_j \in \mathbb{N}_0$.

Linear diophantine equation: A linear diophantine equation is a linear equation whose solution is restricted to be integers [35]. Each variable in the equation has at most a degree of one. Equation 1 represents such equations.

$$b_1 \cdot x_1 + b_2 \cdot x_2 + \cdots + b_n \cdot x_n = g, x_i \in \mathbb{Z} \tag{1}$$

Solving the equation falls into integer optimization problems where the variables take integer values [39]. The SSP can be interpreted as solving Equation 1, where $x_i, g \in \mathbb{N}_0$.

Wallet balance: The wallet balance is the total toll fee based on which the TSP issues an invoice for a driver, and it should be paid by the end of the billing period. Note that each wallet is associated with a driver's identity so that the TSP can charge the driver. The driver's wallet w is the summation of the toll prices of all visited toll stations by the driver within a billing period. Given w, the following linear diophantine equation holds:

$$w = \tau_1 \cdot f_1 + \tau_2 \cdot f_2 + \cdots + \tau_l \cdot f_l \tag{2}$$

τ_j is the toll price of the toll station s_j, and f_j is the corresponding frequency. We define a subset of wallet balances of drivers, associated with ID, as $W = \{(i_1, w_1), (i_2, w_2), \ldots, (i_n, w_n)\}$, where $w_j \in \mathbb{N}_0$ is associated with a driver with identity $i_j \in ID$. For $i_j \in ID$, there is exactly one tuple with the first component i_j in W.

Trace: Intuitively, a trace demonstrates the *history* of the *toll stations* (s_j) visited by a driver and their corresponding *frequency* (f_j) in a *billing period*. We present a trace as the set of tuples denoted as the set $trace = \{(s_1, f_1), (s_2, f_2), \ldots, (s_l, f_l)\}$, where f_i is the frequency associated with the toll station s_i. Given $trace$, one or more of the $f_j \in trace$ may equal zero, meaning the driver has not visited the corresponding toll station s_j. Having defined the trace, we define the correct trace and the plausible trace.

Correct trace: The correct trace is *truly* made by a driver in an ETC system. It corresponds to the toll stations visited by a driver and the frequency of the visited toll stations during the considered billing period.

Plausible trace: The plausible trace of a driver with wallet w is the $trace = \{(s_1, f_1), (s_2, f_2), \ldots, (s_l, f_l)\}$, where the toll prices associated with the toll stations inside the tuples and corresponding frequencies satisfy Equation 2. A plausible trace is not necessarily the correct trace but a *candidate* for being the correct trace. The adversary uses an attack to obtain the set of plausible traces denoted by $plaus\_traces = \{trace_1, trace_2, \ldots, trace_d\}$.

Success rate: We define the success rate as the probability that an attacker selects the correct trace of a driver uniform at random from all plausible traces. Note that the uniform selection of the correct trace from the set can be considered a *baseline strategy*, as the adversary makes no distinction among the traces (they all have an equal probability of being a correct trace). However, the attack could exploit various strategies in which the adversary distinguishes among the traces in the set (each having different probabilities of being a correct trace). In this case, different strategies might result in distinct corresponding success rates comparable to the success rate when the attack employs the baseline strategy. Through this comparison, we can measure the extent to which a strategy performs better or worse than the baseline strategy.

The attack's effectiveness: Two factors mainly impact the attack's effectiveness and, accordingly, affect a driver's privacy: (1) **The number of plausible traces:** In the case of many plausible traces, privacy is preserved as the success rate becomes very small. (2) **The computational complexity:** If it is computationally infeasible to find plausible traces, privacy is preserved. The tables of notations and acronyms are shown in Appendix A.

3 THREAT MODEL

In this work, we consider a passive adversary with access to a subset ID of driver IDs, subsets H and W of home addresses and wallet balances, respectively, associated with those drivers, as well as the set P of all toll prices and the graph G of the ETC system. This knowledge of the adversary is denoted by the set $K = \{ID, P, G, W, H\}$ in the following. Note that this is a small amount of required knowledge, which, in the real world, would be, e.g., already available to payment service providers (e.g., Google Pay [17]) or banks that TSP customers use to make toll payments to the TSP: A payment service provider can certainly identify toll payments to a TSP in its records. These records also contain the corresponding wallet balances. Furthermore, it knows the IDs and home addresses of its customers associated with these payment records, as this information is usually needed to set up an account and issue valid account statements. Finally, the graph, i.e., the map of the toll station infrastructure as well as the pricing, can be considered public information available from the internet (e.g., the website of the TSP).

Certainly, K is a subset of the information available to a TSP. In fact, also the designers of PPETC schemes such as P4TC [15] consider a stronger adversary being the TSP colluding with all RSUs. Such an adversary would additionally receive all *anonymous* transaction records (consisting of locations, timestamps, and fares, but no IDs) from the RSUs. By means of our attacks, we show that even our weaker adversary not obtaining those transaction records can have a significant success rate. Exploiting more information may only lead to stronger attacks or higher success rates.[3]

In the following, we consider adversaries aiming to achieve the following two goals:

Toll station disclosure (TSD) goal. This goal is a variant of the *presence disclosure* goal, defined as "to find out if a given user or a set of users are present at some place(s)" [36]. The TSD goal has two subgoals: firstly, to learn the toll stations visited by a driver and, secondly, to determine the frequency with which the driver visited each toll station within a given billing period. The frequency of visited toll stations could reveal a driver's point of interest/s (POI) [16, 21, 38], e.g., supermarkets, restaurants, tourist spots, and hotels [30, 44]. Note that this goal is equivalent to finding the correct trace defined in Section 2 and denoted as $trace = \{(s_1, f_1), (s_2, f_2), \ldots, (s_l, f_l)\}$.

Cycle disclosure (CD) goal. The intuition behind this goal is to discover regular activities of drivers, represented by cycles [3, 4]. A cycle starts from a driver's home, passes through one or more toll stations, and returns to the home. Besides learning cycles, another subgoal is to determine the frequency of each individual cycle made in the billing period. Regarding this goal, a driver's trace is defined as $trace = \{(c_1, f_1), (c_2, f_2), \ldots, (c_y, f_y)\}$, where the c_j are different cycles the driver made in a billing period. Note CD is a stronger goal than TSD, as it aims to disclose a driver's full trajectories instead of only the visited toll stations.

4 THE TSD ATTACK

This section presents the TSD attack to achieve the TSD goal. The pseudo-code and details of the attack are discussed in Appendix B. We, finally, theoretically compute the attack's success rate. We present the attack as follows.

4.1 Procedure of the attack

The adversary ($\mathcal{A}$) uses an attack based on solving the SSP to obtain the plausible traces of a driver. The attack uses as its input the set $M = \{ID, P, G, W\}$. To find the correct trace of a driver with the wallet w, the adversary needs to obtain the driver's set of plausible traces where the correct trace is located. A plausible trace is denoted as $trace = \{(s_1, f_1), (s_2, f_2), \ldots, (s_l, f_l)\}$; hence, to create a plausible trace, it needs the set of toll stations, i.e., $S = \{s_1, s_2, \ldots, s_l\}$ and the set of corresponding frequencies, i.e., $F = \{f_1, f_2, \ldots, f_l\}$. Note that the set S is concluded from graph $G = (V, E)$, where $S = V$. To obtain the set $F = \{f_1, f_2, \ldots, f_l\}$, it needs to solve the SSP, for which it creates the following linear diophantine equation and solves it via DOcplex (i.e., IBM Decision Optimization CPLEX Modeling [12]), which uses the depth-first search as the default algorithm [22]. We remind that the SSP can be interpreted as a linear diophantine equation.

$$w = \tau_1 \cdot x_1 + \tau_2 \cdot x_2 + \cdots + \tau_l \cdot x_l, x_j \in \mathbb{N}_0, x_j \leq \lceil w/min(P) \rceil \quad (3)$$

The equation holds as explained in the wallet's definition in Section 2. Each x_j, in Equation 3, represents the frequency f_j that the driver visited the toll station s_j. A solution of Equation 3 results in the set F. Note that the $f_j \neq 0$ in the set F means that the corresponding toll station s_j is visited at least once. Then, $\mathcal{A}$ verifies if the visited toll stations in the set S are connected. For instance, if two toll stations are visited without visiting the intermediate toll station (assuming that this is the only connection between the two toll stations), the solution will be discarded. If connectivity holds, $\mathcal{A}$ creates the corresponding plausible trace $trace$ using the sets S and F. The algorithm for checking the connectivity in a graph is fully discussed in Appendix C. It should be highlighted that one solution of Equation 3 leads to a plausible trace; hence, since the equation may have more than one solution, it results in a set of plausible traces as $plaus\_traces = \{trace_1, trace_2, \ldots, trace_d\}$. The correct trace is among the plausible traces. The details of the attack and the corresponding pseudo-code are provided in Appendix B.

Success rate: The probability, namely SR, that $\mathcal{A}$ uniformly guesses the correct trace, i.e., $trace_j$ from the set of plausible traces $plaus\_traces = \{trace_1, trace_2, \ldots, trace_d\}$ is computed as:

$$\mathbf{SR} = 1 \,/\, |plaus\_traces| = 1 \,/\, d \quad (4)$$

The details and pseudo-code of the algorithm computing the success rate are discussed in Appendix D.

4.2 Heuristic-based approaches

We explained in Section 2 (see success rate) that the adversary uses a baseline strategy to guess *uniformly* at random the correct trace from the plausible traces. Here, we offer several heuristics that the TSD attack can utilize to guess *non-uniformly* the correct trace. The presented heuristics are based on drivers' behavior.

The first heuristic. The presented heuristic is based on drivers' behavior, i.e., they tend to visit a restricted number of toll stations within a given billing period. For example, the toll roads in Brisbane are mainly used for activities, such as taking a holiday/getaway, going to the airport, and social activities [41]. This behavior implies that drivers exhibit a constrained toll station usage pattern tied to their regular activities. Concerning this heuristic, $\mathcal{A}$ considers the maximum number ($threshold$) of toll stations a driver visits in a billing period. The threshold can be determined through statistical data [41]. Considering a set of plausible traces associated with a driver, the heuristic assigns the probability of zero to plausible traces in which the number of toll stations exceeds the threshold and assigns the same probabilities to the rest of the plausible traces. Finally, $\mathcal{A}$ ignores the traces with a probability of zero (this is why we consider this strategy as non-uniformly) and selects a trace uniformly from the remaining ones.

The second heuristic. This heuristic, similar to the first one, is based on drivers' behavior, i.e., visiting a limited number of toll stations, with the difference that the heuristic assigns a certain

[3] We stress that we refrain from exploiting additional information because (i) our counter-example is stronger when considering a *weaker* adversary, and (ii) it is hard to obtain the necessary transaction records to evaluate stronger attacks under real-world conditions.

probability to each plausible trace in a set of plausible traces. The probability can be computed using statistical data, which describes the distribution of the number of visited toll stations by drivers in an ETC system (in a billing period). For instance, $\mathcal{A}$ may assign a relatively low probability to plausible traces that involve a large number of toll stations. For example, in the case study of Brisbane, it would be less likely for a driver to visit all nine toll points in the city. On the other hand, $\mathcal{A}$ can assign a relatively high probability to plausible traces that involve a relatively small number of toll points. For example, employees who regularly commute from Yatala to Ipswich would likely pass through only a limited number of toll points on their route (see the map in Figure 1). Finally, $\mathcal{A}$ selects a trace with the highest assigned probability. The details and pseudo-code are discussed in Appendix F.

The third heuristic. This heuristic exploits the regularity in drivers' behavior by utilizing the yearly historical information of wallet balances. The heuristic considers a driver and all their corresponding yearly wallet balances, denoted as $w_j, 1 \leq j \leq 12$, where j represents each month of the year. Using the sets of plausible traces corresponding to the w_js, the heuristic creates a set of clusters, each of which includes the potential traces a driver could have made within a year. Each cluster is assigned a probability based on the similarity of the traces within it, indicating the likelihood that those traces are associated with the driver. To measure the similarity among the traces, $\mathcal{A}$ can use a similarity metric such as Euclidean distance. Clusters with higher probabilities contain traces that demonstrate more similarity and are thus more likely to be associated with a driver displaying regularity in their behavior. Finally, $\mathcal{A}$ can select a cluster with the highest probability (see Appendix F for the details and pseudo-code).

5 EVALUATION OF THE TSD ATTACK

In Section 5.1, we introduce the parameter settings used in the evaluation of the attack. In Sections 5.2 and 5.3, we evaluate the TSD attack and the first heuristic, respectively.

5.1 Parameter settings

In this section, we discuss the parameter settings and evaluate the attack based on the metric SR (see Equation 4). To evaluate the attack, we use real parameters and statistics provided by Transurban Queensland, including toll prices, the number of toll stations, the billing period length, and statistical information on the distribution of wallet balances [5, 40, 41]. Transurban is the operator of all toll roads in Brisbane. The toll stations and toll roads are shown on the map in Figure 1, where the yellow circles represent the toll stations in Brisbane. In the following, we discuss the attack input, i.e., the set $M = \{ID, P, G, W\}$, and the other parameters needed to obtain the plausible traces. The parameter settings are shown in Table 1. Our evaluation is performed on Windows Server 2019 Standard (64-bit), with 96.0 GB RAM, with x64-based CPU 3.70 GHz, Intel(R) Xeon(R) E-2288G.

- **Drivers' identities (ID)**: Drivers' identities are not publicly available; however, we can compute the success rate whatever drivers' identities are as a driver's identity has no role in the computation of the correct trace nor of the success rate. The identity is only used for assigning the correct trace to its corresponding driver's identity (see line 15 of the pseudo-code of the TSD attack, in Appendix B).
- **Toll prices (P):** We define the set of toll prices as follows: $P = \{\tau_1 = 4.55, \tau_2 = 2.68, \tau_3 = 2.84, \tau_4 = 1.72, \tau_5 = 4.09, \tau_6 = 5.46, \tau_7 = 5.11, \tau_8 = 5.11, \tau_9 = 3.19\}$. The real toll prices are based on [40] and are in dollars.
- **Toll stations (S):** We define the set of toll stations' identities as follows: $S = \{A, B, C, D, E, F, G, H, I\}$. The toll stations are shown on the map in Figure 1.

Figure 1: The Brisbane map [5].

- **Wallets (W)**: Drivers' wallet balances in Brisbane are not publicly available. However, we have access to statistics from the sources [40, 41] denoting the wallet ranges. These ranges are represented by $[w_l, w_u]$, where w_l and w_u are the lower and upper bounds of the range. According to the statistics, 61.38% of drivers fall into the range [\$0, \$10], while 14.31% fall into [\$10, \$20], and 11.12% are associated with the range [\$20, \$40]. Figure 2 provides a summary of the statistics. Using the ranges, we can generate drivers' *plausible wallet balances*.
  Plausible wallet balance: A driver's plausible wallet balance is a wallet in the range $[w_l, w_u]$ that could potentially be associated with the driver. To generate a driver's plausible wallet balances, we create the below inequality whose solutions are all plausible wallet balances:

$$w_l < \tau_1 \cdot x_1 + \tau_2 \cdot x_2 + \cdots + \tau_9 \cdot x_9 < w_u \tag{5}$$

Note that the idea of the inequality comes from Equation 3. To create the inequality, we use the wallet ranges shown in Figure 2. For example, given the range [\$10, \$20], the corresponding inequality is as follows: $10 < \tau_1 \cdot x_1 + \tau_2 \cdot x_2 + \cdots + \tau_9 \cdot x_9 \leq 20$. Having solved the inequality for each range $[w_l, w_u]$, the number of corresponding plausible wallet balances becomes 93, 721, and 2000, each of which is associated with drivers with the corresponding wallet

Attack input and parameter settings	Num of plausible wallets	Drivers' proportion	*ASR*	*ASR*
$M = \{ID, P, G, 0 < w \leq 10\}, \lvert S\rvert = 9, \theta$ = a month	93	61.38%	94%	51%
$M = \{ID, P, G, 10 < w \leq 20\}, \lvert S\rvert = 9, \theta$ = a month	721	14.31%	54%	11%
$M = \{ID, P, G, 20 < w \leq 40\}, \lvert S\rvert = 9, \theta$ = a month	2000	11.12%	5%	0.28%

Table 1: The *ASR*s of attacks given Brisbane's parameter settings. The column *ASR* (left-sided) corresponds to the TSD attack, and the column *ASR* (right-sided) corresponds to the CD attack.

range. The numbers are shown in Table 1. We further use the generated plausible wallets to compute the average success rate.

- **Upper bound (u)**: We set the upper bound to $\lceil w/min(P) \rceil$, meaning that $x_j, 1 \leq j \leq l$ (see Equation 3) cannot exceed this upper bound.
- **Billing period (θ)**: We consider a billing period of a month as the wallets reported in [40] are based on one month.
- **Number of variables ($|S|$)**: The number of variables in Equation 3, and Inequality 5 equals 9, i.e., the number of toll stations.

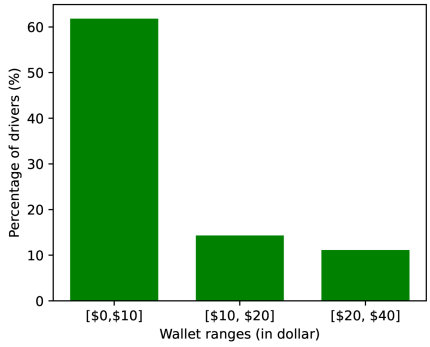


Figure 2: The proportion of drivers across wallet ranges. [40]

5.2 Our evaluation

Having discussed the parameter settings, we proceed to evaluate the TSD attack using the notion of *ASR*.

5.2.1 ***Computation of average success rate.*** Having discussed the parameter settings, we aim to compute the attack's success rate (see Formula 4); however, as said earlier, the adversary does not access drivers' real wallets (W) to run the attack. To tackle the problem, we employ the computed plausible wallet balances discussed earlier and use the notion of *average success rate (ASR)*. The *ASR* metric demonstrates the attack's average success in finding a driver's correct trace concerning all his corresponding potentially plausible wallets. To compute *ASR*, we take the following steps.

- Step 1: Considering drivers associated with the wallet range $[w_l, w_u]$, we compute a driver's plausible wallet balances using Inequality 5. Then, for each plausible wallet balance, we run the TSD attack and accordingly compute its corresponding success rate, i.e., *SR*.
- Step 2: We compute the average of all computed *SR*s corresponding to the plausible wallet balances from Step 1. The resulting value is the attack's *ASR* for finding the correct trace of a driver associated with the range $[w_l, w_u]$. Note that for all drivers within the range $[w_l, w_u]$, the attack's *ASR* has the same value since the range results in the same plausible wallet balances and, accordingly, the same *ASR*. Note that the two steps will be repeated for each of the three ranges shown in Figure 2. The figure shows that approximately 86% (summation of all proportions) pay less than \$40 a month; hence, we are motivated to evaluate the attack's success rate concerning this large proportion of drivers. We do not consider the rest of the drivers (14% of total drivers) with wallet balances above \$40 since the corresponding ASR is negligible.

5.2.2 ***The evaluation results.*** Based on the evaluation, we discuss the distribution of success rates and the corresponding *ASR*s which are shown in Table 1.

Distribution of success rates. To analyze the effectiveness of the TSD attack, we illustrate the distribution of success rates obtained in Step 1. Each box plot in Figure 3 illustrates the distribution of success rates corresponding to all plausible wallet balances within the range of $[w_l, w_u]$. The figure shows that almost all of the success rates (*SR*s) within the range of $[\$0, \$10]$ are 100% (shown in the first box plot). For the range of $[\$10, \$20]$, half of the success rates (*SR*s) are between 50% and 100%, and the other half are between 7% and 50% (shown in the second box plot). Finally, for the range of $[\$20, \$40]$, almost all of the success rates (*SR*s) are below 12% (shown in the third box plot). As a concrete example, we illustrate the distribution of *SR*s in the first box plot in Appendix E.

- The *ASR* for 61.38% of drivers, a large proportion, is roughly 94% which is considerably high. As said, this proportion of drivers is associated with the wallet range of $[\$0, \$10]$.
- The *ASR* for 14.31% of drivers is approximately 54%, yet considerably high. This proportion of drivers is associated with the wallet range of $[\$10, \$20]$.
- The *ASR* for the small proportion (11.12%) of drivers is nearly 5%, which is relatively low.
- The attack achieves these high *ASR*s, i.e., 94% and 54% without utilizing drivers' home addresses and transactions. Furthermore, these *ASR*s are achieved using the baseline strategy without incorporating any heuristics.

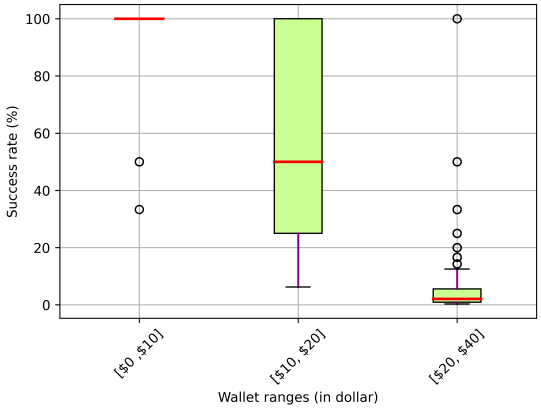


Figure 3: Each box plot shows the distribution of *SR*s across all plausible wallet balances within the range $[w_l, w_u]$.

5.2.3 ***Discussion on the attack's effectiveness.*** As said in Section 2, the attack's effectiveness depends on the number of plausible traces and the complexity of the attack, which hinges on the difficulty of solving Equation 3 (which is *generally* NP-complete). Our evaluation demonstrates that the TSD attack performs effectively as solving the equation is feasible. Figure 4 shows that the distribution of wallet balances significantly impacts the number of plausible traces and the runtime. Figure 4a shows the number of traces for plausible wallet balances below $40, associated with 86% of total drivers. The graph shows that for wallets below $10 (93 red points), the number of plausible traces is one or two, meaning that a large proportion of drivers (61.38%) have very low privacy. The number of traces for wallet balances between $10 and $20 (721 blue points) is between one and sixteen. Drivers' privacy with this range of wallets is also at risk of violation. For balances between $20 and $40 (2000 green points), the number of traces is considerably increasing, i.e., between one and 320. Drivers with balances close to $20 have relatively lower privacy than drivers close to $40. Overall, the graph shows that the balance of wallets noticeably impacts the number of traces and a driver's privacy accordingly.

Figure 4b shows the runtime of the TSD attack, which depends on the difficulty of solving Equation 3. The runtime for the wallets below $10 is between 20 *ms* and 60 *ms*; for wallets between $10 and $20 is between 20 *ms* and 80 *ms*. The runtime for the wallets between $20 and $40 is considerably increasing, from 30 *ms* to 260 *ms*. Although the graph shows an increasing trend, solving Equation 3 is feasible in a short time. This shows that although solving a linear diophantine equation is NP-complete in general, the equation can be solved quickly, given the real settings of Brisbane's ETC system.

5.3 Evaluation of the first heuristic

Due to the page limitation, we evaluate the first heuristic (see Section 4.2) with the parameter settings of Brisbane's ETC system (see Table 6 in Appendix F). We highlight that presenting a more sophisticated and enhanced strategy (to guess the correct trace) would not necessarily strengthen our argument, as our main goal is to provide a counterexample, which we have already achieved. See Appendix F for the details of the heuristic.

For the evaluation, we analyze how the attack's *ASR* and the *average percentage decrease* (*APD*) change with different potential thresholds that the adversary can consider in the case study of Brisbane. The percentage decrease helps us compute the reduction in the number of plausible traces (due to ignoring traces with zero probabilities) as a percentage of the original size of the set of plausible traces. The notion of *APD* is similar to *ASR* (see Section 5.2.1), which is the average of all percentage decreases corresponding to a driver's all plausible wallet balances within the range $[w_l, w_u]$. Table 2 shows that the metrics show an upward trend as the threshold decreases. This trend is particularly noticeable in larger wallet ranges ([$20, $40], [$40, $60]), indicating a contribution to the improvement in ASR compared to the lower ASRs before applying the heuristic (see "without heuristic" column). However, the upward trend is less noticeable for smaller wallet ranges ([$0, $10], [$10, $20]); nevertheless, the corresponding ASRs are already high even without applying the heuristic, i.e., 94% and 51.16% (see "without heuristic" column).

6 THE CD ATTACK

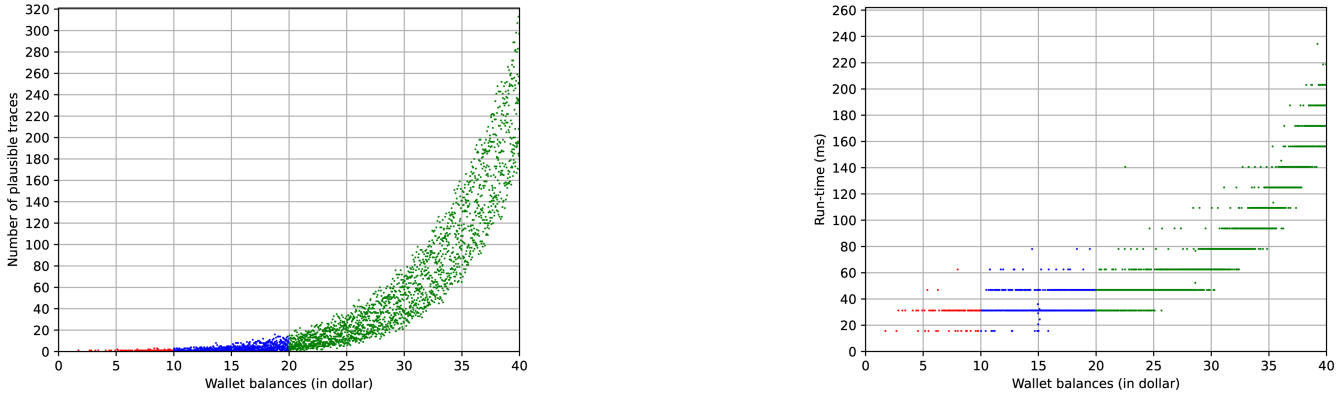
As defined in Section 3, the CD goal is to find the *cycles* and corresponding frequencies a driver made in a billing period. The adversary can attempt to find the cycles where the summation of the price of individual cycles leads to a driver's wallet balance by solving an SSP: The price of a cycle is the summation of the toll prices corresponding to the toll stations included in the cycle. However, this fails; for details, see Appendix I. The adversary would need to solve a linear equation with numerous variables, which is computationally infeasible for a sufficiently large number of toll stations. We introduce the CD attack, which exempts the adversary from solving the SSP. We theoretically elaborate on the algorithm's success rate and then evaluate it with the real parameter settings of Brisbane's ETC system.

6.1 Procedure of the attack

In this section, we present the CD attack. See Appendix J for pseudocode and details and Appendix K for an example.

Basic idea. The adversary exploits the set $K = \{ID, P, G, W, H\}$ to achieve the CD goal. Compared to the TSD goal, the adversary, in this attack, exploits additional information, i.e., drivers' home addresses. The attack's core idea is that $\mathcal{A}$ exploits the visited toll stations and their corresponding frequencies, which are already obtained from the TSD attack and denoted as $correct\_trace = \{(s_1, f_1), (s_2, f_2), \ldots, (s_l, f_l)\}$. The visited toll stations are the building blocks of the cycles made by a driver. Obviously, the toll stations that have not been visited do not contribute to the formation of cycles. In the following, we will first explain the algorithm, namely "find_cycle_algo" for finding cycles utilized by the CD attack. We will then outline the CD attack in three steps.

Algorithm find_cycle_algo. We present an algorithm for finding cycles. The algorithm takes a set of different toll stations, a home location, a graph, and a strategy as the inputs and returns the cycle passing through all the toll points and the home location (if



(a) Number of plausible traces per wallet balance

(b) Run-time (ms) per wallet balance

Figure 4: The number of plausible traces and runtime are two parameters impacting the attack's effectiveness. The value of these parameters increases significantly as the balance of the wallets gets larger.

Metric	Thresholds used in the first heuristic						Without heuristic	Wallet range
	3	4	5	6	7	8		
APD	0%	0%	0%	0%	0%	0%	0%	$(0 < w \leq 10)$
	24.1%	3.95%	0.08%	0%	0%	0%	0%	$(10 < w \leq 20)$
	81.33%	47.47%	16.03%	2.7%	0.18%	0%	0%	$(20 < w \leq 40)$
	97.21%	84.01%	53.7%	20.77%	3.98%	0.27%	0%	$(40 < w \leq 60)$
ASR	94%	94%	94%	94%	94%	94%	94%	$(0 < w \leq 10)$
	65.84%	52.45%	51.17%	51.16%	51.16%	51.16%	51.16%	$(10 < w \leq 20)$
	22.05%	7.07%	4.99%	4.72%	4.69%	4.69%	4.69%	$(20 < w \leq 40)$
	5.2%	0.85%	0.3%	0.18%	0.16%	0.15%	0.15%	$(40 < w \leq 60)$

Table 2: The metrics *APD* and *ASR* indicate the impact of the first heuristic on the reduction of plausible traces and the average success rate, respectively, during a one-month billing period. Each row corresponds to a wallet range and thresholds j, where $3 \leq j \leq 8$. As the threshold decreases, the metrics show an upward trend, especially for wallet ranges that include higher wallet balances.

any). The algorithm can use various strategies to obtain the cycles, such as the shortest distance and shortest time [8]. A driver may use the shortest-distance strategy, taking the shortest possible route to their destination. The algorithm and its pseudo-code are shown in Appendix H.

Step 1: Create partitions. $\mathcal{A}$ uniformly selects a plausible trace as a candidate for the correct trace, namely *correct_trace*, via the TSD attack. Given the $correct\_trace = \{(s_1, f_1), (s_2, f_2), \ldots, (s_l, f_l)\}$, $\mathcal{A}$ first creates a multiset, where the visited toll station s_j is repeated f_j times. It then computes all possible partitions by dividing all visited toll points in the multiset into segments, each containing different visited toll stations. The point is that the toll stations in each segment, along with the driver's home location, could potentially form a cycle that a driver has made. A segment must satisfy two conditions: (1) no toll station should be repeated within a segment, and (2) the order of toll stations in a segment does not matter. All the segments in a partition could potentially correspond to the cycles a driver has made. The driver's correct trace (i.e., cycles and their corresponding frequencies) is associated with only one of the partitions. To compute the partitions, $\mathcal{A}$ uses a partition function explained in the following example.

Example: Given a driver's correct trace $\{(s_1, 1), (s_2, 2), (s_3, 1)\}$, the partition function computes the multiset $visited\_tolls = \{s_1, s_2, s_2, s_3\}$. Then, the function divides the visited toll points in the multiset into six different partitions regarding the two mentioned conditions. The portions are 1: $\{\{s_1, s_2\}, \{s_2, s_3\}\}$, 2: $\{\{s_1, s_2, s_3\}, \{s_2\}\}$, 3: $\{\{s_1, s_2\}, \{s_2\}, \{s_3\}\}$, 4: $\{\{s_1\}, \{s_2\}, \{s_2, s_3\}\}$, 5:$\{\{s_1, s_3\}, \{s_2\}, \{s_2\}\}$, 6: $\{\{s_1\}, \{s_2\}, \{s_2\}, \{s_3\}\}$. The toll stations in each segment, along with a driver's home location (h), lead to a cycle (if any) that the driver could have made. The cycle is found by the algorithm $find\_cycle\_algo$.

Step 2: Transform each partition to a plausible trace. In this step, $\mathcal{A}$ transforms all the segments in each partition to their corresponding cycle (if any), using the aforementioned algorithm $find\_cycle\_algo$. Hence, each partition will be transformed into a

plausible trace, including the cycles and their corresponding frequencies $\{(c_1, f_1), (c_2, f_2), \ldots, (c_y, f_y)\}$. The plausible trace, then, will be stored in a set of plausible traces, namely *plaus_cycle_traces*. Note that there may be a partition where the toll points in a segment cannot lead to a cycle due to their connectivity, and hence, the partition cannot lead to a plausible trace. We emphasize that each plausible trace within the set of plausible traces is linked to one of the partitions obtained in Step 1.

Example: In the example provided in Step 1, concerning partition one, segments $\{s_1, s_2\}$ and $\{s_2, s_3\}$ could lead to corresponding cycles, such as hs_1s_2h and hs_2s_3h, thereby forming the plausible trace $\{(hs_1s_2h, 1), (hs_2s_3h, 1)\}$.

Step 3: Selection of the correct trace. Finally, $\mathcal{A}$ uniformly guesses a plausible trace as the correct trace from the set of plausible traces, i.e., *plaus_cycle_traces*. The success rate of $\mathcal{A}$ to guess correctly is computed as follows:

6.1.1 ***Computation of the success rate.*** To obtain the success rate, we need to compute the probability of selecting the correct trace in Step 3. This requires $\mathcal{A}$ to make two consecutive correct guesses, as outlined below: (1) In Step 1, $\mathcal{A}$ has to guess the correct trace, namely $trace_{d'}$ among the plausible traces $plaus\_traces = \{trace_1, \ldots, trace_d\}$ obtained by the TSD attack, where d' denotes an arbitrary but fixed plausible trace. The probability of the correct guess is denoted and computed as $p[1\text{st}] = \frac{1}{d}$.

(2) Given $trace_{d'}$, using the CD attack, $\mathcal{A}$ obtains the corresponding set of plausible traces, including the cycles and corresponding frequencies denoted by $plaus\_cycle\_traces = \{trace_1, \ldots, trace_z\}$ (see Step 2). Then, $\mathcal{A}$ uniformly selects $trace_{z'}$ as a correct trace from this set with the condition that its first guess is correct, where z' in $trace_{z'}$ denotes an arbitrary but fixed plausible trace. The probability of $trace_{z'}$ being a correct trace is denoted by $p[2\text{nd} \mid 1\text{st}] = \frac{1}{z}$. Now, we use Bayes' theorem [23] to compute the probability of the second guess, i.e., $p[2\text{nd}]$, being a correct trace.

$$p[2\text{nd}] = \frac{p([1\text{st}]) \times p([2\text{nd} \mid 1\text{st}])}{p([1\text{st} \mid 2\text{nd}])} \tag{6}$$

We previously computed the terms $p[1\text{st}]$ and $p[2\text{nd} \mid 1\text{st}]$, which are equal to $\frac{1}{d}$ and $\frac{1}{z}$ respectively. The term $p([1\text{st} \mid 2\text{nd}])$ in Formula 6 represents the probability of the first guess being correct, given that the second guess is correct, which is equal to one. This is because the correctness of the second guess depends on the correctness of the first guess as a prerequisite. Substituting the values of these terms into Formula 6 yields the success rate, i.e., the probability of the second guess being correct.

$$SR_{d'} = p[2\text{nd}] = \frac{1}{d} \times \frac{1}{z} \tag{7}$$

The index d' in $SR_{d'}$ denotes that the success rate is computed with respect to the $trace_{d'}$ as a candidate for the correct trace in Step 1. Since, in this step, each $trace_j$ in the set of plausible traces has a chance of being a correct trace, we compute SR_j with respect to each individual $trace_j$ using Formula 7. Note that the value of z depends on $trace_{d'}$. The final SR is obtained by taking the average over all SR_js (shown in Formula 7), as all of the $trace_j$ have the same probability of being a correct trace.

$$SR = \frac{1}{d} \times \sum_{j=1}^{d} SR_j \tag{8}$$

Formula 8 shows that the success rate of the CD attack (SR) correlates with the success rate of the TSD attack, i.e., $p[1\text{st}] = \frac{1}{d}$.

Discussion on the attack's effectiveness. As said in Section 2, the attack's effectiveness depends on two main factors. (1) **The number of plausible traces:** If in Formula 8, d and z are quite large, i.e., the corresponding sets of plausible traces contain a quite large number of plausible traces, SR becomes negligible, preserving a driver's privacy. (2) **Computational complexity:** The complexity of the CD attack mainly depends on the complexity of the TSD attack (discussed in Section 4.1) and depends on the complexity of the functions used for creating the partitions and cycles. The details are discussed in Appendix J.

7 EVALUATION OF THE CD ATTACK

Here, we discuss the parameter settings used in the CD attack and evaluate it using the success rate discussed in 6.1.1.

Parameter settings. We employ the same real parameter settings of Brisbane's ETC system discussed in Section 5.1. The CD attack requires the set $M = \{ID, P, G, W\}$ and also the set H to achieve the CD goal. Concerning the set H, we do not access the home locations of drivers registered with the Brisbane ETC system, as this information is not publicly available. Consequently, the algorithm for finding the cycles cannot find the cycles since it relies on H; additionally, the algorithm for finding the cycles requires the strategy for finding a cycle as drivers use different strategies for selecting a path. However, we do not access such information; nonetheless, we can evaluate the CD attack as follows.

Our evaluation. We evaluate the CD attack using the notion of *ASR* explained in Section 5.2.1 with the difference that, here, *ASR* is computed based on *SR* in Formula 8. According to the formula, *SR* depends on d and z, where z depends on the size of the set of plausible traces (*plaus_cycle_traces*), including plausible traces. To obtain the set, H and the strategy is needed to find the cycles (see Step 2 of Section 6.1). As said earlier, we do not access such information; nevertheless, we compute *SR* using an upper bound for the size of the set *plaus_cycle_traces*, i.e., z, via the inequality $|plaus\_cycle\_traces| \leq$ number of all partitions. The inequality holds because, as said in Step 2 of Section 6.1, each partition may/may not lead to a corresponding plausible trace. Hence, we employ the total number of partitions to compute an upper bound for $|plaus\_cycle\_traces|$, i.e., z, and, accordingly, to compute *SR*. Given *SR*, we compute *ASR* for each individual wallet range $[w_l, w_u]$.

The evaluation results. We discuss the distribution of success rates and *ASR*s shown in Table 1.

Distribution of success rates. To investigate how successfully the CD attack performs, we demonstrate the distribution of success rates across all plausible wallet balances within each range $[w_l, w_u]$ in Figure 5.

Each box plot in Figure 5 illustrates the distribution of success rates corresponding to all plausible wallet balances within the range

of $[w_l, w_u]$. The figure shows that half of the success rates (SRs) corresponding to the range of [\$0, \$10] are nearly 50%, while the other half fall between 10% and 50% (shown in the first box plot). For the range of [\$10, \$20], half of the success rates (SRs) fall between 5.50% and 35% (shown in the second box plot). Finally, for the range of [\$20, \$40], a few of the success rates (SRs) are below 10%, and the rest are close to zero (shown in the third box plot).

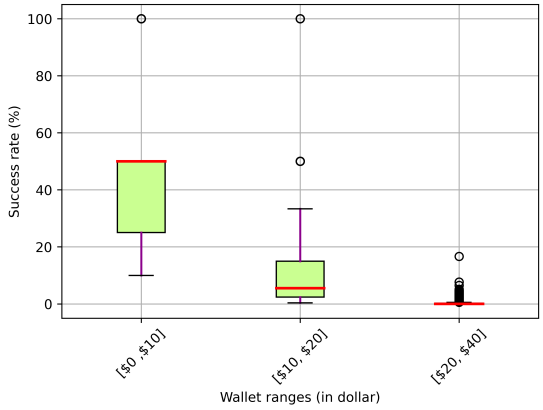


Figure 5: Each box plot shows the distribution of SRs across all plausible wallet balances within the range $[w_l, w_u]$.

- Regarding the CD attack, the *ASR* for approximately 61.38% of drivers is around 51%, which is relatively high. However, it is considerably lower than the *ASR* of 94% achieved by the TSD attack.
- The *ASR* for approximately 14.31% of drivers is relatively low, i.e., 11%, which is lower than the *ASR* of 54% achieved by the TSD attack.
- The *ASR* for the small proportion of drivers 11.12% is approximately 0.28%, which is negligible compared to the *ASR* of 5% in the TSD attack.
- The comparison above demonstrates that the CD attack has a lower *ASR* in comparison to the TSD attack. However, the former attack provides the adversary with more information, such as the driver's cycles, whereas the latter attack only discloses the visited toll stations.

8 IMPACT OF PARAMETER SETTINGS ON PRIVACY

We discuss parameters impacting the TSD attack's success rate and a driver's privacy accordingly. Then, we evaluate the success rate for various parameter settings. To this end, we consider the Brisbane case study with different parameter settings and complexities. This approach offers useful insights to determine the privacy level of a driver under different settings. For instance, we explore alternative toll price ranges rather than Brisbane's, creating diverse complexities. Appendix G provides the details of our experiments.

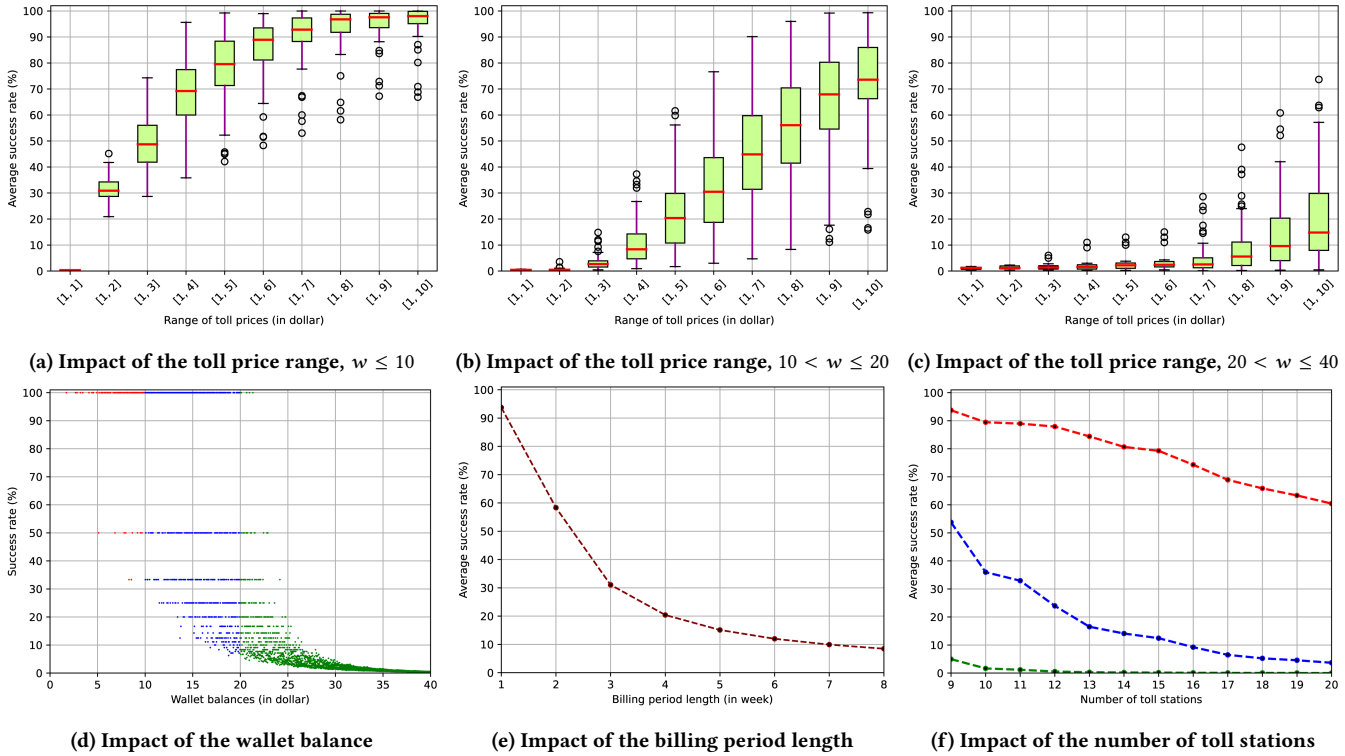
Parameters impacting privacy. The attack's success rate depends on the number of plausible traces, which, in turn, is influenced by the parameters in Equation 3. These parameters are as follows: toll prices (P), wallet balance (w), length of the billing period (θ), and the number of toll stations ($|S|$).

Analysis of different parameter settings on privacy. We investigate the impact of different settings of each parameter on the attack's success rate and, consequently, a driver's privacy. To accomplish this, we consider the Brisbane case study and the corresponding parameter settings outlined in Section 5.1. We create scenarios with varying levels of complexity for each parameter, allowing us to assess the attack's success rate in different settings. To this end, while keeping certain parameters fixed, we vary the settings of a specific parameter, such as toll prices, to examine its impact on privacy. Appendix G and Table 7 in the appendix provide further details on our experiments and the parameter settings. Figure 6 shows the impact of settings of different parameters on the success rate. The red, blue, and green points/graphs concern the wallet ranges [\$0, \$10], [\$10, \$20], and [\$20, \$40], respectively. The results are summarized as follows:

- Parameter P: Each of the toll price ranges $[1, j]$, $1 \leq j \leq 10$ on the x-axis of Figures 6a, 6b, and 6c indicate the span from which toll prices are chosen and assigned to the respective toll stations in Brisbane. Figures 6a, 6b, and 6c demonstrate that the *ASR* increases as the toll price range gets larger. Now, for comparing the box plots, we consider their median. In Figure 6a, the *ASR* increases from zero to almost 96%, in Figure 6b from zero to almost 73%, and in Figure 6c, from zero to about 15%. This concludes that a driver's privacy is more at risk, given a larger toll price range. This manifests itself more evidently for wallets with low balances (see Figure 6a). The figures show that when the assigned toll prices are equal (selected from the range $[1, 1]$), the *ASR* approaches zero, thereby preserving a driver's privacy (see Appendix G for the details of our experiment).
- Parameter w: The wallet's balance highly impacts the number of plausible traces and a driver's privacy accordingly. Overall, small wallet balances lead to higher *ASR*s. Figure 6d shows that drivers (above %75 of total drivers) with wallet balances below \$35 are more at risk of a privacy violation than those with wallets between \$35 and \$40.
- Parameter θ: Figure 6e shows that a short billing period leads to a high *ASR*. Even for a relatively long billing period of eight months (two months), the attack has *ASR* of about 10%, which is considerable.
- Parameter $|S|$: Figure 6f illustrates that the attack's *ASR* remains high for a low number of toll stations. While the *ASR* decreases as the number of toll stations increases, it is still significantly high, approximately 60%, for 20 toll stations and the wallet range [\$0, \$10] (as shown by the red graph). This poses a potential risk to a driver's privacy.
- It should be noted that the overall impact of the parameters on privacy also holds regarding the CD attack since the success rates of the attacks are correlated (see Formula 8).

9 DISCUSSION

We discuss the limitations and future work, and then we give several recommendations concerning PPETC schemes.



(a) Impact of the toll price range, $w \leq 10$ **(b) Impact of the toll price range, $10 < w \leq 20$** **(c) Impact of the toll price range, $20 < w \leq 40$**

(d) Impact of the wallet balance **(e) Impact of the billing period length** **(f) Impact of the number of toll stations**

Figure 6: The figures demonstrate how different settings of a parameter, including toll prices, wallet balances, billing period length, and the number of toll stations, impact the attack's *ASR* or success rate and, accordingly, a driver's privacy.

9.1 Limitations and Future work

- Our TSD attack utilizes a subset of the information available to the TSP in order to achieve the TSD goal. This attack can serve as a baseline for other attacks that aim to achieve the same goal but utilize more information than the set M. For instance, these attacks could incorporate drivers' home addresses and transactions, which are not utilized in our attack. The CD attack uses the sets M and H to achieve the CD goal. Similarly, this attack can be used as a baseline for attacks that exploit additional information, such as transactions. In future work, we can analyze how leveraging such additional information would impact a driver's privacy compared to our attacks.
- It is said in [36], if the adversary has already found some user locations where he visited, completing the user's locations can be done more easily. This is achievable if the adversary accesses the user's *mobility profile*. The mobility profile represents how probable it is for a specific user to move from one location to another in a specific period [36]. The adversary, e.g., could use drivers' identities to find drivers' mobility profiles. Knowing such information would help the adversary complete the locations the driver passed. In future work, given that the adversary has obtained a driver's mobility profile, we can investigate to what extent the adversary can complete the driver's trajectories.
- As future work, we raise the following research question: "To what extent is an adversary successful in converting the cycles obtained by the CD attack to a chain of trips?" A trip made by a driver starts from a source (at time t_s) and ends at a destination (at time t_e), and the driver stays at the destination for a while. Each trip takes place for a main purpose: work, shopping, and education. We are interested in finding the trip constituents of a cycle. The adversary, e.g., could benefit from contextual information to guess the trips made by a driver [11]. For example, given that a cycle passes through one toll station close to a shopping center, the adversary can conclude that the driver could have made a trip to the shopping center with a certain probability. By inferring the trips, the adversary learns more information about a driver's behavior.
- For evaluating the TSD attack, since we did not access drivers' wallet balances, we generated all plausible wallet balances based on wallet ranges provided by statistics. Then, we computed the attack's *ASR* (see Section 5.2.1). Although our evaluation demonstrates a high *ASR*, accessing drivers' wallet balances leads to a more accurate success rate in finding a driver's correct trace.
- This study focuses on static toll pricing, excluding the consideration of PPETC schemes involving dynamic pricing. Dynamic pricing entails toll prices that fluctuate according to time or other parameters. Incorporating dynamic pricing

would negatively affect the efficacy of our attacks, as it introduces additional variables into Equation 3 while solving the SSP. This stands in contrast to a scenario where prices remain constant. In fact, the inclusion of each time-based toll price requires the addition of a corresponding variable within the equation, thereby reducing the success rate.

- In this work, our presented attacks do not employ transactions, including times (stored in the TSP), to achieve the goals. The times are the moments when vehicles pass through toll stations. The reasons for not using times in our attacks are: firstly, based on our threat model, the given "counter-example" is stronger if considering a weaker attacker. Secondly, although there are tracking algorithms [13, 43] for tracking vehicles based on the times and locations of vehicles, these algorithms operate under some conditions, some of which are not held in our case study. The main important condition is that vehicles should report their times and locations *periodically* and with a *short time interval* [20, 43] (e.g., below one second [43]), which does not hold in our study. In our case study, the times corresponding to each vehicle have long time intervals since the distances between two consecutive toll stations are quite long. Hence, in this study, we take advantage of other information for the purpose of tracking, including wallets, toll prices, the city's ETC graph, and home addresses. Finally, to evaluate attacks exploiting timestamps, we would also need realistic traces, including timestamps, for the Brisbane setting. But we do not access real traces, and for generating realistic synthetic traces, we lack the necessary statistics to feed into a simulator we would have to develop. This is why we refrained from exploiting time stamps. As future work, we are interested in how to exploit timestamps as additional information to achieve stronger attacks/higher success rates.
- In Section 1, we categorized PPETC schemes into two different categories: (1) privacy-preserving road-infrastructure based ETC schemes and (2) privacy-preserving autonomous-device based ETC schemes. The focus of this study is on the first category. In future work, we can investigate the privacy of schemes in the second category, such as [2, 31, 34], where the OBU device is used to compute the monthly toll fees with the help of times and locations received by the GPS. We are interested in determining if our attacks can be applied to the schemes in the second category. Since our attacks are fairly generic and only require minimal information as input, we expect that they are also applicable to disclose a driver's road segments in an autonomous-device based ETC.

9.2 Recommendation

To deploy a PPETC system, toll engineers should consider the following recommendations based on our analysis in Section 8: (1) Various parameters impact a driver's privacy, including the range of toll prices (P), wallet balances (w), length of the billing period (θ), and the number of toll stations ($|S|$), which the TSP sets except the wallet balance. A wallet balance depends on the toll prices of toll stations visited by the driver and the frequency of visiting. The parameter θ also indirectly affects a wallet's balance and, accordingly, a driver's privacy. Hence, when deploying PPETC schemes, toll engineers should remember that varying a parameter could influence the other parameters and, accordingly, a driver's privacy. (2) Considering a sufficiently large billing period by the TSP would help reduce the risk of violating a driver's privacy (Figure 6e). In fact, a large billing period would lead to larger wallet balances as drivers would likely visit more toll stations or visit toll stations more frequently within a longer period. Therefore, larger wallet balances provide better privacy for drivers (Figure 6d). It should be noted that this consideration should align with a toll service provider's financial policies. (3) A wide range of toll prices in an ETC system increases the risk of violating a driver's privacy. It is advisable to consider a toll price range that results in a negligible success rate while not conflicting with the TSP's financial policies. For example, if all toll prices associated with the corresponding toll stations are equal, the success rate is close to zero (see Figures 6a, 6b, and 6c). However, such a toll price range should align with a TSP's financial policies.

10 RELATED WORK

In this Section, we discuss the typical information leaked by PPETC schemes. Then, we elaborate on the attacks regarding PPETC systems.

Privacy in ETC schemes: In this section, we discuss the information leaked by PPETC schemes and available to the TSP. The privacy-preserving scheme in [25] stores toll fees and hashes of the locations into the TSP so as to hide them from the server. Apart from this, a small percentage of the locations and toll fees should be revealed to the TSP to detect cheating. In [34], the scheme "VPriv" stores the tagged location-time data at the TSP, which an OBU sends. The driver computes the total toll fee and sends it to the TSP. The notion of privacy in "VPriv" means that the privacy of the current scheme should be the same as the privacy in a scheme in which the TSP stores just location-time data without any identifying information (tags), and the total toll fee should be received from an oracle without executing any protocol. In [26], the TSP stores the OBU's id and a set of signed, encrypted, and committed tuples (location, time, toll fee). This information is transmitted by the OBU at the end of each billing period. Two schemes are presented in [10]: (1) the SPEcTRe spot-record scheme and (2) the SPEcTRe no-record scheme. The first scheme reveals the same data as that in the scheme [34]. The second scheme does not store drivers' private information but can detect cheaters.

Attacks on privacy in ETC: To the best of our knowledge supported by the survey [24], the work [7] is the only relevant study that proposes an attack concerning the post-payment PPETC schemes. In [7], the authors present an attack to find drivers' possible traces in a toy example of an ETC system. A trace is a set of trips a driver has made within a billing period. A trip made by a driver includes anonymous transactions belonging to the driver. The authors assumed a strong adversary that already has access to anonymous trips as its input, which is a strong assumption. Besides the trips, the adversary accesses wallet balances, trip prices, the city map, and contextual information such as the maximal speed of cars. The adversary uses an attack that works based on solving the SSP whose solutions lead to a driver's set of traces where the

correct trace is located. To solve the SSP, [7] utilizes Pisinger's algorithm [33] due to its linear computation time. However, it is important to note that the values corresponding to the set of traces must adhere to specific restrictions. In our approach, we employ DOcplex [12] as the solver, which, by default, has no restrictions and is more convenient and straightforward than Pisinger's algorithm. The attack uses various heuristics based on the connectivity of trips to reduce the number of traces. Unlike our study, the work lacks a detailed and precise threat model, e.g., the information available to an adversary, and its goal is not determined clearly. This work, as opposed to our study, aims to perform an attack by exploiting any ETC data at the adversary's disposal, while our objective is to do an attack using a weaker adversary and using only the minimal ETC data, which makes our attack more applicable and impressive. The main point overlooked in this work is the significance of parameter settings in determining the complexity of the attack, which depends on the hardness of SSP. In contrast, our study examines the impact of various parameters and their settings, including toll price ranges, wallet balances, and number of toll stations, on the difficulty of the SSP problem (see Sections 5.2.3, 8). By analyzing the effect of different parameter settings on the SSP's complexity, we obtain valuable insights into selecting parameter settings that can enhance privacy. The presented attack in [7] is just applied to a toy scenario using the synthetic data within a billing period of 10 days, and the authors do not discuss how their evaluation results can be applied to a real-world scenario. We emphasize that the attack in [7] relies on a very strong assumption (which is not justifiable according to [20, 37] with respect to our setting). Therefore, we cannot regard it as a suitable baseline for the purpose of benchmarking. The comparison between our work and [7] is summarized in the table shown in Appendix L.

In [34], the authors briefly mention an attack based on the SSP and argue that if the billing period is large enough, it is unlikely for an adversary to learn the toll stations a driver visits. However, unlike ours, their work primarily focuses on designing a PPETC scheme and does not provide a detailed attack to evaluate a driver's privacy. Our study demonstrates, as opposed to their perception, that our attack can achieve a significant success rate even for a one-month billing period, which is large enough. Similarly, the work [2] briefly acknowledges the possibility of an attack by solving the SSP in PPETC schemes. However, since their aim is to design a PPETC scheme, they do not present any attacks, unlike ours.

11 CONCLUSION

This research focuses on the issue of driver privacy in post-payment privacy-preserving road-infrastructure based ETC schemes. We offer a counterexample using the case study of Brisbane's ETC system, illustrating that these PPETC schemes are sometimes not sufficient to provide privacy. We present two attacks that employ a subset of all information available to the TSP with real parameter settings of Brisbane's ETC system. The first attack aims to achieve the toll station disclosure goal, and the results show that the attack's *ASR* is considerably high, i.e., 94% affecting 61% of total drivers' privacy. The second attack aims to achieve the cycle disclosure goal, where an adversary learns more information. The evaluation shows that the attack's *ASR* is high, i.e., 51% for 61% of total drivers. These high *ASR*s are achievable considering a weak adversary employing a baseline strategy without exploiting any heuristics. Our attack evaluation shows that solving the SSP (i.e., generally NP-complete), upon which the schemes' privacy is built, is feasible. We then present various parameters impacting a driver's privacy, such as toll prices and wallets, and then analyze how different settings of individual parameters impact a driver's privacy. Finally, we give recommendations to toll engineers when deploying privacy-preserving ETC systems.

ACKNOWLEDGMENTS

Andy Rupp was supported by funding from the topic Engineering Secure Systems (46.23.01 Methods for Engineering Secure Systems and 46.23.03 Engineering Security for Mobility Systems) of the Helmholtz Association (HGF) and by KASTEL Security Research Labs.

REFERENCES

[1] Apple Pay 2023. *Apple Pay.* Retrieved July 1, 2023 from https://www.apple.com/apple-pay/
[2] Josep Balasch, Alfredo Rial, Carmela Troncoso, Bart Preneel, Ingrid Verbauwhede, and Christophe Geuens. 2010. {PrETP}:{Privacy-Preserving} Electronic Toll Pricing. In *19th USENIX Security Symposium (USENIX Security 10).*
[3] Haris Ballis and Loukas Dimitriou. 2020. Optimal synthesis of tours from multi-period origin-destination matrices using elements from graph theory and integer programming. *European Journal of Transport and Infrastructure Research* 20, 4 (2020), 1–21.
[4] Haris Ballis and Loukas Dimitriou. 2020. Revealing personal activities schedules from synthesizing multi-period origin-destination matrices. *Transportation research part B: methodological* 139 (2020), 224–258.
[5] Linkt Brisbane. 2022. *Queensland toll calculator.* Retrieved 2022 from https://www.linkt.com.au/using-toll-roads/toll-calculator/brisbane
[6] Mathilde Carlier. 2021. *Projections for the global electronic toll collection market size between 2019 and 2030.* Retrieved 2022 from https://www.statista.com/statistics/1254629/global-electronic-toll-collection-market-forecast
[7] Xihui Chen, David Fonkwe, and Jun Pang. 2012. Post-hoc user traceability analysis in electronic toll pricing systems. In *Data Privacy Management and Autonomous Spontaneous Security*. Springer, 29–42.
[8] Wilner Ciscal-Terry, Mauro Dell'Amico, Natalia Selini Hadjidimitriou, and Manuel Iori. 2016. An analysis of drivers route choice behaviour using GPS data and optimal alternatives. *Journal of transport geography* 51 (2016), 119–129.
[9] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2022. *Introduction to algorithms.* MIT press.
[10] Jeremy Day, Yizhou Huang, Edward Knapp, and Ian Goldberg. 2011. Spectre: spot-checked private ecash tolling at roadside. In *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society.* 61–68.
[11] Rinku Dewri, Prasad Annadata, Wisam Eltarjaman, and Ramakrishna Thurimella. 2013. Inferring trip destinations from driving habits data. In *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society.* 267–272.
[12] DOcplex 2022. *DOcplex Python Modeling API.* Retrieved May 1, 2022 from https://www.ibm.com/docs/en/icos/12.8.0.0?topic=docplex-python-modeling-api
[13] Karim Emara, Wolfgang Woerndl, and Johann Schlichter. 2013. Vehicle tracking using vehicular network beacons. In *2013 IEEE 14th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM).* IEEE, 1–6.
[14] European Commission. 2019. *Directive of the European Parliament and of the Council on the Interoperability of Electronic Road Toll Systems and Facilitating Crossborder Exchange of Information on the Failure to Pay Road Fees in the Union.* https://eur-lex.europa.eu/eli/dir/2019/520/oj
[15] Valerie Fetzer, Max Hoffmann, Matthias Nagel, Andy Rupp, and Rebecca Schwerdt. 2018. P4TC—Provably-Secure yet Practical Privacy-Preserving Toll Collection. *Cryptology ePrint Archive* (2018).
[16] Fosca Giannotti, Mirco Nanni, Fabio Pinelli, and Dino Pedreschi. 2007. Trajectory pattern mining. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining.* 330–339.
[17] Google Pay 2023. *Google Pay.* Retrieved July 1, 2023 from https://pay.google.com/about/
[18] Vladimir Grebinski and Gregory Kucherov. 1997. Optimal query bounds for reconstructing a Hamiltonian cycle in complete graphs. In *Proceedings of the Fifth Israeli Symposium on Theory of Computing and Systems.* IEEE, 166–173.
[19] Gregory Gutin and Abraham P Punnen. 2006. *The traveling salesman problem and its variations.* Vol. 12. Springer Science & Business Media.

[20] Baik Hoh, Marco Gruteser, Hui Xiong, and Ansaf Alrabady. 2007. Preserving privacy in GPS traces via uncertainty-aware path cloaking. In *Proceedings of the 14th ACM conference on Computer and communications security*. 161–171.
[21] Yongfeng Huo, Bilian Chen, Jing Tang, and Yifeng Zeng. 2021. Privacy-preserving point-of-interest recommendation based on geographical and social influence. *Information Sciences* 543 (2021), 202–218.
[22] IBM. 2022. *Module docplex.cp.parameters.* Retrieved 2023 from https://ibmdecisionoptimization.github.io/docplex-doc/cp/docplex.cp.parameters.py.html#docplex.cp.parameters.CpoParameters.SearchType
[23] Edwin T Jaynes. 2003. *Probability theory: The logic of science.* Cambridge university press.
[24] Amirhossein Adavoudi Jolfaei, Abdelwahab Boualouache, Andy Rupp, Stefan Schiffner, and Thomas Engel. 2023. A Survey on Privacy-Preserving Electronic Toll Collection Schemes for Intelligent Transportation Systems. *IEEE Transactions on Intelligent Transportation Systems* (2023).
[25] Wiebren de Jonge and Bart Jacobs. 2008. Privacy-friendly electronic traffic pricing via commits. In *International Workshop on Formal Aspects in Security and Trust*. Springer, 143–161.
[26] Florian Kerschbaum and Hoon Wei Lim. 2015. Privacy-preserving observation in public spaces. In *European Symposium on Research in Computer Security*. Springer, 81–100.
[27] Donald E Knuth. 2004. *GENERATING ALL PARTITIONS.* Retrieved 2023 from http://www.kcats.org/csci/464/doc/knuth/fascicles/fasc3a.pdf
[28] Jeffrey C Lagarias and Andrew M Odlyzko. 1985. Solving low-density subset sum problems. *Journal of the ACM (JACM)* 32, 1 (1985), 229–246.
[29] Gilbert Laporte. 1992. The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research* 59, 2 (1992), 231–247.
[30] Xin Li, Guandong Xu, Enhong Chen, and Yu Zong. 2015. Learning recency based comparative choice towards point-of-interest recommendation. *Expert Systems with Applications* 42, 9 (2015), 4274–4283.
[31] Sarah Meiklejohn, Keaton Mowery, Stephen Checkoway, and Hovav Shacham. 2011. The Phantom Tollbooth: {Privacy-Preserving} Electronic Toll Collection in the Presence of Driver Collusion. In *20th USENIX Security Symposium (USENIX Security 11)*.
[32] KW Ogden. 2001. Privacy issues in electronic toll collection. *Transportation Research Part C: Emerging Technologies* 9, 2 (2001), 123–134.
[33] David Pisinger. 1999. Linear time algorithms for knapsack problems with bounded weights. *Journal of Algorithms* 33, 1 (1999), 1–14.
[34] Raluca Ada Popa, Hari Balakrishnan, and Andrew J Blumberg. 2009. VPriv: Protecting privacy in location-based vehicular services. (2009).
[35] Tyronza R Richmond and Arunachalam Ravindran. 1974. A generalized euclidean procedure for integer linear programs. *Naval Research Logistics Quarterly* 21, 1 (1974), 125–144.
[36] Reza Shokri, Julien Freudiger, and Jean-Pierre Hubaux. 2010. *A unified framework for location privacy.* Technical Report.
[37] Reza Shokri, George Theodorakopoulos, Jean-Yves Le Boudec, and Jean-Pierre Hubaux. 2011. Quantifying location privacy. In *2011 IEEE symposium on security and privacy*. IEEE, 247–262.
[38] Gang Sun, Shuai Cai, Hongfang Yu, Sabita Maharjan, Victor Chang, Xiaojiang Du, and Mohsen Guizani. 2019. Location privacy preservation for mobile users in location-based services. *IEEE Access* 7 (2019), 87425–87438.
[39] Hamdy A Taha. 2014. *Integer programming: theory, applications, and computations.* Academic Press.
[40] Transport and Public Works Committee. 2018. *Inquiry into the operations of toll roads in Queensland.* Retrieved 2022 from https://www.transurban.com/content/dam/transurban-pdfs/02/news/transurban-submission-inquiry-qld.pdf
[41] Transurban. 2022. *Travel on our roads.* Retrieved 2022 from https://insights.transurban.com/travel/travel-on-our-roads/#toll-spend-data
[42] William Thomas Tutte and William Thomas Tutte. 2001. *Graph theory*. Vol. 21. Cambridge university press.
[43] Björn Wiedersheim, Zhendong Ma, Frank Kargl, and Panos Papadimitratos. 2010. Privacy in inter-vehicular networks: Why simple pseudonym change is not enough. In *2010 Seventh international conference on wireless on-demand network systems and services (WONS)*. IEEE, 176–183.
[44] Guoming Zhang, Lianyong Qi, Xuyun Zhang, Xiaolong Xu, and Wanchun Dou. 2021. Point-of-interest recommendation with user's privacy preserving in an iot environment. *Mobile Networks and Applications* 26, 6 (2021), 2445–2460.

A NOTATIONS

The notations and acronyms are shown in Tables 3 and 4.

Notation	Description
ASR	average success rate
$correct\_trace$	correct trace
$\mathbb{D}$	set of decimal numbers
f	frequency
F	set of frequencies
G	graph equivalent of a city's map
H	set of drivers' home addresses
ID	set of drivers' identities
K	set of adversary's knowledge, namely $K = \{ID, P, G, W, H\}$
n	number of drivers
$\mathbb{N}_0$	set of non-negative integers
P	set of toll prices
$plaus\_traces$	set of plausible traces
s	toll station's identity
S	set of toll stations' identities
SR	success rate
T	set of transactions
u	upper bound
w	wallet balance
W	set of drivers' wallet balances
τ	toll price
θ	length of the billing period

Table 3: The notations.

Full name	Acronym
Average percentage decrease	APD
Average success rate	ASR
Cycle disclosure attack	CD(A)
Depth First Search	DFS
Dedicated short-range communications	DSRC
Electronic Toll Collection	ETC
On-Board Unit	OBU
Privacy-preserving ETC	PPETC
Road-side unit	RSU
Subset sum problem	SSP
Toll station disclosure attack	TSD(A)
Toll service provider	TSP
Toll service providers	TSPs

Table 4: The acronyms.

B THE TSD ATTACK

We formalize the attack in three steps and present the attack's pseudo-code in Algorithm 1. To find the set of plausible traces of a driver with identity i and its associated wallet w, $\mathcal{A}$ performs the following steps: (1) In Step 1, it creates vectors $\vec{P}, \vec{X}$, and accordingly, the linear equation E. (2) In Step 2, it solves Equation E to create all plausible traces associated with the driver. (3) In Step 3, $\mathcal{A}$ selects a plausible trace (as the correct trace) uniformly from the set of plausible traces.

Step 1. The adversary creates the following vectors and equation.

- Vector $\vec{P}$: The adversary creates the vector $\vec{P}$ using the function *create_vector* (line 3). The function takes as input the set P and the vector's size l that equals the size of the set of toll stations (line 2). The vector's elements represent toll prices. The index j in $\tau_j \in \vec{P}$ denotes the toll station's identity.
$$\vec{P} = \begin{bmatrix} \tau_1 \\ \tau_2 \\ \vdots \\ \tau_l \end{bmatrix}, \forall \tau_j \in \vec{P} : \tau_j > 0, \vec{P} \subseteq \mathbb{D}.$$
- Vector $\vec{X}$: It creates the vector $\vec{X}$ (line 4) in which each element is an unknown variable representing the frequency corresponding to the toll station s_j. The index j in $x_j \in \vec{X}$ indicates the toll station's identity.
$\vec{X} = \begin{bmatrix} x_1 & x_2 & \dots & x_l \end{bmatrix}, \vec{X} \subseteq \mathbb{N}_0, 0 \leq x_j \leq u.$
Each x_j is limited by the same upper bound, i.e., u. We set the value of the upper bound to $\left\lceil w/min(\vec{P}) \right\rceil$ (line 5) since the unknown variable x_j cannot exceed this value. The solver uses the upper bound value to solve the equation.
- The linear Equation E: It creates Equation E via the function *create_eq* (line 6). The function takes as its input the parameters $\vec{P}, \vec{X}, w$, and computes Equation 3 as $w = \vec{P} \times \vec{X}$.

Algorithm 1 The TSD attack

Input: $M = \{ID, P, G, W\}$
Output: $(i, correct\_trace)$
1: **function** TSD_ATTACK(ID, P, G, W)
2: $l \leftarrow |S|$
3: $\vec{P} \leftarrow create\_vector(l, P)$
4: $\vec{X} \leftarrow create\_vector(l)$
5: $u \leftarrow \left\lceil w/min(\vec{P}) \right\rceil$
6: $E \leftarrow create\_eq(\vec{P}, \vec{X}, w)$
7: $all\_sols \leftarrow ilp\_solver(E, u)$
8: **for** $sol \in all\_sols$ **do**
9: $F \leftarrow get\_freq(sol)$
10: $visit\_tolls \leftarrow get\_tolls(sol, S, F)$
11: **if** $check\_graph\_algo(visit\_tolls, G, S_{int}, S_{main})$ **then**
12: $plaus\_traces \leftarrow create\_trace(F, S)$
13: **end if**
14: **end for**
15: $correct\_trace \leftarrow select\_randomly(plaus\_traces)$
16: $(i, correct\_trace) \leftarrow assign\_trace(i, correct\_trace)$
17: **return** $(i, correct\_trace)$
18: **end function**

Step 2. In this step, $\mathcal{A}$ creates the set of plausible traces. To this end, the attack uses the function *ilp_solver* that takes E and u as its input (line 7) and solves the equation. For solving, the function uses DOcplex, i.e., IBM Decision Optimization CPLEX Modeling [12], then stores all solutions in the set *all_sols* (line 7).

Having obtained the solutions, for each $sol \in all\_sols$ (line 8), it calls the functions get_freq, taking sol as its input (line 9) and outputs F. The function *get_tolls* takes sol, S, and F as its input and outputs *visit_tolls*, which is the set of visited toll stations. The set *visit_tolls* includes $s_j \in S$, where its corresponding $f_j \neq 0$. Then, it uses the algorithm *check_graph_algo* to check the connectivity between the visited toll stations (line 11). The algorithm *check_graph_algo* and its pseudo-code are discussed in detail in Appendix C. This algorithm takes *visit_tolls*, G, the set of intermediate toll stations S_{int}, and main toll stations S_{main} as its inputs; if it returns *False*, it will fetch another *sol* from the set *all_sols* (line 8); otherwise, if *check_graph_algo* returns *True*, it will create the trace using the function *create_trace* (line 12), taking F and S as the input, and stores it in the set of plausible traces, i.e., $plaus\_traces = \{trace_1, trace_2, \dots, trace_d\}$. Each $trace_i \in plaus\_traces$ is a plausible trace corresponding to a solution of Equation 3. It should be mentioned that a set of plausible traces is empty if Equation 3 has no solution. The correct trace is in the set of plausible traces.

Step 3. In this step, $\mathcal{A}$ guesses uniformly the correct trace out of the set *plaus_traces* using the function *select_randomly* (line 15). The function takes *plaus_traces* as its input and outputs the correct trace, namely *correct_trace*. Finally, in order to define to whom the correct trace corresponds, $\mathcal{A}$ assigns it to the driver i with the function *assign_trace*, taking i, and *correct_trace* as the input and outputting the tuple $(i, correct\_trace)$ (line 16). Note that the adversary knows the identity of a driver to whom the wallet w belongs (see the wallet's definition in Section 2). Finally, the attack returns the tuple as its output (line 17).

Note that, for simplicity, in Section 2, we assumed that each driver's identity is associated with only one vehicle. We explain that this assumption does not affect the TSD attack's success rate. Based on this assumption, in a billing period, only one wallet balance will be assigned to each driver's identity. We remind that based on the definition of wallet balance in Section 2, the TSP knows all tuples (i, w), where i represents a driver's identity and w represents the corresponding wallet balance that the driver owes to the TSP. The assumption that a driver could use multiple vehicles implies that more than one tuple could be associated with a driver with identity i. This assumption does not impact the attack's effectiveness. In fact, to find the correct trace of a driver with identity i, the adversary executes the attack for all of the tuples corresponding to the driver.

C THE CHECK_GRAPH_ALGO ALGORITHM

The algorithm checks the connectivity among the visited toll stations (associated with a plausible trace). The pseudo-code is shown in Algorithm 2.

Before explaining the algorithm, we define two types of toll stations in ETC systems. (1) main toll station: This type of toll station is the main entrance for drivers who want to reach the corresponding intermediate toll stations. (2) intermediate toll station: This type of station is only reachable from its corresponding main toll station/s. We denote the sets of main and intermediate toll stations of graph G by S_{main} and S_{int}, respectively (this information is publicly available). In the following, we explain the algorithm, taking $visited\_tolls, G, S_{int}$, and S_{main} as its input and outputs $True$ or $False$, denoting if toll stations are connected or not, respectively.

The function *induced_subgraph* takes *visited_tolls* (the set of visited toll stations) and G as the inputs and creates the induced subgraph G' where it removes all vertices that do not belong to the set *visited_tolls* from graph G. The induced subgraph includes every edge in the original graph G that only uses vertices from *visited_tolls*. The algorithm creates the set S'_{int}, including the intermediate toll stations in the set *visited_tolls*. To this end, it uses the function *get_inter_tolls*, which computes the intersection of the sets *visited_tolls* and S_{int} and outputs the set S'_{int} (line 3).

For each intermediate toll station in *visited_tolls*, its corresponding main toll station/s must exist in the set *visited_tolls*; otherwise, the intermediate toll station is unreachable. Hence, for each $inter\_toll \in S'_{int}$ (line 4), it creates the set of corresponding main toll stations, i.e., S'_{main}. To this end, it uses the *get_main_toll* function, which takes *inter_toll* and S_{main} as the inputs and outputs the set S'_{main} (line 5). Then, it checks if the set S'_{main} is the subset of the set *visited_tolls*; if not, the set *visited_tolls* is not valid, and the algorithm returns *False*. Otherwise, if for each intermediate toll station its corresponding main toll stations exist, it checks the connectivity of graph G' using the function *check_connected* (line 10). The function checks if the directed graph G' is weakly connected, meaning that the underlying undirected graph is connected. We say an undirected graph is connected if it has a path connecting any two vertices [9]. The function *check_connected* operates based on the Depth First Search (*DFS*) algorithm. If the graph is connected, it returns *True*; otherwise, the graph is unconnected, and the algorithm returns *False*.

Computational complexity of the algorithm. The computational complexity of the algorithm primarily depends on the function *check_connected*, which takes G' as its input and utilizes the *DFS* algorithm. The complexity of the *DFS* algorithm is $O(|V'| + |E'|)$, where $|V'|$ and $|E'|$ represent the number of toll stations and the corresponding edges associated with a trace.

Algorithm 2 The graph's connectivity algorithm

Input: $visited\_tolls, G, S_{int}, S_{main}$
Output: *False* or *True*

```
1:  function CHECK_GRAPH_ALGO(visited_tolls, G, S_int, S_main)
2:      G' ← induced_subgraph(visited_tolls, G)
3:      S'_int ← get_inter_tolls(visited_tolls, S_int)
4:      for inter_toll ∈ S'_int do
5:          S'_main ← get_main_toll(inter_toll, S_main)
6:          if S'_main ⊄ visited_tolls then
7:              return False
8:          end if
9:      end for
10:     if check_connected(G') then
11:         return True
12:     else
13:         return False
14:     end if
15: end function
```

D COMPUTATION OF THE SUCCESS RATE

This algorithm computes the success rate of the TSD attack, and its pseudo-code is shown in Algorithm 3. It takes as inputs the driver's identity, namely i, and *plaus_traces* obtained from the attack. It creates the *correct_trace* (see Section 2) of driver i via the function *create_correct_trace* (line 2). Then, it checks if the condition $correct\_trace \in plaus\_traces$ holds (line 3), meaning that the correct trace exists in the set *plaus_traces* (a set of plausible traces). If not, it sets the *SR* to null (line 6); otherwise, the success rate is computed via the function *success_rate* (line 4). It should be noted that the success rate is formally computed in Section 4.1.

Algorithm 3 The compute success rate algorithm

Input: $i, plaus\_traces$
Output: *SR*

```
1: function COMPUTE_SUCCESS_RATE_ALGO(i, plaus_traces)
2:     correct_trace ← create_correct_trace(i)
3:     if correct_trace ∈ plaus_traces then
4:         SR ← success_rate(plaus_traces)
5:     else
6:         SR ← null
7:     end if
8:     return SR
9: end function
```

E DISTRIBUTION OF SUCCESS RATES

As a concrete example, we illustrate the distribution of *SR*s in the first box plot concerning the wallet range [\$0, \$10]. We compute all plausible wallet balances that fall into the range [\$0, \$10] (see Step 1 in Section 5.2.1), shown in the second row of Table 5. The row includes 93 different tuples $(j, plausible\ wallet)$, demonstrating the jth plausible wallet balance. The third row includes 93 tuples (j, SR), demonstrating the *SR* (in percentage) corresponding to the jth plausible wallet balance (in the second row). The *ASR* equals the average of all *SR*s in the second row, i.e., 94% (see Step 2 in Section 5.2.1). The results are shown in Table 5.

F HEURISTIC ALGORITHMS

The details and pseudo-code of the heuristics are discussed in the following.

F.1 The first heuristic

We present a heuristic algorithm that the TSD attack (see Algorithm 1) can apply. The heuristic contributes to a higher success rate for certain settings. The heuristic takes the set of plausible traces *plaus_traces* (obtained by Algorithm 1) and *threshold* as inputs and outputs the updated set *plaus_traces*. The pseudo-code of the first heuristic algorithm is shown in Algorithm 4. To apply the heuristic, the TSD attack should call the heuristic algorithm between lines 14 and 15 shown in Algorithm 1.

Parameter settings	$M = \{ID, P, G, 0 < w \leq 10\}, \lvert S\rvert = 9, u = 6, \theta =$ a month
Plausible wallets	(1, 1.72), (2, 2.68), (3, 2.84), (4, 3.19), (5, 3.44), (6, 4.09), (7, 4.4), (8, 4.55), (9, 4.56), (10, 4.91), (11, 5.11), (12, 5.16), (13, 5.36), (14, 5.46), (15, 5.52), (16, 5.68), (17, 5.81), (18, 5.87), (19, 6.03), (20, 6.12), (21, 6.27), (22, 6.28), (23, 6.38), (24, 6.63), (25, 6.77), (26, 6.83), (27, 6.88), (28, 6.93), (29, 7.08), (30, 7.18), (31, 7.23), (32, 7.24), (33, 7.28), (34, 7.39), (35, 7.4), (36, 7.53), (37, 7.59), (38, 7.74), (39, 7.75), (40, 7.79), (41, 7.84), (42, 7.95), (43, 7.99), (44, 8.0), (45, 8.04), (46, 8.1), (47, 8.14), (48, 8.18), (49, 8.2), (50, 8.3), (51, 8.35), (52, 8.36), (53, 8.49), (54, 8.52), (55, 8.55), (56, 8.6), (57, 8.64), (58, 8.65), (59, 8.71), (60, 8.8), (61, 8.87), (62, 8.9), (63, 8.95), (64, 8.96), (65, 9.0), (66, 9.06), (67, 9.1), (68, 9.11), (69, 9.12), (70, 9.2), (71, 9.22), (72, 9.25), (73, 9.31), (74, 9.45), (75, 9.46), (76, 9.47), (77, 9.51), (78, 9.55), (79, 9.56), (80, 9.57), (81, 9.61), (82, 9.66), (83, 9.67), (84, 9.71), (85, 9.72), (86, 9.76), (87, 9.77), (88, 9.82), (89, 9.86), (90, 9.9), (91, 9.91), (92, 9.92), (93, 9.96)
Success rates (SR)	(1, 100.0), (2, 100.0), (3, 100.0), (4, 100.0), (5, 100.0), (6, 100.0), (7, 100.0), (8, 100.0), (9, 100.0), (10, 100.0), (11, 50.0), (12, 100.0), (13, 100.0), (14, 100.0), (15, 100.0), (16, 100.0), (17, 100.0), (18, 100.0), (19, 100.0), (20, 100.0), (21, 100.0), (22, 100.0), (23, 100.0), (24, 100.0), (25, 100.0), (26, 50.0), (27, 100.0), (28, 100.0), (29, 100.0), (30, 100.0), (31, 100.0), (32, 100.0), (33, 100.0), (34, 100.0), (35, 100.0), (36, 100.0), (37, 100.0), (38, 100.0), (39, 100.0), (40, 50.0), (41, 100.0), (42, 50.0), (43, 100.0), (44, 100.0), (45, 100.0), (46, 100.0), (47, 100.0), (48, 100.0), (49, 100.0), (50, 33.33), (51, 100.0), (52, 100.0), (53, 100.0), (54, 100.0), (55, 33.33), (56, 100.0), (57, 100.0), (58, 50.0), (59, 100.0), (60, 100.0), (61, 100.0), (62, 100.0), (63, 100.0), (64, 100.0), (65, 100.0), (66, 100.0), (67, 100.0), (68, 100.0), (69, 100.0), (70, 50.0), (71, 100.0), (72, 100.0), (73, 100.0), (74, 100.0), (75, 100.0), (76, 100.0), (77, 50.0), (78, 100.0), (79, 100.0), (80, 100.0), (81, 100.0), (82, 50.0), (83, 50.0), (84, 100.0), (85, 100.0), (86, 100.0), (87, 100.0), (88, 100.0), (89, 100.0), (90, 100.0), (91, 100.0), (92, 100.0), (93, 100.0)

Table 5: Plausible wallet balances and corresponding *SR*s for the range $[\$0, \$10]$.

Algorithm 4 The first heuristic algorithm

Input: *plaus_traces*, *threshold*
Output: *plaus_traces*

```
1: function FIRST_HEURISTIC_ALGO(plaus_traces, threshold)
2:     for trace ∈ plaus_traces do
3:         if |trace| > threshold then
4:             remove(plaus_traces, trace)
5:         end if
6:     end for
7:     return plaus_traces
8: end function
```

The *threshold* is the maximum number of toll points a driver has visited during a billing period. From the set *plaus_traces*, the heuristic algorithm keeps plausible traces with a length equal to and less than the threshold and discards plausible traces with a length greater than the *threshold*. Note that the length of a plausible trace, i.e., $|trace| = |\{(s_1, f_1), (s_2, f_2), \ldots, (s_l, f_l)\}|$, equals the number of visited toll stations in the set *trace*. Hence, for each plausible trace *trace* in the set *plaus_traces*, the heuristic checks the condition $|trace| > threshold$. If it holds, the algorithm removes the plausible trace from the set by the function *remove*. Finally, it outputs the updated set *plaus_traces*.

Then, the adversary uniformly guesses the correct trace from the updated set *plaus_traces*. Note that this heuristic helps to increase the attack's success rate $SR = (1 \,/\, |plaus\_traces|)$ since the length of the updated set *plaus_traces* becomes smaller after removing implausible traces, i.e., the traces with a length greater than the *threshold*.

The heuristic evaluation. To evaluate the heuristic, we employ the parameter settings discussed in Section 5.1 for Brisbane's ETC system. However, some settings differ from those discussed in Section 5.1. For each range $[w_l, w_u]$, we only consider those plausible wallet balances where the corresponding updated *plaus_traces* (i.e., the output) in the heuristic algorithm does not become empty for thresholds ranging from 3 to 8. This is because the success rate is not defined for the empty set *plaus_traces* with a length of zero. Note that the set *plaus_traces* will become empty if, for every *trace* in the set, the condition $|trace| > threshold$ holds (see Algorithm 4). Moreover, we do not perform our analysis for the thresholds 1 and 2 since a large number of the updated *plaus_traces* corresponding to plausible wallet balances become empty, which leaves only a few plausible wallet balances for our analysis. This makes the computation of the metrics *APD* and *ASR* inaccurate. We highlight that the threshold values are the potential ones that can be deduced from the statistical data by an adversary in the case study of Brisbane. The details of the parameter settings are presented in Table 6.

***Impact of the heuristic on** APD.* We evaluate to what extent the heuristic helps to reduce, on average, the number of plausible traces associated with all plausible wallet balances within the wallet range $[w_l, w_u]$. To this end, we use APD, which is the average percentage decrease in the number of plausible traces for each wallet range $[w_l, w_u]$. The metric "percentage decrease" shows to what extent the number of plausible traces in a set of plausible traces has reduced after applying the heuristic. The *APD* denotes the average of all percentage decreases corresponding to all plausible wallet balances within the wallet range $[w_l, w_u]$.

For the analysis, we consider four different wallet balance ranges, namely $0 < w \leq 10, 10 < w \leq 20, 20 < w \leq 40$, and $40 < w \leq 60$. Then, we apply the heuristic to all sets of plausible traces associated with all plausible wallet balances within each range $[w_l, w_u]$. For each wallet range $[w_l, w_u]$, we vary the potential threshold from

Parameter settings	Num of plausible wallets	Drivers' proportion
$M = \{ID, P, G, 0 < w \leq 10\}$, $\lvert S\rvert = 9$, θ = a month, $threshold \in [3, 8]$	93	61.38%
$M = \{ID, P, G, 10 < w \leq 20\}$, $\lvert S\rvert = 9$, θ = a month, $threshold \in [3, 8]$	652	14.31%
$M = \{ID, P, G, 20 < w \leq 40\}$, $\lvert S\rvert = 9$, θ = a month, $threshold \in [3, 8]$	1975	11.12%
$M = \{ID, P, G, 40 < w \leq 60\}$, $\lvert S\rvert = 9$, θ = a month, $threshold \in [3, 8]$	2001	4.57%

Table 6: . Each row shows the parameter settings for the case study of Brisbane's ETC system to analyze the impact of the first heuristic. The rows show the number of plausible wallets and drivers' proportion corresponding to the wallet range $[w_l, w_u]$.

3 to 8 and analyze how it impacts *APD*. The results are shown in Table 2. Regarding the wallet range $0 < w \leq 10$, the values for *APD* are the same, i.e., 0% for all thresholds. The reason is that the size of the corresponding plausible traces is equal and below 3 for all the thresholds, which does not contribute to reducing the number of plausible traces. Nevertheless, the success rate is already considerably high (94%) prior to applying the heuristic. For the wallet range $10 < w \leq 20$, the *APD* is increasing from 0% to 24.1% (from threshold 8 to 3). For the wallet range $20 < w \leq 40$, the *APD* increases from 0% to 81.33% (from threshold 8 to 3). For the wallet range $40 < w \leq 60$, the *APD* is increasing from 0.27% to 97.21% (from threshold 8 to 3). Overall, *APD* demonstrates an upward trend by decreasing the threshold value.

Given a fixed threshold j, where $3 \leq j \leq 8$, the *APD* values show a significant increase across all wallet ranges. For instance, when the threshold is set to 3, the *APD* values are 0%, 24.1%, 81.33%, and 97.21% (see the second column). This increasing trend can be attributed to the fact that larger wallet balances encompass a greater number of plausible traces, many of which are ultimately discarded by the heuristic.

***Impact of the heuristic on** ASR.* We analyze how the heuristic, with threshold j, impacts *ASR* for each wallet range $[w_l, w_u]$. We remind that *ASR* is the average of all success rates corresponding to all plausible wallet balances within the range $[w_l, w_u]$. The results are shown in Table 2. For each wallet range $[w_l, w_u]$, we vary the threshold from 3 to 8 and analyze how it impacts *ASR* by discarding implausible traces. Furthermore, we calculate the "percentage increase" to quantify the extent to which the ASR, associated with threshold 3, has increased compared to the ASR without applying any heuristic. For the wallet range $0 < w \leq 10$, the *ASR* has the same value of 94%, associated with all thresholds. This is because the size of the corresponding plausible traces is equal and below 3 for all the thresholds, which does not contribute to reducing the number of plausible traces. For the wallet range $10 < w \leq 20$ (for thresholds from 8 to 3), the *ASR* is increasing from 51.16% to 65.84%, resulting in a percentage increase of $\frac{65.84-51.16}{51.16} \approx 29\%$. For the wallet range $20 < w \leq 40$, the *ASR* increases from 4.69% reaching to 22.05%. This results in a percentage increase of $\frac{22.05-4.69}{4.69} \approx 370\%$. For the wallet range $40 < w \leq 60$, the *ASR* increases from 0.15% reaching to 5.2%, leading to the percentage increase of $\frac{5.2-0.15}{0.15} \approx 3367\%$, which is significantly high. Overall, *ASR* shows an upward trend by decreasing the threshold. Besides, the values of percentage increases demonstrate that as the wallet ranges get larger, their corresponding "percentage increase" significantly increases. This is because the plausible wallet balances within a larger wallet range lead to a larger number of plausible traces, many of which are discarded by the heuristic, leading to a relatively large *ASR*.

F.2 The second heuristic

The heuristic algorithm takes a set of plausible traces and a distribution function as inputs. The algorithm outputs a set of plausible traces, including the traces and their corresponding probabilities. The pseudo-code of the heuristic is shown in Algorithm 5. To apply the heuristic, the TSD attack should call the heuristic algorithm between lines 14 and 15 shown in Algorithm 1. The algorithm performs as follows. For each trace in the set of plausible traces, it computes the probability p using the distribution function $P(x)$. The distribution function describes the probability that a given trace is the correct one. The function takes the number of visited toll stations in a trace, i.e., $|trace|$ and outputs the corresponding p. Using the function *assign_prob*, the algorithm assigns the probability p to the *trace*, creating the tuple $(trace, p)$. Then, it stores the tuple in the set E. The adversary may use different strategies to select the correct trace in the set E. For example, it can select the trace with the highest probability (i.e., non-uniformly). If two or more traces are assigned the same highest probability, the adversary can uniformly select one of them.

Algorithm 5 The second heuristic algorithm

Input: $plaus\_traces, P(x)$
Output: E
1: **function** SECOND_HEURISTIC_ALGO($plaus\_traces, P(x)$)
2: **for** $trace \in plaus\_traces$ **do**
3: $p \leftarrow P(|trace|)$
4: $(trace, p) \leftarrow assign\_prob(trace, p)$
5: $E \leftarrow (trace, p)$
6: **end for**
7: **return** E
8: **end function**

F.3 The third heuristic

The third heuristic algorithm takes as inputs ID, P, G, and the list of yearly wallet balances (*yearly_wallets*) associated with a driver. Then, it outputs a set of tuples, including a cluster and its corresponding probability. Each of the clusters includes the potential traces a driver might have made within a year. The pseudo-code of the heuristic is shown in Algorithm 6.

Algorithm 6 The third heuristic algorithm

Input: $\{ID, P, G, yearly\_wallets\}$
Output: $set\_cluster\_prob$

```
function THIRD_HEURISTIC_ALGO({ID, P, G, yearly_wallets})
    for w ∈ list_yearly_wallets do
        plaus_traces ← TSD_attack(ID, P, G, w)
        N ← plaus_traces
    end for
    set_clusters ← create_clusters(N)
    for cluster ∈ set_clusters do
        similarity_dist ← Euclidean_distance(cluster)
        set_distances ← similarity_dist
    end for
    list_probs ← normalize(set_distances)
    for cluster ∈ set_clusters do
        (cluster, p) ← assign_prob(cluster, list_probs)
        set_cluster_prob ← (cluster, p)
    end for
    return set_cluster_prob
end function
```

The algorithm performs as follows. For each wallet w in the list $yearly\_wallets$, it computes the corresponding set of plausible traces via the TSD attack (see Algorithm 1). Then, it stores the set of plausible traces in the list N. Using the function $create\_clusters$, the algorithm creates all possible clusters, where each cluster includes a different permutation of traces, each of which belongs to a different set of plausible traces in the set N. For creating the permutation of traces, the algorithm uses the Cartesian product of the sets in N. The made clusters will be stored in the set $set\_clusters$. For each $cluster$ in the set $set\_clusters$, the algorithm computes the similarity distance among the traces in the cluster using the $Euclidean\_distance$ function. Then, the algorithm stores $similarity\_dist$ to the set of similarity distances, i.e., $set\_distances$. The Euclidean distance between two plausible traces, $trace_1 = \{(s_1, f_1), (s_2, f_2), \ldots, (s_j, f_j)\}$ and $trace_2 = \{(s_1, f_1'), (s_2, f_2'), \ldots, (s_j, f_j')\}$, is calculated using the below formula:

$$distance = \sqrt{(f_1 - f_1')^2 + (f_2 - f_2')^2 + \cdots + (f_j - f_j')^2} \quad (9)$$

Note that each pair of frequencies inside the parentheses corresponds to the same toll station sj. The formula indicates that if the frequencies within each pair of parentheses are close to each other, the distance will be closer to zero.

The function $normalize$ takes the set of similarity distances and normalizes its elements to the corresponding probabilities stored in the list $list\_probs$. Each value in the list is the probability that the driver could have made the traces inside the $cluster$. Then, for each cluster in the set $set\_clusters$, the algorithm via the function $assign\_prob$ assigns the cluster to its corresponding probability p and outputs the tuple $(cluster, p)$. Then, the algorithm stores the tuple $(cluster, p)$ to the set $set\_cluster\_prob$. Finally, the adversary may use different strategies for selecting the correct cluster from the set $set\_cluster\_prob$. By the correct cluster, we mean a cluster including the correct traces of a driver made within a one-year billing period. One example of a strategy is that the adversary selects a cluster with the highest assigned probability (i.e., non-uniformly). If two or more of the clusters have equal highest probabilities, the adversary will uniformly select one of the clusters. It should be noted that the cluster with the highest probability may not always be the correct one. This is because, inside the cluster, there could be plausible traces where there is a coincidental similarity among them. With this heuristic, the adversary achieves a stronger goal than the TSD goal. By stronger, we mean that the adversary learns all plausible traces associated with a driver within a year.

G IMPACT OF PARAMETERS ON PRIVACY

We analyze the impact of different settings for each parameter on a driver's privacy. To achieve this, we vary the settings of a specific parameter, such as toll prices, while keeping the settings for the remaining parameters fixed. We then observe how the success rate changes with respect to the different settings. Table 7 displays the settings for the parameters we are analyzing.

Parameter P. To analyze the impact of toll prices on a driver's privacy, we consider ten different ranges of toll prices as $[1, j], 1 \leq j \leq 10$, where j ranges from one to ten, creating a diversity of range spans. We consider different wallet ranges, $[w_l, w_u]$, shown in Figure 2, and for each wallet range, we perform the following experiment to compute the *ASR* explained in Section 5.2.1. For each toll price range $[1, j]$, we uniformly randomly select nine toll prices from the range $[1, j]$ (one price for each toll station in Brisbane) and assign them to the corresponding toll stations. Then, we compute the *ASR*. We repeat this experiment 50 times, each with different selected toll prices from the range $[1, j]$. Repeating experiments multiple times reduces the impact of random fluctuations, enabling us to observe a range of results and determine the average outcome. This provides more reliable and accurate *ASR*s. Then, we record the resulting *ASR*s in a set that results in a box plot (in green) for the toll price range $[1, j]$. The *ASR*s corresponding to the wallet ranges [\$1, \$10], [\$10, \$20], and [\$20, \$40] are demonstrated in Figures 6a, 6b, and 6c respectively. The second row in Table 7 shows all settings for the parameter P.

Parameter w. To analyze the impact of the wallet balance on a driver's privacy, we consider the three different wallet ranges discussed in Figure 2. Then we compute the success rate (see Section 4.1) for all plausible wallets within each wallet range $[w_l, w_u]$. The red points in Figure 6d show the success rate for all plausible wallets in the range [\$0, \$10]. The success rate associated with most wallets is 100%, and for some wallets are 50% and 33%. The blue points correspond to all plausible wallets in the range [\$10, \$20]. The density of blue points demonstrates that for the significant number of wallet balances, the success rates are 100%, 50%, and 33%, and for the rest, the success rate is between 6% and 33%. The green points concern all plausible wallets in the range [\$20, \$40]. For the wallets between \$20 and \$25, the success rate is between 3% and 100%, and for the wallets between \$25 and \$40, the success rate is between 0% and 20%. The values for the toll prices and the length of the billing period are fixed. The parameter settings are summarized in Table 7.

Parameter θ. To analyze the impact of the length of the billing period (θ), we consider different lengths, i.e., from one week to eight weeks. In each billing period, drivers' wallet range can be

Parameter	P	θ	u	$\|S\|$	w
P	$[\$1, \$j], 1 \leq j \leq 10$	a month	[0, 10], [0, 20], [0, 40]	9	[\$0, \$10], [\$10, \$20], [\$20, \$40]
w	P	a month	6, 12, 24	9	[\$0, \$10], [\$10, \$20], [\$20, \$40]
θ	P	[1, 8]	6, 12, 18, 24, 30, 35, 41, 47	9	$[\$0, \$10 * j], 1 \leq j \leq 8$
$\|S\|$	[\$1.72, \$5.46]	a month	[0, 6], [0, 12], [0, 24]	[9, 20]	[\$0, \$10], [\$10, \$20], [\$20, \$40]

Table 7: Parameter settings to evaluate the impact of each parameter on a driver's privacy. Note that each row is concerned with the parameter being analyzed.

estimated by the statistics [40]. It shows that 85% of drivers pay wallet balances of less than \$10 a week. This concludes that the wallet range for each of the eight billing periods is: $w \leq \$10$ for one week up to $w \leq \$80$ for eight weeks. Then, for each wallet range, we compute the *ASR* discussed in Section 5.2.1. The number of toll stations and their corresponding toll prices are fixed. The impact of the length is shown in Figure 6e. The graph shows a significant decrease in the *ASR*, from approximately 93% to almost 10%. The graph demonstrates that drivers' privacy is at risk of violation, even considering an eight-week billing period. The details of parameter settings are in Table 7.

Parameter $|S|$. To investigate the impact of the number of toll stations on a driver's privacy, we increase the toll stations' number from nine (current Brisbane's number of toll stations) to 20. We consider all nine toll stations in Brisbane, then for each newly added toll station, we assign a toll price to it within the range of [\$1.72, \$5.46], where the lower and upper bounds are the minimum and maximum of the set of toll prices in Brisbane, i.e., P. The range ensures that the selected toll price is close to Brisbane's toll prices, creating more realistic scenarios. We consider different wallet ranges, $[w_l, w_u]$, shown in Figure 2, and for each range, we compute the *ASR* as explained in Section 5.2.1. The graphs in Figure 6f show that overall the *ASR* decreases as the number of toll stations increases. The red graph concerns the wallet range [\$0, \$10] and decreases from about 93% to almost 60%. The blue graph concerns the wallet range [\$10, \$20] and varies from 53% to 4%, and the green graph concerns the range [\$20, \$40] and changes from 5% to zero. Drivers' privacy with wallets below \$20 (red and blue graphs) is violated even considering the maximum number of toll stations (20). Drivers' privacy associated with the range [\$20, \$40] (green graph) is at risk when the toll station's number is between 9 and 11, although the success rate is not very high. Overall, a driver's privacy is more at risk with a low density of toll stations. The details of the settings are in Table 7.

H FIND_CYCLE_ALGO ALGORITHM

We present an algorithm that gets as inputs a set of toll stations, a home location, a city's graph, and a strategy and then outputs the corresponding cycle passing through the toll points and the home location. The pseudo-code is shown in Algorithm 7.

The algorithm first checks if the toll stations are connected using the algorithm *check_graph_algo* (see Appendix C). If the toll stations are not connected, it returns *False*; otherwise, it computes the induced subgraph G' using the function *induced_subgraph*.

Algorithm 7 The find cycle algorithm

Input: $visited\_tolls, G, h_i$
Output: $cycle$

```
function FIND_CYCLE_ALGO(visited_tolls, G, h_i, strategy)
    flag ← check_graph_algo(visited_tolls, G, S_int, S_main)
    if flag == False then
        return False
    end if
    G' ← induced_subgraph(visited_tolls, G)
    cycle ← find_cycle(G', h_i, strategy)
    if cycle == null then
        return null
    end if
    return cycle
end function
```

Then, it uses the function $find\_cycle$ taking G', h_i, and the $strategy$ as its inputs and finds the corresponding cycle starting from and ending h_i, and passing through all the toll points at least once in G' if any. The function stores its output in the *cycle*. If no cycle is found, the algorithm will return *null*. In the following, we highlight some points concerning the cycles.

- A cycle includes one or more toll roads on which at least one toll station is located. The cycle might also include one or more roads on which no toll station is located.
- It is worth mentioning that the adversary does not know a driver's destination/s, where the driver stops for a while to do an activity. The driver could exit somewhere on the toll road to reach his destination to do an activity and then reenter the road again to continue his cycle. For example, the toll roads in Brisbane are mainly used for activities, such as taking a holiday/getaway, going to the airport, and social activities [41]. Since the adversary has no information about the driver's destination/s, it cannot precisely follow the driver. Nevertheless, it can approximately follow the cycle, ignoring the paths to the destination/s. For example, after passing one or more toll stations, a driver exits the toll road to get to his workplace and then reenters the road to return home. In this example, although the adversary cannot follow the driver exactly to his workplace, it can approximately follow the cycle made by the driver.
- It is worth mentioning that although the adversary does not know a driver's destination/s, it can guess it/them. For example, considering that a driver visited a toll station close to the airport, he might have exited the toll road to go to the

airport. Or, given that the driver visited a toll station close to an industrial area, he might have exited the road to work in a factory.

The computational complexity of Algorithm 7. The complexity of the algorithm depends on the complexity of function $find\_cycle$, which varies by the strategy the function uses. For example, we compute the complexity considering that the strategy is the shortest distance, i.e., selecting a route that minimizes the overall distance traveled to reach a destination. In this case, the function $find\_cycle$ is a variant of the algorithm used for solving the traveling salesman problem. This variant asks the following question: "Given a set of cities and the roads connecting them, what is the shortest cycle that visits each city *at least* once and returns to the origin city?" [19]. The TSP is NP-hard for which exact and approximate algorithms exist [29]. The complexity of an exact algorithm using dynamic programming is $O(2^{|V'|} \cdot |V'|^2)$, where $|V'|$ is the graph order, i.e., the number of vertices in graph G'. Although the complexity is exponential in the length of $|V'|$, the exact algorithms can efficiently compute the cycle in our application as the graph order is very small. The graph order, in our case, equals the number of visited toll stations by a driver in a city, which is typically a small number. The example in Appendix K uses this strategy, i.e., the shortest distance.

I SSP-CD ATTACK

We introduce the SSP-CD attack to achieve the CD goal (the pseudo-code is shown in Algorithm 8). We employ a similar idea used to accomplish the TSD goal where the idea was to solve the SSP to obtain the set of plausible traces, where a plausible trace is defined as $trace = \{(s_1, f_1), (s_2, f_2), \ldots, (s_l, f_l)\}$ (see Section 2).

Concerning the CD goal, similarly, $\mathcal{A}$ first needs to find the set of plausible traces of a driver (with the wallet w) defined as $trace = \{(c_1, f_1), (c_2, f_2), \ldots, (c_y, f_y)\}$, where c_i is the cycle and f_i is its corresponding frequency. Then, $\mathcal{A}$ uniformly guesses the correct trace from the set. Hence, to create a plausible trace, it needs two items: (1) the set of cycles $C = \{c_1, \ldots, c_y\}$ and (2) frequencies, i.e., $F = \{f_1, \ldots, f_y\}$, for which it performs the following steps:

(1) To obtain the set of cycles *Cycle* considering graph G, $\mathcal{A}$ computes all different combinations of toll stations that could potentially be constituents of a cycle. To this end, $\mathcal{A}$ stores the graph's nodes to the set S using the function *get_toll*, taking G as its input (line 2). Then, it uses the function *compute_combinations*, taking S as the input, and creates the set of all different combinations of toll stations that can be made, which is stored in *all_comb* (line 3). Each combination consists of $k, 1 \leq k \leq |S|$ toll points out of the total different toll stations, i.e., $|S|$. For each $comb \in all\_comb$, $\mathcal{A}$ finds its corresponding cycle, if any. To find the cycles, it uses the algorithm $find\_cycle\_algo$. The function takes $comb, G, h$, and the strategy as inputs and outputs the cycle, which is stored in the set of cycles *Cycle* (line 5).

   Example. Given that an ETC system consists of three toll points, i.e., s_1, s_2, and s_3, function *compute_combinations* results in seven combinations denoted as $\{s_1\}, \{s_2\}, \{s_3\}, \{s_1, s_2\}, \{s_1, s_3\}, \{s_2, s_3\}, \{s_1, s_2, s_3\}$. The toll stations in each combination, along with a driver's home location h, could potentially lead to a cycle, if any. The combination $\{s_1, s_2, s_3\}$ along with h could result in a cycle, for example, denoted as $hs_3s_2s_1h$.

(2) To obtain F, it needs to solve the SSP, for which it creates the following linear diophantine equation. The adversary, using the function *create_eq*, creates a similar equation as Equation 3, where the cycle price π is used instead of the toll price τ (line 11). The equation is as follows:

$$w = \pi_1 \cdot x_1 + \pi_2 \cdot x_2 + \cdots + \pi_y \cdot x_y, x_j \in \mathbb{N}_0, 1 \leq j \leq y \qquad (10)$$

   In Equation 10, the cycle price π_j is the summation of k toll prices $\tau_i \in P$ corresponding to k toll stations along the cycle c_j, which is computed as $\pi_j = \sum_{i=1}^{k} \tau_i$. The set of cycle prices is denoted as $\Pi = \{\pi_1, \pi_2, \ldots, \pi_y\}$. The interpretation of Equation 10 is that the summation of the prices of the cycles made by a driver in a billing period results in w. Each x_j, in the equation, represents the frequency f_j that the driver made the cycle c_j. Then, $\mathcal{A}$ solves the equation via the function *ilp_solver* and stores the solutions in the set *all_sols* (line 12).

(3) For each solution *sol* in the set *all_sols*, $\mathcal{A}$ creates the set of frequencies F (line 14). Having obtained the sets *Cycle* (from the previous steps) and F, the adversary computes the plausible trace and stores it in the set *plaus_cycle_traces* (line 15). Note that one solution of Equation 3 leads to a plausible trace, and since the equation may have more than one solution, this leads to a set of plausible traces as *plaus_cycle_traces*. Finally, $\mathcal{A}$ guesses the correct trace uniformly from the set *plaus_cycle_traces* (line 17).

Algorithm 8 The SSP-CD attack

Input: $M = \{ID, P, G, W\}, H$
Output: $(i, correct\_trace)$

1: **function** SSP_CD_ATTACK(M, H)
2: $S \leftarrow get\_tolls(G)$
3: $all\_comb \leftarrow compute\_combinations(S)$
4: **for** $comb \in all\_comb$ **do**
5: $Cycle \leftarrow find\_cycle\_algo(comb, G, h_i, strategy)$
6: **end for**
7: $y \leftarrow |C|$
8: $\vec{\Pi} \leftarrow create\_vector(y, \Pi)$
9: $\vec{X} \leftarrow create\_vector(y)$
10: $u \leftarrow \left\lceil w/min(\vec{\Pi}) \right\rceil$
11: $E \leftarrow create\_eq(\vec{\Pi}, \vec{X}, w)$
12: $all\_sols \leftarrow ilp\_solver(E, u)$
13: **for** $sol \in all\_sols$ **do**
14: $F \leftarrow get\_freq(sol)$
15: $plaus\_cycle\_traces \leftarrow create\_trace(F, C)$
16: **end for**
17: $correct\_trace \leftarrow select\_randomly(plaus\_cycle\_traces)$
18: $(i, correct\_trace) \leftarrow assign\_trace(i, trace)$
19: **return** $(i, correct\_trace)$
20: **end function**

The issue with this idea is that the total number of different cycles (which equals the number of variables n Equation 10) in a city's graph grows exponentially in the size of $|S|$. This is because the number of cycles is correlated with the number of combinations of toll stations (see the first step) obtained by the following formula.

$$\sum_{k=1}^{|S|} \binom{|S|}{k} = 2^{|S|} - 1 \tag{11}$$

The exponential number of variables in Equation 10 makes solving the equation computationally infeasible. While this idea may be viable for a small number of toll stations in a sparse graph, it is generally infeasible for a large number of toll stations and a dense graph. Hence, in Section 6.1, we present a new idea that exploits the TSD goal so as to achieve the CD goal without needing to create and solve Equation 10.

I.0.1 ***The computational complexity of the attack:*** The complexity mainly depends on two factors: the complexity of the algorithm $find\_cycle\_algo$ to obtain the cycles and the complexity of solving Equation 10. We consider G as a complete undirected graph and compute all Hamiltonian cycles in the graph, resulting in the worst-case number of cycles. To obtain the total number of cycles, for each combination $comb \in all\_comb$ (including k toll points), we compute the number of Hamiltonian cycles, which is $\frac{(k-1)!}{2}$ [18]. Based on Formula 11, the total number of combinations is computed as $\binom{|S|}{k}$. Therefore, the total number of cycles in G is equal to the number of combinations multiplied by the associated number of cycles, which is $\frac{(k-1)!}{2}$, as given by the following formula.

$$\sum_{k=3}^{|S|} \binom{|S|}{k} \times \frac{(k-1)!}{2} \tag{12}$$

Formula 12 shows the total number of Hamiltonian cycles in G, which is at least exponential. For example, given G including 10 toll stations, the total number of Hamiltonian cycles equals 556014, which is the number of variables in Equation 10. This number of variables makes solving the equation computationally infeasible. In Section 6.1, we present a new idea to handle the infeasibility problem of solving Equation 10 due to many variables.

J THE CD ATTACK

We discuss in detail the CD attack presented to achieve the CD goal. The attack takes as inputs the sets M, H, and outputs the correct trace guessed uniformly by $\mathcal{A}$. The pseudo-code is shown in Algorithm 9. The full example of the attack is illustrated in Appendix K. The main idea of the attack is that $\mathcal{A}$ exploits a driver's correct trace, obtained by the TSD attack, to create the corresponding plausible traces, including cycles and their associated frequencies. In six stages, we explain the CD attack.

(1) Create the multiset. The CD attack executes the TSD attack to obtain the correct trace of driver i denoted as $correct\_trace = \{(s_1, f_1), \ldots, (s_l, f_l)\}$, including the visited toll stations and their associated frequencies. Given the $correct\_trace$, $\mathcal{A}$ creates a multiset, namely $m\_visited\_tolls$ out of the correct trace including all the visited toll stations. Each toll station s_i included in the correct trace is repeated f_i times in the multiset. For example, if we consider $correct\_trace = \{(s_1, 2), (s_2, 3)\}$, it results in the multiset $m\_visited\_tolls = \{s_1, s_1, s_2, s_2, s_2\}$. $\mathcal{A}$ utilizes the function $create\_multiset$ (line 3) to generate the multiset.

(2) Create partitions. $\mathcal{A}$ obtains all the partitions by partitioning the multiset into parts/*segments*, each with different visited toll stations. Each partition denoted as the multiset $partition = \{segment_1, \ldots, segment_y\}$. Each segment $segment_j$ in the $partition$ is a set representing different visited toll stations that, along with h_i, could potentially be *constituents* of a cycle made by the driver in a billing period. Two condition holds concerning a segment: (1) all the toll stations in a segment are distinct, and (2) the order of the toll stations in a segment does not matter. Note that the multiset $m\_visited\_tolls$ can be partitioned in different ways, resulting in different partitions. To create the partitions, it uses the function $create\_partitions$ taking the multiset $m\_visited\_tolls$ as the input and outputs the set of partitions denoted as $all\_partitions = \{partition_1, \ldots, partition_z\}$, where z denotes the number of partitions (line 4).

Example. Given the example of multiset $\{s_1, s_1, s_2, s_2, s_2\}$, the function $create\_partitions$ divides the multiset into the following partitions: 1: $\{\{s_1\}, \{s_1, s_2\}, \{s_2\}, \{s_2\}\}$, 2: $\{\{s_1, s_2\}, \{s_1, s_2\}, \{s_2\}\}$, 3: $\{\{s_1\}, \{s_1\}, \{s_2\}, \{s_2\}, \{s_2\}\}$. For example, considering h_i as the driver's home location, the segment $\{s_1, s_2\}$ in the first partition could lead to a cycle denoted as $h_i s_2 s_1 h_i$, meaning that the toll points in each segment along with the home location could lead to a cycle.

Algorithm 9 The CD attack

Input: $M = \{ID, P, G, W\}$, H
Output: $(i, correct\_trace)$

```
1:  function CD_ATTACK(M, H)
2:      (i, correct_trace) ← TSD_attack(ID, P, G, W)
3:      m_visited_tolls ← create_multiset(correct_trace)
4:      all_partitions ← create_partitions(m_visited_tolls)
5:      visited_tolls ← get_tolls(correct_trace)
6:      all_comb ← compute_combinations(visited_tolls)
7:      for comb ∈ all_comb do
8:          list_cycles ← find_cycle_algo(comb, G, h_i, strategy)
9:      end for
10:     for partition ∈ all_partitions do
11:         for segment ∈ partition do
12:             Cycle ← list_cycles(segment)
13:         end for
14:         F, C ← merge_segments_algo(partition, Cycle)
15:         if C ≠ False then
16:             trace ← create_traces(F, C)
17:             plaus_cycle_traces ← trace
18:         end if
19:     end for
20:     correct_trace ← select_randomly(plaus_cycle_traces)
21:     (i, correct_trace) ← assign_trace(i, correct_trace)
22:     return (i, correct_trace)
23: end function
```

(3) Create the pre-computed list of cycles. To transform each segment (in the partition) to its corresponding cycle, $\mathcal{A}$ selects the

corresponding cycle from a pre-computed list of cycles. This prevents computing the cycle for the same segments belonging to different partitions, as different partitions might have some segments in common. Hence, $\mathcal{A}$ first precomputes the list of all cycles that can be made from combinations of the visited toll stations. To this end, it first obtains all different combinations of visited toll stations and then finds the corresponding cycle of each combination. To make the pre-computed list, $\mathcal{A}$ gets the visited toll from *correct_trace* using the function *get_toll* and stores it in the *visited_tolls* (line 5). Then, using the function *compute_combinations*, it generates all combinations and stores the output in the set *all_comb* (line 6). For each combination *comb* in the set *all_comb*, it finds its corresponding cycle (if any) via Algorithm 7 ($find\_cycle\_algo$), taking $comb, G, h_i$, and strategy as the inputs (line 8), and stores the output (a cycle) in the list *list_cycles*. It should be noted that if $find\_cycle\_algo$ can not find any cycle given the inputs, it returns *null*.

(4) Transform each partition to the corresponding cycles. Given the pre-computed list *list_cycles*, for each *partition* belonging to the set *all_partitions* (line 10), $\mathcal{A}$ creates the partition's corresponding multiset of cycles, namely *Cycle*. To this end, it takes the following steps. For each $segment \in partition$, it fetches its corresponding cycle from the pre-computed list using the segment as the index and stores the corresponding cycle (if any) to the multiset *Cycle*. Note that the cycle *null* is also stored in *Cycle* (line 12).

(5) Merging. Algorithm *merge_segments_algo* takes *partition* and *Cycle* as inputs. Then, it merges some segments in the partition if needed, updates the corresponding cycles (if needed), and computes the corresponding frequencies. If merging is successful, the algorithm *merge_segments_algo* returns the multiset of corresponding cycles as C and their corresponding frequencies as F; otherwise, it returns *False* (line 14). Algorithm *merge_segments_algo* is explained in detail in Appendix J.1, and its pseudo-code is shown in Algorithm 10.

(6) Create the set of plausible traces. If the multiset C is valid (not *False*), the plausible trace *trace* will be created using the function *create_traces*. It takes F and C as the inputs and outputs the plausible trace denoted as $trace = \{(c_1, f_1), (c_2, f_2), \ldots, (c_x, f_x)\}$.

Then, it stores *trace* to the set of plausible traces denoted as $plaus\_cycle\_traces = \{trace_1, trace_2, \ldots, trace_z\}$ (lines 16 and 17). Note that each $trace_j$ in the set *plaus_cycle_traces* corresponds to $partition_j$ in the set of *all_partitions*. Having obtained the set *plaus_cycle_traces*, $\mathcal{A}$ selects the correct trace uniformly via the function *select_randomly* and finally assigns the correct trace to the corresponding driver i via the function *assign_trace*. In the following, we explain the *merge_segments_algo* algorithm used in the CD attack (Algorithm 9).

J.1 merge_segments_algo algorithm

The algorithm *merge_segments_algo* takes as inputs a partition and a multiset of cycles obtained by the CD attack (line 12 of Algorithm 9). Then, it merges some segments in the partition if needed, accordingly updates the multiset of cycles, and computes the corresponding frequencies. The algorithm's pseudo-code is shown in Algorithm 10. The full example of the algorithm is illustrated in Appendix K.

Core idea of the algorithm. Before explaining the algorithm in detail, we discuss *merging*, which is the main functionality of the algorithm *merge_segments_algo*. We said earlier that each segment includes different toll stations (without repetition of a toll station in a segment), which are the constituents of a cycle (if any). However, there would be cases where two or more repetitions of each toll point in a segment are required as the constituents of the cycle. This is because one or more toll points could be *revisited* when finding the cycle using the algorithm $find\_cycle\_algo$. In this case, the algorithm takes the following step. In the partition, the algorithm looks for a segment including one toll point equivalent to the toll point revisited in the cycle and then merges the toll point into the segment. The algorithm repeats the step until the toll points in the segment are equivalent to the ones used in the cycle. In the following, we give an example of merging.

Algorithm 10 The merge segments algorithm

Input: partition, *Cycle*
Output: F, C

```
1:  function MERGE_SEGMENTS_ALGO(partition, Cycle)
2:      for cycle ∈ Cycle, segment ∈ partition do
3:          if cycle == null and get_tolls(segment) ≠ 1 then
4:              return False
5:          end if
6:      end for
7:      for cycle ∈ Cycle, segment ∈ partition do
8:          if cycle ≠ null then
9:              if get_tolls(cycle) ≠ get_tolls(segment) then
10:                 R ← get_tolls(cycle) − get_tolls(segment)
11:                 for seg_one_toll ∈ partition do
12:                     if seg_one_toll ∈ R then
13:                         flag ← merge(segment, seg_one_toll)
14:                     end if
15:                 end for
16:                 if flag == False then
17:                     return False
18:                 else
19:                     C ← update(Cycle)
20:                     F ← compute_freq()
21:                 end if
22:             end if
23:         end if
24:     end for
25:     for cycle ∈ C do
26:         if cycle == null then
27:             return False
28:         end if
29:     end for
30:     return F, C
31: end function
```

Example of merging. Considering the segment $\{s_1, s_2, s_3\}$, the algorithm $find\_cycle\_algo$ finds the corresponding cycle where

the toll points s_1, s_2 and s_3 are constituents of the cycle. Let us say the found cycle is $h_i s_1 s_3 s_2 s_3 s_1 h_i$, where s_1 and s_3 have been visited twice. Hence, to form the cycle, the segment $\{s_1, s_2, s_3\}$ needs one more s_1 and one more s_3, for which the algorithm looks for two segments (if any), i.e., $\{s_1\}$ and $\{s_3\}$, in the corresponding partition and then merges them with the segment $\{s_1, s_2, s_3\}$, leading to the segment $\{s_1, s_2, s_3, s_1, s_3\}$. Now, the toll points in the segment $\{s_1, s_2, s_3, s_1, s_3\}$ are equivalent to the ones in the cycle $h_i s_1 s_3 s_2 s_3 s_1 h_i$. If either of the segments $\{s_1\}$ and $\{s_3\}$ could not be found, merging would not be possible, and the toll points in the segment would not form the corresponding cycle.

The details of the algorithm. In the following, we explain the algorithm's functionality in detail. We first explain how the algorithm detects an invalid multiset of cycles and then discuss the merging functionality.

Detection of an invalid multiset of cycles. Here, we discuss how the algorithm *merge_segments_algo* detects an invalid multiset of cycles, i.e., *Cycle*. The algorithm takes the following steps for each $cycle \in Cycle$ and its corresponding $segment \in partition$ (line 2).

(1) If $cycle \in Cycle$ (line 3) is null (that is, the toll points in the corresponding *segment* cannot lead to a cycle), it checks the number of toll points in the corresponding segment. If there is only one toll point in the segment $get\_tolls(segment) == 1$, it is likely that the *segment* will later be merged into another segment in the function *merge* (line 13). By merging, its corresponding cycle (which is null) will be removed from the multiset C, which does not make the multiset invalid. Hence, it continues the loop.
(2) Otherwise, if *cycle* is null and there are two or more toll points in its corresponding segment (line 3), the algorithm returns *False*. This is because the segments, including two or more toll points, will not be considered for merging, which makes the multiset invalid.

Merging functionality. Here, we discuss how the algorithm merges some segments if needed. The algorithm performs the following steps for each $cycle \in Cycle$ and its corresponding $segment \in partition$ (line 7).

(1) If the cycle is not null ($cycle \neq null$) (which is made from the toll points in the *segment*), the algorithm checks if any toll points in the *segment* are repeated in the cycle. To check, it executes the code $get\_tolls(cycle) \neq get\_tolls(segment)$ (line 9), where the function *get_tolls* returns the toll points used in a cycle or a segment. If the inequality holds (some toll points in the segment are repeated in the cycle), it inserts the repeated toll points in the set R.
(2) Then, for each segment, namely $seg\_one\_toll \in partition$, including only one toll point, it repeats this step. If the segment *seg_one_toll* is a toll point that belongs to the set of repeated/revisited toll points ($seg\_one\_toll \in R$), it calls the function *merge*, taking *segment* and *seg_one_toll* as the inputs and merges *seg_one_toll* into the *segment*. Finally, the function returns the $flag$ (lines 11, 12, and 13).
(3) If merging is complete ($flag \neq False$), meaning that the toll points in the segment (after merging) equal those used in the corresponding cycle, then the corresponding multiset *Cycle* should be updated (function *update*). This is because the function *merge* removes one or more segments from the partition and merges them with one or more segments, which changes the partition, and, accordingly, the corresponding multiset of cycles (*Cycle*) should be updated. Then, it computes the set of frequencies F via the function $compute\_freq$ (lines 19 and 20).
(4) If merging cannot be completed ($flag == False$), meaning that the toll points in the segment (after merging) are not equal to those used in the corresponding cycle, the algorithm returns *False* (lines 16 and 17).

Finally, in the loop (line 25), it checks if there is a null cycle in the updated multiset of cycles, indicating the presence of a segment where the toll points cannot form a cycle. If such a cycle exists, it returns *False*. Otherwise, it returns the multisets C and F.

J.2 The computational complexity of the CD attack

The CD attack's computational complexity mainly depends on different algorithms and functions as follows. (1) *TSD attack:* We discussed that the CD attack uses the TSD attack whose complexity is explained in Section 5.2.3. (2) *create_partitions:* The function's complexity is $O(2^m)$, where m is the size of the multiset [27]. (3) *compute_combinations:* The computational complexity of this function is exponential in terms of the number of visited toll points, shown in Formula 11. The $|S|$ in the formula should be substituted with the size of the function's input, i.e., $|visited\_tolls|$, resulting in $(2^{|visited\_tolls|} - 1)$. (4) *find_cycle_algo:* Since this algorithm (Algorithm 7) finds the cycle associated with each combination computed by *compute_combinations*, it should find $(2^{|visited\_tolls|} - 1)$ cycles in total. Given that the complexity of $find\_cycle\_algo$ for finding a cycle is $O(\omega)$, the complexity for finding the total cycles becomes $(2^{|visited\_tolls|} - 1) \cdot O(\omega)$. The complexity of the CD attack is the summation of the abovementioned complexities.

K FULL EXAMPLE

In the following, we give a concrete example to have a better understanding of the CD attack (Algorithm 9) and Algorithm 10 which is used by the attack. Note that the paragraphs' titles shown here (in bold) correspond to those mentioned in the explanation of the attack in Section J and the algorithm in Section J.1.

We consider a simple graph of a city's ETC system, including three toll stations $S = \{A, B, C\}$, and a driver's home location h. The assigned numbers to the edges show the distance between each of the two toll points. The graph is shown in Figure 7. Given that the output of the TSD attack is the $correct\_trace = \{(A, 2), (B, 2), (C, 1)\}$, Algorithm 9 achieves the CD goal in the following six steps.

(1) Create the multiset. The CD attack (Algorithm 9), using the function $create\_multiset$, computes the multiset $m\_visited\_tolls = \{A, A, B, B, C\}$.

(2) Create partitions. Then, it computes the set of all partitions *all_partitions* via the function *create_partitions*, taking the multiset *m_visited_tolls* as its input. There are nine different partitions, shown in Table 8.

(3) Create the pre-computed list of cycles. Before the attack pre-computes the list of cycles, it obtains the set of visited toll stations, which is $visited\_tolls = \{A, B, C\}$ (computed by *get_tolls*). The set is used as the input for the function *compute_combinations*, which computes all combinations of the visited toll points, shown in Table 9. Then, the attack finds the corresponding cycle for each combination using the algorithm $find\_cycle\_algo$. The algorithm uses the "shortest distance" strategy for finding the cycles, where a driver takes the shortest possible route to their destination. The resulting cycles are stored in the list *list_cycles*, shown in Table 9.

As shown in Table 9, the corresponding cycle of combination $\{B\}$ is *null*. This is because, given the nodes h and B (see Figure 7), no cycle can be found (to reach B from node h, node A is required, which is not in the combination).

(4) Transform each partition to its corresponding cycles. Then, the attack replaces each segment in the partition, as shown in Table 8, with its corresponding cycle from the list *list_cycles*, resulting in the multiset *Cycle* depicted in Table 8. For instance, in the first partition (refer to Table 8), the segments $\{A\}$, $\{A, B\}$, and $\{B, C\}$ are replaced with their respective cycles (as shown in Table 9): *hAh*, *hABAh*, and *null*. Consequently, the resulting multiset is $\{hAh, hABAh, null\}$.

(5) Merging. Then, the algorithm *merge_segments_algo* takes a partition and its corresponding set of cycles *Cycle* (Table 8) as the inputs and merges some of the segments (if needed) and computes the corresponding updated cycles and frequencies as follows. The results are shown in Table 10.

- **Detection of an invalid multiset of cycles:** For the multiset of partition and multiset of cycles in rows 1 and 7, the algorithm returns *False* since $get\_tolls(\{B, C\}) \neq 1$ (number of toll points in the segment $\{B, C\}$ is greater than one), and the segment's corresponding cycle is *null*. The algorithm *merge_segments_algo*'s output is shown in column "Validity of multiset" of Table 10. The symbol cross in the column "Validity of multiset" represents the algorithm's output, "False".
- **Merging functionality:** Concerning row 2, since the inequality $get\_tolls(hABCBAh) \neq get\_tolls(\{A, B, C\})$ holds, the algorithm merges the segments $\{A\}$ and $\{B\}$ (in the partition $\{\{A\}, \{A, B, C\}, \{B\}\}$) into the segment $\{A, B, C\}$, resulting in the update partition $\{A, B, C, B, A\}$. Due to the merging, the algorithm updates the multiset $Cycle$ = {hAh, **hABCBAh**, null} to $C = \{\mathbf{hABCBAh}\}$.
  Concerning row 3, since the condition $get\_tolls(hACAh) \neq get\_tolls(\{A, C\})$ holds, the algorithm looks for the segment $\{A\}$, in the partition $\{\{A, C\}, \{A, B\}, \{B\}\}$, to merge it with the segment $\{A, C\}$; but no segment $\{A\}$ can be found; hence, the algorithm returns *False*. For the same reason, the algorithm returns *False* concerning rows 5 and 6.
  Concerning row 4, since the condition $get\_tolls(hABAh) \neq get\_tolls(\{A, B\})$ holds, the algorithm merges the segment $\{A\}$ with the segment $\{AB\}$ (see the partition $\{\{A\}, \mathbf{\{A,B\}}, \{B\}, \{C\}\}$), resulting in the updated partition $\{\mathbf{\{A,B,A\}}, \{B\}, \{C\}\}$. However, as segments $\{B\}$ and $\{C\}$ lead to null cycles, the algorithm *merge_segments_algo* returns *False*. For the same reason, the algorithm returns *False* concerning row 8. Concerning row 9, the algorithm returns *False* as the segments $\{B\}$ and $\{C\}$ lead to null cycles. Note that if the segments were merged with other segments, the algorithm would not return *Fasle* as the segments would not exist after merging, and their corresponding null cycles would accordingly be removed (updated) from the corresponding multiset of cycles, i.e., *Cycle*.
  Finally, the algorithm *merge_segments_algo* returns the multisets $C = \{hABCBAh\}$ and $F = \{1\}$, as the outputs.

(6) Create the set of plausible traces. Having obtained the multisets C and F by *merge_segments_algo*, the CD attack creates the set of plausible traces denoted as $plaus\_cycle\_traces = \{trace_1\}$, where $trace_1 = \{(c_1 = hABCBAh, f_1 = 1)\}$. Since the set *plaus_cycle_traces* contains only one element, the adversary selects it as the correct trace.

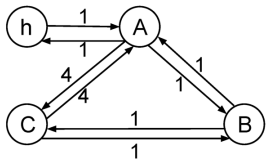


Figure 7: Example of an ETC system's graph, including three toll stations A, B, and C and the home location h

L RELATED WORK

Table 11 summarizes the comparison made between our work and [7] in Section 10 (attacks on privacy in ETC).

#	*partition*	*Cycle*
1	$\{\{A\},\{A,B\},\{B,C\}\}$	$\{hAh, hABAh, null\}$
2	$\{\{A\},\{A,B,C\},\{B\}\}$	$\{hAh, hABCBAh, null\}$
3	$\{\{A,C\},\{A,B\},\{B\}\}$	$\{hACAh, hABAh, null\}$
4	$\{\{A\},\{A,B\},\{B\},\{C\}\}$	$\{hAh, hABAh, null, null\}$
5	$\{\{A,B\},\{A,B,C\}\}$	$\{hABAh, hABCBAh\}$
6	$\{\{A,B\},\{A,B\},\{C\}\}$	$\{hABAh, hABAh, null\}$
7	$\{\{A\},\{A\},\{B\},\{B,C\}\}$	$\{hAh, hAh, null, null\}$
8	$\{\{A\},\{B\},\{B\},\{A,C\}\}$	$\{hAh, null, null, hACAh\}$
9	$\{\{A\},\{A\},\{B\},\{B\},\{C\}\}$	$\{hAh, hAh, null, null, null\}$

Table 8: Each row shows a partition and its corresponding set of cycles.

#	*combination*	*cycle*
1	$\{A\}$	*hAh*
2	$\{B\}$	*null*
3	$\{C\}$	*null*
4	$\{A,B\}$	*hABAh*
5	$\{A,C\}$	*hACAh*
6	$\{B,C\}$	*null*
7	$\{A,B,C\}$	*hABCBAh*

Table 9: Each row shows a combination of visited toll stations and the corresponding cycle.

#	*partition*	*Cycle*	Validity of the multiset	Updated *partition*	Updated *Cycle*	*C*	*F*
1	$\{\{A\},\{A,B\},\{B,C\}\}$	$\{hAh, hABAh, null\}$	×	×	×	×	0
2	$\{\{A\},$ **{A,B,C}**$,\{B\}\}$	$\{hAh,$ **hABCBAh**$, null\}$	✓	$\{\{$**ABCBA**$\}\}$	$\{$**hABCBAh**$\}$	$\{$**hABCBAh**$\}$	1
3	$\{$**{A,C}**, **{A,B}**$,\{B\}\}$	$\{$**hACAh**, **hABAh**$, null\}$	✓	×	×	×	0
4	$\{\{A\},$ **{A,B}**$,\{B\},\{C\}\}$	$\{hAh,$ **hABAh**$, null, null\}$	✓	$\{\{$**ABA**$\},\{B\},\{C\}\}$	$\{$**hABAh**$, null, null\}$	×	0
5	$\{$**{A,B}**, **{A,B,C}**$\}$	$\{$**hABAh**, **hABCBAh**$\}$	✓	×	×	×	0
6	$\{$**{A,B}**, **{A,B}**$,\{C\}\}$	$\{$**hABAh**, **hABAh**$, null\}$	✓	×	×	×	0
7	$\{\{A\},\{A\},\{B\},\{B,C\}\}$	$\{hAh, hAh, null, null\}$	×	×	×	×	0
8	$\{\{A\},\{B\},\{B\},$ **{A,C}**$\}$	$\{hAh, null, null,$ **hACAh**$\}$	✓	$\{\{B\},\{B\},\{$**ACA**$\}\}$	$\{null, null,$ **hACAh**$\}$	×	0
9	$\{\{A\},\{A\},\{B\},\{B\},\{C\}\}$	$\{hAh, hAh, null, null, null\}$	✓	×	×	×	0

Table 10: The table shows a concrete example of how Algorithm 10 performs. The symbol "cross" in the column "Validity of the multiset" represents that the multiset *Cycle* is invalid. The bold segments and cycles have resulted after the merging process (step 5 of the example). The columns *C* and *F* denote the algorithm's outputs.

Factors	Our studies	[7]
Core idea	Solving the SSP	Solving the SSP
Assumptions	The assumption is weak as the adversary accesses the subset of the information available to the TSP.	The assumption is strong as, in the first place, the anonymous trips are part of the adversary's knowledge.
Adversary	A weak adversary is defined.	A strong adversary is used.
Goals	Toll station disclosure, Cycle disclosure	Obtaining the trips associated with each driver.
Real settings	Real settings are used, including the map, toll prices, number of toll stations, billing period, and distribution of wallet balances.	Synthetic data are used.
Heuristics	Heuristics are based on drivers' behavior and the road infrastructure.	Heuristics are based on drivers' behavior and the road infrastructure.
SSP solver	DOcplex is applied, which is more convenient.	Pisinger's algorithm is applied, which is complicated to use.
Privacy	The parameters impacting a driver's privacy are investigated, such as toll prices, wallet balances, etc.	The parameters impacting a driver's privacy are not discussed.

Table 11: The table compares our work with the most relevant one, i.e., [7] in terms of different factors.