Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2024) 32: 198 – 218
© TÜBİTAK
doi:10.55730/1300-0632.4063

Longitudinal attacks against iterative data collection with local differential privacy

Mehmet Emre GÜRSOY*
Department of Computer Engineering, Faculty of Engineering, Koç University, İstanbul, Turkiye,

Received: 27.08.2023 • **Accepted/Published Online:** 05.01.2024 • **Final Version:** 07.02.2024

Abstract: Local differential privacy (LDP) has recently emerged as an accepted standard for privacy-preserving collection of users' data from smartphones and IoT devices. In many practical scenarios, users' data needs to be collected repeatedly across multiple iterations. In such cases, although each collection satisfies LDP individually by itself, a longitudinal collection of multiple responses from the same user degrades that user's privacy. To demonstrate this claim, in this paper, we propose longitudinal attacks against iterative data collection with LDP. We formulate a general Bayesian adversary model, and then individually show the application of this adversary model on six popular LDP protocols: GRR, BLH, OLR, RAPPOR, OUE, and SS. We experimentally demonstrate the effectiveness of our attacks using two metrics, three datasets, and various privacy and domain parameters. The effectiveness of our attacks highlights the privacy risks associated with longitudinal data collection in a practical and quantifiable manner and motivates the need for appropriate countermeasures.

Key words: Local differential privacy, cybersecurity, Bayesian inference, Internet of things

1. Introduction

In recent years, local differential privacy (LDP) has emerged as a popular standard for privacy-preserving collection of users' data [1–3]. To preserve privacy, LDP enables each user's data to be locally perturbed on their own device (e.g., smartphone, IoT device) before being sent to the data collector. LDP has been applied in many contexts related to the Internet of things (IoT) and cyber-physical systems, including but not limited to geolocation data [4, 5], indoor positioning [6, 7], health data and wearables [8–10], Internet of vehicles [11], cybersecurity [12], and smart meters [13, 14]. LDP has also seen real-world deployment, including Google's RAPPOR for analyzing Chrome browser settings [15], Apple's implementation in iOS to collect popular emojis and trending words for typing recommendation [16, 17], and Microsoft's implementation in Windows 10 for collecting application telemetry [18].

On the other hand, in many practical scenarios, the data collector needs to collect users' data periodically and repeatedly across multiple iterations (with timestamps). One example of such iterative data collection is LDP deployments in the industry, e.g., Apple collects Safari data twice every day and Health data once every day [19], and Microsoft collects telemetry statistics once every 6 h [1, 18]. Another example is locally private federated learning, in which a user iteratively contributes to the training of a global machine learning model across many rounds [20, 21]. In such cases, LDP protocols are used for multiple iterations of data collection,

*Correspondence: emregursoy@ku.edu.tr

whereas the underlying data being collected remains mostly static. For example, a user's browser homepage, most visited websites, or Health app settings on their smartphone are unlikely to change every day. In this case, although each perturbed response satisfies ε-LDP individually by itself, we conjecture that *longitudinal* collection of multiple responses from the same user will degrade that user's privacy.

In order to demonstrate this claim, in this paper we propose longitudinal attacks against iterative data collection with LDP. The added privacy risks of longitudinal data collection have been recognized in the past LDP literature [15, 18, 22, 23]; however, to the best of our knowledge, there does not exist a practical attack which can experimentally and quantifiably measure these privacy risks. Our paper fills this gap. We first formulate a Bayesian adversary model that observes multiple perturbed responses from the user (each of which satisfies LDP), and aims to predict the user's true value using the observed responses and Bayesian inference. Considering that the inference process is dependent on the specifics of the LDP protocol, we then apply our adversary model to six popular LDP protocols: GRR, BLH, OLH, RAPPOR, OUE, and SS. For each protocol, we algorithmically describe the inference attack step by step. In addition, we propose two metrics for measuring the success of the attack (formally stated in Sec. 3.3): Adversarial success rate (ASR) and group inference rate (GIR). ASR measures the ratio of exactly correct predictions made by the adversary, whereas GIR measures the adversary's ability to predict whether the user's true value is among a group of predefined sensitive values.

We experimentally demonstrate the effectiveness of our attacks using three datasets (MSNBC, Kosarak, Uniform), under varying privacy budgets ε, number of observations n, and domain and sensitive group sizes. The key take-away messages from our experiments are as follows: (i) Our longitudinal attacks are effective. For common values of ε such as $\varepsilon = 2$ or 4, although ASR is typically below 0.2 with $n = 1$ observation, when many observations are made longitudinally (e.g., $n = 9$ or 11), ASR can exceed 0.8 which shows that users effectively get very little privacy protection. (ii) The vulnerability of each LDP protocol to our attack depends on multiple factors such as ε, domain size, and sensitive group size. For example, GRR protocol shows higher vulnerability compared to other protocols for high ε values (such as $\varepsilon = 4$ or 6), but lower vulnerability for low ε values. (iii) Increasing the sensitive group size increases attack effectiveness when measured by the GIR metric, whereas increasing the domain size decreases attack effectiveness. (iv) Our Bayesian attack always outperforms a random attacker model by a large margin.

The remainder of this paper is organized as follows. In Section 2, we give the preliminaries regarding LDP and LDP protocols. In Section 3, we give the general Bayesian attack formulation, describe how the attack is applied individually to each protocol, and define the attack success metrics. In Section 4, we provide our experimental analysis and discuss the results. Finally, Section 5 concludes the paper.

2. Preliminaries

2.1. Local differential privacy

Local differential privacy (LDP) has recently emerged as a popular standard for privacy-preserving data collection. In a typical LDP setup, there exists a population of users (*clients*) and a data collector (*server*). We denote the user population by $\mathcal{P}$ and the domain of users' possible values by $\mathcal{D}$. For user $u \in \mathcal{P}$, we denote this user's true value by v_u, such that $v_u \in \mathcal{D}$. While the server wants to collect statistics regarding users' values, in order to protect privacy, each user perturbs his/her v_u using a randomized mechanism $\mathcal{M}$ and shares the perturbed response with the server. For mechanism $\mathcal{M}$ to satisfy LDP, the following definition must hold.

Definition 1 (ε-LDP) *A randomized mechanism $\mathcal{M}$ satisfies ε-local differential privacy (ε-LDP), where $\varepsilon > 0$, if and only if for any two inputs $v_1, v_2 \in \mathcal{D}$ we have:*

$$\forall y \in Range(\mathcal{M}): \quad \frac{Pr[\mathcal{M}(v_1) = y]}{Pr[\mathcal{M}(v_2) = y]} \le e^{\varepsilon} \tag{1}$$

where $Range(\mathcal{M})$ denotes the set of all possible outputs of $\mathcal{M}$.

Here, ε is a key parameter that determines the privacy level, also commonly known as the *privacy budget*. By definition of LDP, lower ε yields stronger privacy whereas higher ε yields weaker privacy.

Several LDP protocols have been developed in the literature [2, 15, 24, 25], which serve as building blocks for more complex end applications with richer capabilities. Depending on the data syntax, application semantics, domain $\mathcal{D}$, budget ε and other factors, different protocols may be preferred in different applications to maximize accuracy, reduce user-side computation effort or minimize user-server communication cost [1, 2]. In the next section, we summarize six protocols that have been commonly used in the related literature [1, 26–29]: GRR, BLH, OLH, RAPPOR, OUE, SS. We will use these protocols to demonstrate our longitudinal attacks.

2.2. LDP protocols

To summarize the technical descriptions of the protocols, we follow an approach similar to [2, 28] and specify an LDP protocol in 3 fundamental steps: Encode, Perturb, Aggregate. The first two steps occur on the user-side, whereas the third step occurs on the server-side after perturbed responses are collected from users. For protocol Prot, we denote its encoding step by Θ_{Prot}, its perturbation step by Ψ_{Prot}, and its aggregation step by Φ_{Prot}. Collectively, the three steps $\langle \Theta_{\text{Prot}}, \Psi_{\text{Prot}}, \Phi_{\text{Prot}} \rangle$ are sufficient to describe how Prot works.

Generalized Randomized Response (GRR) is an extension of the *randomized response* survey collection technique. GRR generalizes and extends this technique to support nonbinary finite $\mathcal{D}$ and arbitrary ε. The encoding step of GRR is trivial: $\Theta_{\text{GRR}}(v_u) = v_u$. Then, the perturbation step Ψ_{GRR} takes as input v_u and outputs $y_u \in \mathcal{D}$ with the following probabilities:

$$\Pr[\Psi_{\text{GRR}}(v_u, \varepsilon) = y_u] = \begin{cases} p = \frac{e^{\varepsilon}}{e^{\varepsilon}+|\mathcal{D}|-1} & \text{if } y_u = v_u \\ q = \frac{1}{e^{\varepsilon}+|\mathcal{D}|-1} & \text{if } y_u \neq v_u \end{cases} \tag{2}$$

where $|\mathcal{D}|$ denotes domain size. This satisfies ε-LDP since $\frac{p}{q} = e^{\varepsilon}$. The user sends y_u to the server.

On the server side, upon receiving perturbed responses from all users, to perform estimation for some value $v \in \mathcal{D}$ the server first finds $\widehat{C}(v)$: total number of users who reported v as their perturbed output. Then, Φ_{GRR} computes the estimate $\bar{C}(v)$ as:

$$\bar{C}(v) = \Phi_{\text{GRR}}(\widehat{C}(v), \varepsilon) = \frac{\widehat{C}(v) - |\mathcal{P}| \cdot q}{p - q} \tag{3}$$

Binary local hashing (BLH) is similar to [24], but instead of expensive matrix projection and multiplication operations, it uses a logically equivalent construction by drawing a random hash function H from a universal hash function family $\mathcal{H}$. It was noted in [30] that Hadamard transform is also similar in essence to BLH.

Let $\mathcal{H}$ be a universal hash function family such that each hash function $H \in \mathcal{H}$ maps a value from $\mathcal{D}$ into one bit, i.e. $H : \mathcal{D} \to \{0,1\}$. For encoding, each user u first draws a hash function uniformly randomly from $\mathcal{H}$, i.e. $H_u \leftarrow_{\$} \mathcal{H}$. Then, user's bit b_u is computed by: $b_u = H_u(v_u)$. Overall, the output of the encoding step of BLH is: $\Theta_{\text{BLH}}(v_u) = \langle H_u, b_u \rangle$. The perturbation step Ψ_{BLH} takes as input b_u and outputs b'_u with the following probabilities:

$$\Pr[b'_u = 1] = \Pr[\Psi_{\text{BLH}}(b_u, \varepsilon) = 1] = \begin{cases} \frac{e^\varepsilon}{e^\varepsilon+1} & \text{if } b_u = 1 \\ \frac{1}{e^\varepsilon+1} & \text{if } b_u = 0 \end{cases} \tag{4}$$

The user sends tuple $\langle H_u, b'_u \rangle$ to the server.

The server receives tuples $\langle H_u, b'_u \rangle$ from users $u \in \mathcal{P}$. When performing estimation for $v \in \mathcal{D}$, the server first computes $Sup(v)$, which denotes the total number of tuples for which the condition $b'_u = H_u(v)$ holds. Then, Φ_{BLH} computes the estimate $\bar{C}(v)$ as:

$$\bar{C}(v) = \Phi_{\text{BLH}}(Sup(v), \varepsilon) = \frac{(e^\varepsilon + 1) \cdot (2 \cdot Sup(v) - |\mathcal{P}|)}{e^\varepsilon - 1} \tag{5}$$

Optimized local hashing (OLH) differs from BLH in that the output spaces of hash functions in family $\mathcal{H}$ are now g-ary instead of binary, where $g \geq 2$ is an adjustable parameter of the protocol. Hence, OLH enables users to encode their v_u into an integer $[1, g]$ instead of a single bit. The benefit of g-ary encoding is to combat BLH's high utility loss caused by binary encoding, especially when ε and $\mathcal{D}$ are large [2]. The default value of g is $g = e^\varepsilon + 1$ as derived and used in [2, 27], which is also the default value we will assume in the rest of the paper.

Let $\mathcal{H}$ be a universal hash function family where each $H \in \mathcal{H}$ maps a value from $\mathcal{D}$ into an integer in $[1, g]$, i.e. $H : \mathcal{D} \to [1, g]$. For encoding, each user u first draws a hash function uniformly randomly from $\mathcal{H}$, i.e. $H_u \leftarrow_{\$} \mathcal{H}$. Then, user computes integer x_u as: $x_u = H_u(v_u)$. Overall, the output of the encoding step of OLH is: $\Theta_{\text{OLH}}(v_u) = \langle H_u, x_u \rangle$. The perturbation step Ψ_{OLH} takes as input x_u and outputs perturbed integer $x'_u \in [1, g]$ with the following probabilities:

$$\Pr[x'_u = i] = \Pr[\Psi_{\text{OLH}}(x_u, \varepsilon) = 1] = \begin{cases} \frac{e^\varepsilon}{e^\varepsilon+g-1} & \text{if } x_u = i \\ \frac{1}{e^\varepsilon+g-1} & \text{if } x_u \neq i \end{cases} \tag{6}$$

The user sends tuple $\langle H_u, x'_u \rangle$ to the server.

The server receives tuples $\langle H_u, x'_u \rangle$ from users $u \in \mathcal{P}$. When performing estimation for $v \in \mathcal{D}$, the server first computes $Sup(v)$, which denotes the total number of tuples for which the condition $x'_u = H_u(v)$ holds. Then, Φ_{OLH} computes the estimate $\bar{C}(v)$ as:

$$\bar{C}(v) = \Phi_{\text{OLH}}(Sup(v), \varepsilon) = \frac{(e^\varepsilon + g - 1) \cdot (g \cdot Sup(v) - |\mathcal{P}|)}{(e^\varepsilon - 1) \cdot (g - 1)} \tag{7}$$

RAPPOR was originally developed by Google's researchers and implemented in Chrome [15, 25]. In RAPPOR, the user's value is encoded into a bitvector and the bitvector is perturbed in a randomized fashion to satisfy LDP. Here, we give a general version of RAPPOR with parameter Δ that is determined by how many 1 bits can exist in the bitvector. If unary (one-hot) encoding is used as in [2, 12], in which only one 1 bit is allowed in the bitvector, then $\Delta = 2$. If more complex encodings are used (such as Bloom filters, set-valued encoding or graph neighbor lists as in [25, 31, 32]), in which at most h bits are allowed to be 1, then $\Delta = 2h$.

The encoding step of RAPPOR takes the user's true value v_u and encodes it into a bitvector B_u, i.e. $\Theta_{\text{RAPPOR}}(v_u) = B_u$. In case of unary encoding, B_u is constructed as follows:

$$\forall_{i\in[1,|\mathcal{D}|]}: \quad B_u[i] = \begin{cases} 1 & \text{if } v_u = i \\ 0 & \text{if } v_u \neq i \end{cases} \tag{8}$$

The perturbation step Ψ_{RAPPOR} takes B_u as input and produces a perturbed bitvector B'_u. This perturbed bitvector is constructed by considering each bit in B_u one by one, and either keeping the existing bit or flipping it with probabilities controlled by ε and Δ:

$$\Pr[B'_u[i] = 1] = \begin{cases} \frac{e^{\varepsilon/\Delta}}{e^{\varepsilon/\Delta}+1} & \text{if } B_u[i] = 1 \\ \frac{1}{e^{\varepsilon/\Delta}+1} & \text{if } B_u[i] = 0 \end{cases} \tag{9}$$

The user sends B'_u to the server.

The server receives perturbed bitvectors B'_u from users $u \in \mathcal{P}$. When performing estimation for index $i \in [1, |B|]$, the server first finds $\tilde{C}[i]$ as:

$$\tilde{C}[i] = \sum_{u\in\mathcal{P}} B'_u[i] \tag{10}$$

Then, Φ_{RAPPOR} computes the estimate $\bar{C}(i)$ as:

$$\bar{C}(i) = \Phi_{\text{RAPPOR}}(\tilde{C}[i], \varepsilon) = \frac{\tilde{C}[i] + |\mathcal{P}| \cdot (\alpha - 1)}{2\alpha - 1} \tag{11}$$

where α is the bit keeping probability: $\alpha = \frac{e^{\varepsilon/\Delta}}{e^{\varepsilon/\Delta}+1}$.

Optimized unary encoding (OUE) has the same encoding step as RAPPOR with unary encoding, i.e. Θ_{OUE} is equivalent to Equation 8. However, OUE's perturbation step Ψ_{OUE} is different than Ψ_{RAPPOR} – it treats the 0 and 1 bits asymmetrically. More concretely, Ψ_{OUE} takes bitvector B_u as input and produces perturbed bitvector B'_u according to the following probabilities:

$$\Pr[B'_u[i] = 1] = \begin{cases} \frac{1}{2} & \text{if } B_u[i] = 1 \\ \frac{1}{e^{\varepsilon}+1} & \text{if } B_u[i] = 0 \end{cases} \tag{12}$$

The user sends B'_u to the server.

The server receives perturbed bitvectors B'_u from users $u \in \mathcal{P}$. When performing estimation for index $i \in [1, |B|]$, the server first finds $\tilde{C}[i]$ in the same way as in Equation 10. Then, Φ_{OUE} computes the estimate $\bar{C}(i)$ as:

$$\bar{C}(i) = \Phi_{\text{OUE}}(\tilde{C}[i], \varepsilon) = \frac{2 \cdot \big((e^{\varepsilon} + 1) \cdot \tilde{C}[i] - |\mathcal{P}|\big)}{e^{\varepsilon} - 1} \tag{13}$$

Subset selection (SS). In SS, each user u sends a randomized subset of values Z_u to the server. The size of the subset, denoted by $k = |Z_u|$ is a parameter of the protocol. According to [30, 33], the default value of k is $k = \frac{|\mathcal{D}|}{e^{\varepsilon}+1}$.

On the user side, Z_u is first initialized as an empty set. Ψ_{SS} adds v_u to Z_u with probability $\frac{k \cdot e^{\varepsilon}}{k \cdot e^{\varepsilon} + |\mathcal{D}| - k}$. Then, Ψ_{SS} constructs the remainder of Z_u as follows:

- If v_u was added to Z_u in the previous step, then $k-1$ items are sampled from $\mathcal{D} \setminus \{v_u\}$ uniformly at random without replacement, and they are added to Z_u.
- If v_u was not added to Z_u in the previous step, then k items are sampled from $\mathcal{D} \setminus \{v_u\}$ uniformly at random without replacement, and they are added to Z_u.

The user sends the resulting Z_u to the server.

The server receives randomized sets Z_u from users $u \in \mathcal{P}$. The server defines two constants σ_k and θ_k:

$$\sigma_k = \frac{ke^{\varepsilon}}{ke^{\varepsilon} + |\mathcal{D}| - k} \qquad \theta_k = \frac{(k-1)(ke^{\varepsilon}) + (|\mathcal{D}| - k)k}{(|\mathcal{D}| - 1)(ke^{\varepsilon} + |\mathcal{D}| - k)}$$

To perform estimation for value $v \in \mathcal{D}$, the server computes $Sup(v)$ as the total number of users whose reported set Z_u contains v. Then, Φ_{SS} computes the estimate $\bar{C}(v)$ as:

$$\bar{C}(v) = \Phi_{\text{SS}}(Sup(v), \varepsilon) = \frac{Sup(v) - |\mathcal{P}| \cdot \theta_k}{\sigma_k - \theta_k} \tag{14}$$

3. Longitudinal attacks against LDP

3.1. General attack formulation

We formulate our longitudinal attack against iterative data collection using a Bayesian adversary $\mathcal{A}$. This adversary can be the server (i.e. data collector), a man-in-the-middle who observes the communication between a user and the server, or a third-party analyst with whom the collected data is shared. For user u, let $\mathcal{O}_u$ denote the perturbed responses $\mathcal{A}$ has observed from u. Since we study *iterative* data collection, $\mathcal{A}$ is assumed to observe multiple perturbed responses from the user. We denote the number of observations by $n = |\mathcal{O}_u|$. We extend the notation from Section 2.2 and write superscripts on protocol outputs (ranging from 1 to n) to denote the observation number. For example, in case of GRR we write: $\mathcal{O}_u = \{y_u^1, y_u^2, ..., y_u^n\}$; in case of RAPPOR we write: $\mathcal{O}_u = \{B_u^1, B_u^2, ..., B_u^n\}$; and in case of OLH we write: $\mathcal{O}_u = \{\langle H_u^1, x_u^1 \rangle, \langle H_u^2, x_u^2 \rangle, ..., \langle H_u^n, x_u^n \rangle\}$.

Armed with $\mathcal{O}_u$, the goal of $\mathcal{A}$ is to correctly predict v_u. Denoting $\mathcal{A}$'s prediction by v_u^p, the Bayesian strategy to compute v_u^p can be derived using the Bayes theorem:

$$v_u^p = \underset{\hat{v} \in \mathcal{D}}{\operatorname{argmax}} \ \Pr[\hat{v} | \mathcal{O}_u] = \underset{\hat{v} \in \mathcal{D}}{\operatorname{argmax}} \ \frac{\Pr[\mathcal{O}_u | \hat{v}] \cdot \Pr[\hat{v}]}{\Pr[\mathcal{O}_u]} = \underset{\hat{v} \in \mathcal{D}}{\operatorname{argmax}} \ \Pr[\mathcal{O}_u | \hat{v}] \cdot \Pr[\hat{v}] \tag{15}$$

$$\propto \underset{\hat{v} \in \mathcal{D}}{\operatorname{argmax}} \ \Pr[\mathcal{O}_u | \hat{v}] \tag{16}$$

The computation of $\Pr[\mathcal{O}_u | \hat{v}]$ is protocol-dependent and it varies from protocol to protocol. In the next section, we describe how to compute it for each of the protocols given in Section 2.2.

3.2. Applying the attack to individual LDP protocols

The computation of $\Pr[\mathcal{O}_u | \hat{v}]$ is dependent on Θ_{PROT} and Ψ_{PROT} functions of each protocol, since these functions determine the probabilities of obtaining certain perturbed responses from $\hat{v}$. Therefore, we need to analyze the

Θ_{PROT} and Ψ_{PROT} functions of each protocol given in Section 2.2 (GRR, BLH, OLH, RAPPOR, OUE, and SS) one by one, and describe how to apply $\mathcal{A}$ to each protocol.

Application to GRR: In case of GRR, the output of Ψ_{GRR} is a value from the original $\mathcal{D}$. Therefore, we have: $\mathcal{O}_u = \{y_u^1, y_u^2, ..., y_u^n\}$ such that $y_u^i \in \mathcal{D}$ for all i. Next, we observe from Equation 2 that $p > q$, and consequently, $\Pr[y_u^i = v_u] > \Pr[y_u^i = v']$ for all $v' \in \mathcal{D} \setminus \{v_u\}$. Since v_u is more likely to be observed in $\mathcal{O}_u$ than any other $v' \in \mathcal{D}$, the value of $\hat{v}$ that maximizes $\Pr[\mathcal{O}_u|\hat{v}]$ can be found by counting the occurrences of each distinct v in $\mathcal{O}$, and assigning the one that has the highest number of occurrences as v_u^p. The pseudocode of this approach is given in Algorithm 1.

Algorithm 1 Attack application to GRR protocol.

```
1: function AttackGRR($\mathcal{O}_u = \{y_u^1, y_u^2, ..., y_u^n\}$)
2:     for $v \in \mathcal{D}$ do
3:         Initialize $count(v) = 0$
4:     for $i \in [1, n]$ do
5:         Increment $count(y_u^i) = count(y_u^i) + 1$
6:     Find $v^* = \underset{v}{\operatorname{argmax}}\ count(v)$
7:     return $v^*$
```

Application to BLH and OLH: Since BLH and OLH both use local hashing, we describe the attack strategy for them together. For ease of explanation, we follow OLH's notation and terminology. However, we note that BLH can be parsed as an instance of our attack explanation with a binary hash ($g = 2$) instead of OLH's g-ary hash.

In case of OLH, we have: $\mathcal{O}_u = \{\langle H_u^1, x_u^1\rangle, \langle H_u^2, x_u^2\rangle, ..., \langle H_u^n, x_u^n\rangle\}$ such that each H_u^i is a random g-ary hash function from a universal hash function family and x_u^i is an integer between $[1, g]$. We make two observations from Ψ_{OLH}. First, for all pairs of values $v_j, v_k \in \mathcal{D}$ such that $H_u(v_j) = H_u(v_k)$, the following holds: $\Pr[x_u^i|v_j] = \Pr[x_u^i|v_k]$. In other words, v_j and v_k have an equal probability of resulting in x_u^i. Second, consider that a certain x_u^i was reported to the server, and we divide $\mathcal{D}$ into two disjoint subsets $\mathcal{D}_1, \mathcal{D}_2$ such that $\mathcal{D}_1 = \{v|v \in \mathcal{D}, H_u^i(v) = x_u^i\}$ and $\mathcal{D}_2 = \mathcal{D} \setminus \mathcal{D}_1$. In that case, by construction of Ψ_{OLH} in Equation 6, the following inequality regarding the user's true value v_u must hold: $\Pr[v_u \in \mathcal{D}_1] > \Pr[v_u \in \mathcal{D}_2]$.

Combining these two observations, we apply $\mathcal{A}$ to OLH as shown in Algorithm 2. For $i \in [1, n]$, we consider each $\langle H_u^i, x_u^i\rangle$ pair individually, and construct the subset of $\mathcal{D}$ such that $\{v|v \in \mathcal{D}, H_u^i(v) = x_u^i\}$ under each pair. We call this subset $\mathcal{D}_{sub}$. According to the second observation, $\Pr[v_u \in \mathcal{D}_{sub}] > \Pr[v_u \in \mathcal{D} \setminus \mathcal{D}_{sub}]$, therefore $\mathcal{A}$ increases the scores of values in $\mathcal{D}_{sub}$. The fact that scores of each value in $\mathcal{D}_{sub}$ should be increased equally follows from the first observation. There are two options regarding how much the score of each value should be increased: (i) inversely proportional to the size of $\mathcal{D}_{sub}$, e.g., $\frac{1}{|\mathcal{D}_{sub}|}$ so that the probability is uniformly distributed across all values in $\mathcal{D}_{sub}$, or (ii) by a fixed constant c across all iterations, regardless of $|\mathcal{D}_{sub}|$. We choose the second option in our attack because the size of $\mathcal{D}_{sub}$ often changes in different iterations ($i \in [1, n]$), and this may cause certain values' scores to be unfairly high if they happened to be in $\mathcal{D}_{sub}$ in an iteration that had small $|\mathcal{D}_{sub}|$ by chance. Such situations would negatively impact the overall success rate of our attack. The second option does not suffer from this problem since scores are increased by a fixed constant c (e.g., $c = 1$) that stays the same across all i. In fact, we implemented and empirically tested both options to validate this intuition.

Algorithm 2 Attack application to BLH and OLH protocols

```
1: function AttackLH($\mathcal{O}_u = \{\langle H_u^1, x_u^1\rangle, ..., \langle H_u^n, x_u^n\rangle\}$)
2:     Let $c$ be a global constant such as $c = 1$ (see main text)
3:     for $v \in \mathcal{D}$ do
4:         Initialize $score(v) = 0$
5:     for $i \in [1, n]$ do
6:         Initialize $\mathcal{D}_{sub} = \{\}$
7:         for $v \in \mathcal{D}$ do
8:             if $H_u^i(v) = x_u^i$ then
9:                 $\mathcal{D}_{sub} = \mathcal{D}_{sub} \cup \{v\}$
10:        for $v \in \mathcal{D}_{sub}$ do
11:            $score(v) = score(v) + c$
12:    Find $v^* = \underset{v}{\operatorname{argmax}}\ score(v)$
13:    return $v^*$
```

Application to RAPPOR: The output of Ψ_{RAPPOR} is a bitvector, therefore we have: $\mathcal{O}_u = \{B_u^1, B_u^2, ..., B_u^n\}$. By design of RAPPOR, the decision of Ψ_{RAPPOR} to keep or flip each bit $j \in [1, |B|]$ is independent of other bits; in addition, the decision at each iteration $i \in [1, n]$ is also independent of the other iterations. Thus, for RAPPOR with unary encoding, if we consider a particular index j and a particular iteration i, we can derive from Equations 8 and 9 that:

$$\Pr[B_u^i[j] = 1 | v_u = j] = \frac{e^{\varepsilon/2}}{e^{\varepsilon/2} + 1} \tag{17}$$

$$\Pr[B_u^i[j] = 0 | v_u = j] = \frac{1}{e^{\varepsilon/2} + 1} \tag{18}$$

Then, taking into account the independence of the iterations $i \in [1, n]$:

$$\Pr[\mathcal{O}_u | v_u = j] = \prod_{i=1}^{n} \Pr[B_u^i | v_u = j] \tag{19}$$

Indeed, this is the derivation used in the function shown in Algorithm 3. The inner *for* loop computes the score of a particular j value using Equation 19, which are internally determined by Equations 17 and 18. The outer *for* loop ensures scores are calculated for all possible j. Finally, the value v^* that corresponds to the index with the highest score is returned by the algorithm.

Algorithm 3 Attack application to RAPPOR protocol

```
1: function AttackRAPPOR($\mathcal{O}_u = \{B_u^1, B_u^2, ..., B_u^n\}$)
2:     Initialize vector $scores$ of length $|B|$
3:     For all $j \in [1, |B|]$, initialize $scores[j] = 1$
4:     for $j \in [1, |B|]$ do
5:         for $i \in [1, n]$ do
6:             if $B_u^i[j] = 1$ then
7:                 $scores[j] = scores[j] \cdot \frac{e^{\varepsilon/2}}{e^{\varepsilon/2}+1}$
8:             else
9:                 $scores[j] = scores[j] \cdot \frac{1}{e^{\varepsilon/2}+1}$
10:    Find $v^* = \underset{j}{\text{argmax}}\ scores[j]$
11:    return $v^*$
```

Application to OUE: Similar to RAPPOR, in OUE we have: $\mathcal{O}_u = \{B_u^1, B_u^2, ..., B_u^n\}$. Also similar to RAPPOR, the decision of Ψ_{OUE} to keep or flip each bit $j \in [1, |B|]$ is independent of other bits, and the decision at each iteration $i \in [1, n]$ is independent of other iterations. However, the key difference of OUE compared to RAPPOR is caused by the perturbation probabilities of Ψ_{OUE}. It can be observed from Equation 12 that:

$$\Pr[B_u^i[j] = 1 | v_u = j] = \Pr[B_u^i[j] = 0 | v_u = j] = \frac{1}{2} \tag{20}$$

since the original 1 bit is kept or flipped with 1/2 probability. Hence, as opposed to RAPPOR, conditioning the probabilities on $v_u = j$ is not effective in OUE. Instead, we observe from Equation 12 that the original 0 bits are kept with a large probability but flipped with a low probability. Therefore, we can condition on $v_u \neq j$ and obtain:

$$\Pr[B_u^i[j] = 1 | v_u \neq j] = \frac{1}{e^{\varepsilon} + 1} \tag{21}$$

$$\Pr[B_u^i[j] = 0 | v_u \neq j] = \frac{e^{\varepsilon}}{e^{\varepsilon} + 1} \tag{22}$$

Across $i \in [1, n]$, the following property holds:

$$\Pr[\mathcal{O}_u | v_u \neq j] = \prod_{i=1}^{n} \Pr[B_u^i | v_u \neq j] \tag{23}$$

This derivation results in the attack strategy used in Algorithm 4. Instead of RAPPOR's attack which aims to keep scores proportional to the likelihood of each index $j \in [1, |B|]$ being originally 1 and choosing the index with the highest score at the end, OUE's attack maintains vector *scores_zero* which stores the likelihood that the corresponding index is originally 0. The internal score multiplications follow from Equations 21 and 22. At the end, whichever index has lowest likelihood of being 0 is the index whose likelihood of being 1 is highest, therefore an argmin is performed.

Algorithm 4 Attack application to OUE protocol

```
1: function AttackOUE($\mathcal{O}_u = \{B_u^1, B_u^2, ..., B_u^n\}$)
2:     Initialize vector $scores\_zero$ of length $|B|$
3:     For all $j \in [1, |B|]$, initialize $scores\_zero[j] = 1$
4:     for $j \in [1, |B|]$ do
5:         for $i \in [1, n]$ do
6:             if $B_u^i[j] = 1$ then
7:                 $scores\_zero[j] = scores\_zero[j] \cdot \frac{1}{e^\varepsilon + 1}$
8:             else
9:                 $scores\_zero[j] = scores\_zero[j] \cdot \frac{e^\varepsilon}{e^\varepsilon + 1}$
10:    Find $v^* = \underset{j}{\text{argmin}}\ scores\_zero[j]$
11:    return $v^*$
```

Application to SS: In case of SS, we have: $\mathcal{O}_u = \{Z_u^1, Z_u^2, ..., Z_u^n\}$ where each Z_u^i is a subset of values from $\mathcal{D}$. From the definition of Ψ_{SS}, we know that v_u is added to Z_u with probability $\frac{k \cdot e^\varepsilon}{k \cdot e^\varepsilon + |\mathcal{D}| - k}$. In addition, we can use the definition of Ψ_{SS} to compute that for any $v \in \mathcal{D} \setminus \{v_u\}$, the probability that v is added to Z_u is: $\frac{ke^\varepsilon}{ke^\varepsilon + |\mathcal{D}| - k} \cdot \frac{k-1}{|\mathcal{D}|-1} + \frac{|\mathcal{D}|-k}{ke^\varepsilon + |\mathcal{D}| - k} \cdot \frac{k}{|\mathcal{D}|-1}$. Considering that k is the subset size that must satisfy $k \leq |\mathcal{D}|$, we find that the prior probability is always larger than the latter. Hence, we establish that it is not possible for any fake value's probability of being added to Z_u to be higher than the probability that v_u is added to Z_u.

Next, we observe that the only way v_u is added to Z_u is if Ψ_{SS} adds v_u with probability $\frac{k \cdot e^\varepsilon}{k \cdot e^\varepsilon + |\mathcal{D}| - k}$. If v_u is not added to Z_u by this step, then v_u cannot appear in Z_u because the rest of the items are always sampled from $\mathcal{D} \setminus \{v_u\}$ uniformly at random without replacement. Combining this finding with the above, it can be concluded that the value of $\hat{v}$ that maximizes $\Pr[\mathcal{O}_u|\hat{v}]$ can be found by counting the occurrences of each v in the sets $\{Z_u^1, Z_u^2, ..., Z_u^n\}$, and finally choosing as v_u^p the value which has the highest number of occurrences. The pseudocode of this approach is given in Algorithm 5.

Algorithm 5 Attack application to SS protocol

```
1: function AttackSS($\mathcal{O}_u = \{Z_u^1, Z_u^2, ..., Z_u^n\}$)
2:     for $v \in \mathcal{D}$ do
3:         Initialize $count(v) = 0$
4:     for $i \in [1, n]$ do
5:         for $v \in Z_u^i$ do
6:             Increment $count(v) = count(v) + 1$
7:     Find $v^* = \underset{v}{\text{argmax}}\ count(v)$
8:     return $v^*$
```

3.3. Metrics for measuring attack success

Recall that for user u, v_u denotes the user's true value and v_u^p denotes the adversary $\mathcal{A}$'s prediction of the user's value. Over a large user population $\mathcal{P}$, it is in the best interests of $\mathcal{A}$ to achieve $v_u^p = v_u$ or $v_u^p \approx v_u$ for as many $u \in \mathcal{P}$ as possible. We propose two metrics to measure the success of the adversary in achieving this: Adversarial success rate (ASR) and group inference rate (GIR).

Adversarial success rate (ASR) is used to measure what ratio of the adversary's predictions is *exactly* correct, i.e. $v_u^p = v_u$. It is defined as follows:

$$ASR = \frac{\#\text{ of users } u \in \mathcal{P} \text{ such that } v_u^p = v_u}{|\mathcal{P}|} \tag{24}$$

Group inference rate (GIR): In some circumstances, even if the adversary cannot predict the user's true value exactly, it is sufficient for the adversary to predict that the user's value is within a group of sensitive values (or range of values) to cause a privacy breach. For example, consider that users' heart rates are being collected using LDP through a health app. Even if the adversary cannot predict the user's heart rate exactly, inferring that the user's heart rate is among very high values can be sufficient to learn that the user has a heart problem, which constitutes a serious privacy breach.

Our GIR metric is designed to address such situations. Let $\mathcal{G} \subset \mathcal{D}$ be a group of sensitive values among the domain $\mathcal{D}$, e.g., $\mathcal{D}$ contains all heart rate values whereas $\mathcal{G}$ contains only high heart rates. We denote by $\mathcal{P}_\mathcal{G} = \{u \in \mathcal{P} \mid v_u \in \mathcal{G}\}$ and $\mathcal{P}_\mathcal{G}^* = \{u \in \mathcal{P} \mid v_u \in \mathcal{G}, v_u^p \in \mathcal{G}\}$. In other words, $\mathcal{P}_\mathcal{G}$ is the subset of the population whose true values fall within $\mathcal{G}$, and $\mathcal{P}_\mathcal{G}^*$ is the subset of the population whose true values and adversary's predicted values for those users fall within $\mathcal{G}$. Then, GIR is defined as:

$$GIR = \frac{|\mathcal{P}_\mathcal{G}^*|}{|\mathcal{P}_\mathcal{G}|} \tag{25}$$

For both ASR and GIR metrics, higher ASR and GIR imply higher success for the adversary, which means there is weaker privacy protection for users.

3.4. Discussion and extensions

Changing true values of users. Throughout the paper, we denoted the user's true value as v_u. In reality, the user's true value may remain constant or change over the duration of iterative data collection. That is, if we denote by v_u^i the user's true value at timestamp i, it is possible that $v_u^i = v_u^j$ for all $1 \leq i, j \leq n$ (user's true value is constant) or there exist $i, j \in [1, n]$ such that $v_u^i \neq v_u^j$ (user's true value changes over time). Our attack strategies are applicable in both scenarios. However, we should note that if the user's true value is changing frequently and arbitrarily, then the attack boils down to attacking individual observations, e.g., inferring v_u^i from y_u^i in case of GRR, inferring v_u^i from B_u^i in case of RAPPOR, etc. Considering that each LDP protocol satisfies ε-LDP, such inference is not possible beyond the probabilities given by Equation 1. (Otherwise, the protocol would have been violating LDP.)

Overall, our longitudinal attacks are expected to be more successful if the user's true value remains mostly constant over time. As the amount of change increases, we would expect ASR and GIR to decrease. We perform an experimental study regarding this in Section 4.5. When the user's true value is changing over time, the aim is to figure out the most frequently occurring value, i.e. we say that the adversary's prediction is correct if $v_u^p = \text{mode}(v_u^1, v_u^2, ..., v_u^n)$.

Extension to mean inference. In cases where the user's true value is changing $(v_u^1, v_u^2, ..., v_u^n)$, it is possible to extend our attack not just to infer the mode (i.e. single value) but also the mean of the vector: $\text{mean}(v_u^1, v_u^2, ..., v_u^n)$. To achieve this, recall that each of Algorithm 1-5 maintains a list of *scores* (or *counts*) for each $v \in \mathcal{D}$, and then an argmin or argmax is taken over the scores of the values. For mean inference, instead of taking argmin or argmax, we can predict the *mean* as:

$$mean = \frac{\sum\limits_{v \in \mathcal{D}} v \cdot score(v)}{\sum\limits_{x \in \mathcal{D}} score(x)} \tag{26}$$

Differences compared to related works. Our work contributes to a recent line of research in LDP which aims to investigate the privacy properties and/or privacy risks of LDP protocols. The main differences between our work and the related works in this domain are as follows. In [34], Murakami et al. evaluate reidentification attacks in LDP, which aim to link obfuscated data to users. A new measure of reidentification risk (called PIE) is proposed. Although Bayesian principles are used in this work, the goal of their attack (reidentification) is different than ours; in addition, there is no longitudinal aspect which is the main focus of our work. In [35], Gursoy et al. study LDP protocols from the perspective of a Bayesian adversary, and show that the adversary's inference ability can be different under varying protocols, parameters, and adversarial knowledge. Again, there is no longitudinal aspect in [35], i.e. only a single round of data collection is assumed, whereas the focus of our paper is longitudinal data collection. In [36], Gadotti et al. propose a new class of attacks called *pool inference* in which the adversary defines pools of interest for the attack (e.g., groups of true values), and then runs the attack to determine the user's preferred pool. In this attack, the goal is to infer the user's preferred pool rather than exact true value, whereas our attack aims to recover the user's true value (which, we believe, is of higher severity). Furthermore, the attack in [36] is evaluated on the Count Mean Sketch (CMS) mechanism, whereas our attacks are evaluated on multiple LDP protocols, including protocols that are more recent compared to CMS (e.g., OLH, OUE). In [37], Arcolezi et al. study the risks of multidimensional data collection under LDP. In this work, the main focus is on the *multidimensionality* of the user's record, i.e. the record contains multiple attributes. When collecting such records, typically two solutions are used: Sampling (SMP) and RS+FD [38]. Arcolezi et al. [37] show that SMP and RS+FD are vulnerable against reidentification and attribute inference attacks, e.g., due to the disclosure of the sampled attribute. Finally, the Bayes security measure is proposed in [39], which quantifies the expected gain of an adversary who observes the output of a perturbation mechanism. While the work of [39] is not specific to LDP, GRR is analyzed under the Bayes security measure as a fundamental LDP mechanism. However, this work does not consider longitudinal data collection, does not use other popular LDP protocols, and therefore does not show the empirical privacy risks of popular LDP protocols under longitudinal collection as our paper does.

4. Experiments and discussion

4.1. Experiment setup

We conduct experiments to measure the effectiveness of our longitudinal attacks. The goals of our experiments are to: (i) demonstrate that our longitudinal attacks are effective and therefore constitute a cause of practical concern, (ii) analyze the impact of different protocols, ε budgets, n and $|\mathcal{D}|$ values on attack effectiveness, (iii) understand the impact of $\mathcal{G}$ on the GIR metric, (iv) show that our attack is substantially more effective than baseline adversary models, and (v) analyze the impact of consistency in users' true values on attack effectiveness.

In our experiments, we use the six LDP protocols under consideration (GRR, BLH, OLH, RAPPOR, OUE, SS) and three datasets: MSNBC, Kosarak, Uniform. MSNBC and Kosarak are real datasets, Uniform is a synthetic dataset. We implemented the code for protocols and attacks in Python. We execute our longitudinal attacks on all six protocols and three datasets, for varying and commonly used values of the ε parameter: $\varepsilon = 0.5, 1, 2, 4, 6$. For the GIR metric, by default we designated the size of the group $\mathcal{G}$ as 10% of the overall domain $\mathcal{D}$, i.e.: $|\mathcal{G}| = |\mathcal{D}| \times 10\%$. We used Python version 3.9 and popular Python libraries such as numPy, math, and pandas. Hashing operations in BLH and OLH were implemented using the xxhash library. All experiments were conducted on a Windows 10 laptop with Intel i7 CPU and 16 GB RAM.

MSNBC contains logs from msnbc.com website for September 28, 1999. Each row in the dataset corresponds to one user's page visit sequence[1]. Page visits are recorded at the granularity of the webpage category, e.g., frontpage, news, tech, weather, health, sports, business, etc. There are $|\mathcal{D}| = 17$ categories and $|\mathcal{P}| = 989,818$ users total. A large number of users either visit one category of pages or for users that visit multiple categories, their visits are often dominated by one category. Hence, for each user, we determine the most visited category and assume the user's true value is equal to that category.

Kosarak contains click stream data from a Hungarian online news portal[2]. Similar to MSNBC, each row in the dataset corresponds to one user's page visit sequence. However, as opposed to MSNBC, page visits are recorded at the granularity of URL IDs. There are around one million users and 41,270 unique URL IDs. Due to many URLs having extremely small numbers of occurrences (e.g., visited only once or twice in the whole dataset), we pre-processed the dataset by identifying the top-128 most visited URLs across the whole dataset and removing the remaining URLs from each user's page visit sequence. For users who had more than one URL in their resulting stream, the most frequently occurring URL in their stream was picked as their v_u. Users who had zero URLs in their stream were discarded. At the end, we had: $|\mathcal{D}| = 128$ different URLs and $|\mathcal{P}| = 929,669$ users total.

Uniform: We created a synthetic dataset consisting of $|\mathcal{P}| = 100,000$ users and domain size $|\mathcal{D}| = 50$. Each user was assigned a true value sampled from a Uniform distribution across $\mathcal{D}$, i.e. for all $u \in \mathcal{P}$ and for all $j \in \mathcal{D}$, $\Pr[v_u = j] = 1/|\mathcal{D}|$. The assignment of v_u to u occurs once per user.

In Figure 1, we illustrate the distributions of users' true values in each dataset. It can be observed from the figure that the three datasets have varying distributions, e.g., the Uniform dataset resembles a uniform distribution (as expected), Kosarak is highly skewed (some values are much more frequent than others), and MSNBC contains some skew but it is more balanced (between Kosarak and Uniform). This variety in distributions is beneficial for us to perform experiments in diverse conditions.

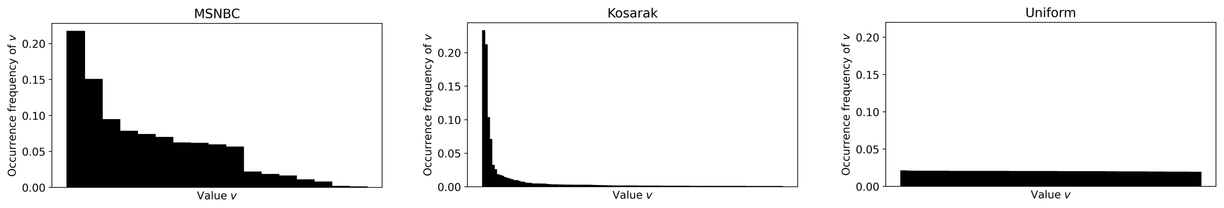

Figure 1. Ranked histograms of users' values in each dataset (MSNBC, Kosarak, Uniform).

[1]UCI Machine Learning Repository: MSNBC.com Anonymous Web Data Set. http://archive.ics.uci.edu/ml/datasets/msnbc.com+anonymous+web+data

[2]Frequent Itemset Mining Dataset Repository. http://fimi.uantwerpen.be/data/

4.2. Impact of number of observations

In Figure 2, we plot how the ASR values change according to the increasing number of observations n. Results with the GIR metric are not included in this section due to the page limit, but we remark that the overall trends and take-away messages are very similar to those in Figure 2. First and foremost, results demonstrate the clear increase in ASR as n becomes larger, which means that the adversary's ability to infer the user's true value keeps increasing as the adversary obtains more observations. For example, consider the RAPPOR protocol and a commonly used ε value such as $\varepsilon = 2$. Under this setting, ASR is below 0.2 on all three datasets when $n = 1$, which means the adversary's inference capability is limited. However, as $n \geq 5$, the adversary's inference capability becomes higher and higher (ASR ≥ 0.5). Eventually, as n approaches 15, ASR becomes close to 1 which means the adversary's inferences are almost always correct.

The second key observation is the positive correlation between ε and ASR. For example, ASRs are lowest when $\varepsilon = 0.5$, they are relatively higher when $\varepsilon = 1$ or $\varepsilon = 2$, and they are highest when $\varepsilon = 6$. This positive correlation agrees with the expectations of ε-LDP because according to ε-LDP, lower ε causes higher amount of perturbation and stronger privacy, therefore ASRs of the adversary are lower.

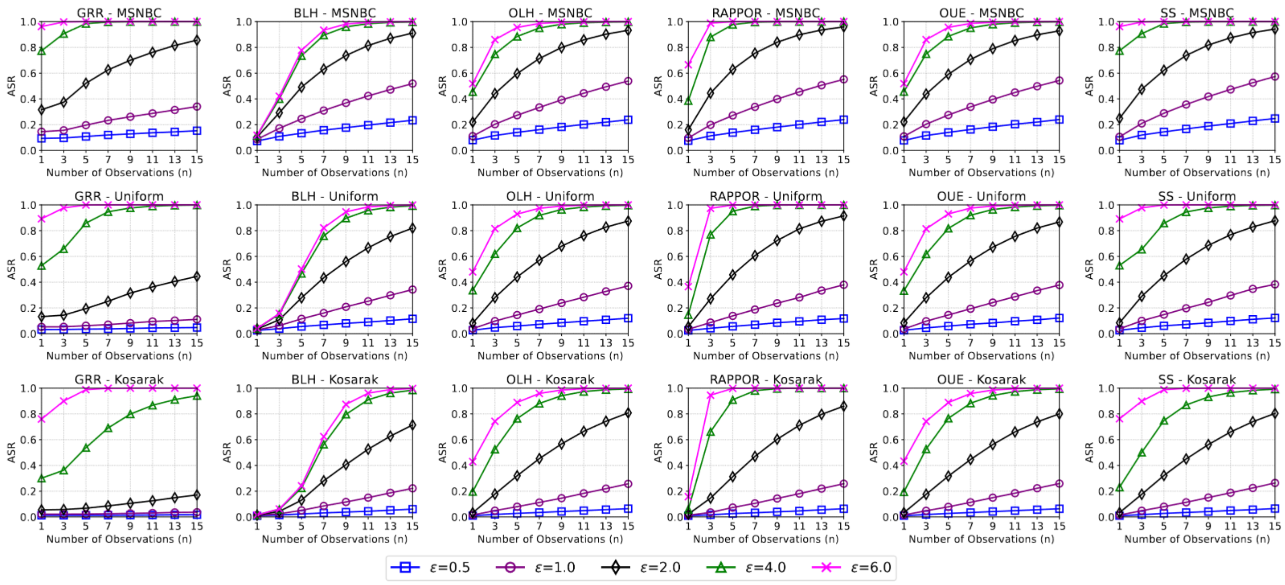

Figure 2. Attack results on six protocols, five different ε values, and three datasets: MSNBC (first row), Uniform (second row), Kosarak (third row). Overall, longitudinal attacks become increasingly effective as n increases.

The third observation is that ASRs can differ from one dataset to another. For example, consider the OLH protocol with $\varepsilon = 4$ and $n = 3$. In this setting, the ASR of OLH is 0.75 on the MSNBC dataset, it is 0.62 on the Uniform dataset, and it is 0.53 on the Kosarak dataset. The reason for this is the varying prediction difficulty on different datasets. Consider a completely random attack strategy: $\mathcal{A}$ predicts v_u^p by sampling an element from $\mathcal{D}$ uniformly at random, without taking into consideration protocol outputs. In expectation, the ASR of this $\mathcal{A}$ would be $1/|\mathcal{D}|$ (see Table 1). Now recall that $|\mathcal{D}|$ is the smallest for MSNBC and highest for Kosarak. Thus, one would expect ASR to be highest on MSNBC and smallest on Kosarak. The experiment results indeed validate this expectation.

Another observation is that ASRs can differ from one protocol to another. Furthermore, the rate of change in ASR may also differ from protocol to protocol, and from ε to ε. For example, as n is increased from 1 to 3, we observe a super-linear increase in ASR for RAPPOR when ε is 4 or 6, but a linear or sublinear increase occurs on many other protocols and ε values. These differences in ASR behavior are not surprising – each protocol has a different way of encoding and perturbing user data, and they are affected differently by changing $\mathcal{D}$ and ε. For example, the bit keeping and flipping probabilities in Ψ_{RAPPOR} are affected by ε but not $\mathcal{D}$. In contrast, Ψ_{GRR} is impacted by both ε and $\mathcal{D}$. The output space of the hash encoding (g) in Θ_{OLH} is also affected by ε. Considering the varying Ψ and Θ functions of protocols, it is natural that changing dataset characteristics and ε values yield different ASR and also different rates of change in ASR.

Overall, despite varying characteristics and behaviors, the main take-away message holds across all protocols, datasets, and ε values: longitudinal attacks become increasingly effective in inferring users' true values when the number of observations (n) becomes larger. Even in cases where users have relatively strong privacy when n is small (such as ASR $\leq$ 0.2 when $n = 1$), longitudinal observations may cause ASR to approach 1.0 as n becomes larger.

4.3. Impact of group size in GIR

Next, we explore the impact of the group size $|\mathcal{G}|$ on the results of the GIR metric. Since the Kosarak dataset has the largest $\mathcal{D}$, it gives us the highest flexibility in terms of trying different $|\mathcal{G}|$ values; hence, this experiment is conducted using the Kosarak dataset. Furthermore, we fix $\varepsilon = 2$ and $n = 9$ to best observe the trend of impact on GIR, e.g., in other settings of ε and n, GIR values can become equal to 1 across different $|\mathcal{G}|$ values and therefore the trends may not be clearly visible.

The results of this experiment are shown in Figure 3. We vary $|\mathcal{G}|$ between $5\% \times |\mathcal{D}|$ and $35\% \times |\mathcal{D}|$, and plot a separate graph for each of the five LDP protocols. For all protocols, the results show that GIR increases as the size of $\mathcal{G}$ increases, but the speed of increase can be different from protocol to protocol. When $\mathcal{G}$ is larger, the sizes of $\mathcal{P}_{\mathcal{G}}$ and $\mathcal{P}^*_{\mathcal{G}}$ are also naturally larger. On the other hand, the increasing GIR shows that the rate of increase in $\mathcal{P}^*_{\mathcal{G}}$ is higher than that in $\mathcal{P}_{\mathcal{G}}$. This indicates that although there may exist cases in which the adversary's predictions are not exactly correct, the predictions may be close to the user's true value, and therefore an increase in $\mathcal{G}$ will benefit the adversary's inference success.

4.4. Impact of domain size and comparison to other approaches

In this section, we explore the impact of $|\mathcal{D}|$ on ASR and GIR. Since Uniform is a synthetic dataset, we are able to modify its $\mathcal{D}$ and generate datasets with different $\mathcal{D}$ while keeping $|\mathcal{P}|$ constant. Therefore, we conduct this experiment using the Uniform dataset, $\varepsilon = 2$, $n = 5$, and $|\mathcal{D}|$ varying between 10 and 90. In addition to the Bayesian adversary $\mathcal{A}$ proposed in Section 3, we consider two additional adversary models for comparison. First one is a random adversary $\mathcal{A}$ whose attack strategy is to randomly sample v_u^p from $\mathcal{D}$. The ASR of this random $\mathcal{A}$ is equal to $\frac{1}{|\mathcal{D}|}$ and the GIR of this random $\mathcal{A}$ is equal to $\frac{|\mathcal{G}|}{|\mathcal{D}|}$. The second one is an adversary bound by randomized response (denoted by RR Bound). Recall from Equation 1 that LDP allows v_u to be reported with probability e^{ε} times higher than some other value $v \in \mathcal{D} \setminus \{v_u\}$. Then, following the GRR principle, the definition of LDP allows an ASR of $\frac{e^{\varepsilon}}{e^{\varepsilon}+|\mathcal{D}|-1}$ and a GIR of $\frac{e^{\varepsilon}+|\mathcal{G}|-1}{e^{\varepsilon}+|\mathcal{D}|-1}$. Thus, for our Bayesian $\mathcal{A}$ to be considered effective, it should have higher ASR and GIR compared to Random $\mathcal{A}$ and RR Bound. In Table 1, we report the ASR and GIR of the random $\mathcal{A}$, the RR Bound, as well as the Bayesian $\mathcal{A}$ under varying $\mathcal{D}$.

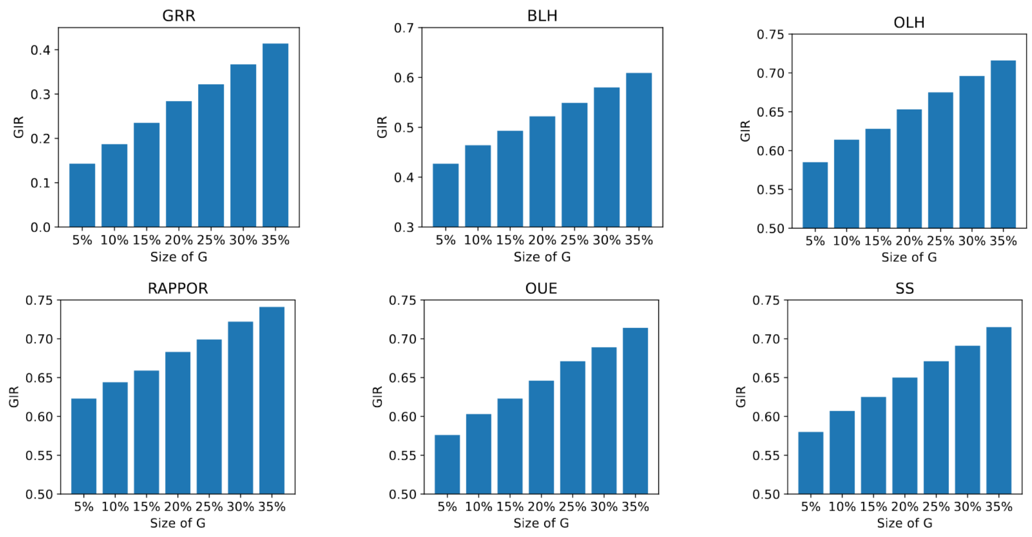

Figure 3. GIR results on six protocols using Kosarak dataset, $\varepsilon = 2$ and $n = 9$. Size of $\mathcal{G}$ is varying between $5\% \times |\mathcal{D}|$ and $35\% \times |\mathcal{D}|$.

We make several observations from Table 1. First, our Bayesian $\mathcal{A}$ always achieves substantially higher ASR and GIR compared to Random $\mathcal{A}$ and RR Bound. This demonstrates the effectiveness and added value of our attacks. As indicated by the ratio $\frac{\text{Bayesian } \mathcal{A}}{\text{Random } \mathcal{A}}$, the differences between the two are most noticeable in case of OLH, RAPPOR and OUE protocols. Second, when we study the ASR metric, we observe that in case of GRR, as $|\mathcal{D}|$ increases we first observe an increase in $\frac{\text{Bayesian } \mathcal{A}}{\text{Random } \mathcal{A}}$ from $|\mathcal{D}| = 10$ to 30, which is followed by a decrease in $\frac{\text{Bayesian } \mathcal{A}}{\text{Random } \mathcal{A}}$ from $|\mathcal{D}| = 30$ to 90. In contrast, when we study the ASR of all remaining protocols, $\frac{\text{Bayesian } \mathcal{A}}{\text{Random } \mathcal{A}}$ increases as $|\mathcal{D}|$ is increased. This is likely because of GRR's reduced accuracy caused by increasing $|\mathcal{D}|$. As shown in Equation 2, GRR's perturbation function is affected by $|\mathcal{D}|$; however, the perturbation functions of the remaining protocols are not affected. Since GRR's perturbation becomes less accurate as $|\mathcal{D}|$ is increased, Bayesian $\mathcal{A}$'s ASR also drops more quickly in GRR compared to other protocols. Third, in terms of GIR, $\frac{\text{Bayesian } \mathcal{A}}{\text{Random } \mathcal{A}}$ decreases as $|\mathcal{D}|$ increases for all protocols. Since $|\mathcal{G}| = |\mathcal{D}| \times 10\%$ in all experiments, the GIR of random $\mathcal{A}$ remains constantly equal to 0.1. On the other hand, Bayesian $\mathcal{A}$'s GIR steadily decreases as $|\mathcal{D}|$ increases, since making a correct prediction (or close-to-correct prediction) becomes more difficult as the prediction space $\mathcal{D}$ is enlarged in general. Thus, we observe a steady decrease in $\frac{\text{Bayesian } \mathcal{A}}{\text{Random } \mathcal{A}}$ in terms of GIR. The speed of decrease is faster for the GRR protocol compared to other protocols (e.g., it is slowest for OLH, RAPPOR and OUE) because of the same reason explained before, i.e. GRR's reduced accuracy as $|\mathcal{D}|$ increases. Fourth, we compare RR Bound and Bayesian $\mathcal{A}$. We observe that Bayesian $\mathcal{A}$ always has higher ASR and GIR compared to RR Bound, but their difference is more pronounced for some protocols and $|\mathcal{D}|$ values. For example, the Bayesian $\mathcal{A}$ is particularly stronger than RR Bound in case of SS protocol, as well as OLH, RAPPOR and OUE protocols with large $|\mathcal{D}|$. It should be noted that the results in Table 1 are with $n = 5$, which is not too high. If higher n values were used (such as $n = 10$), the difference between Bayesian $\mathcal{A}$ and compared approaches (Random $\mathcal{A}$ and RR Bound) would have been even more significant.

Table 1. Comparison of Random adversary, Bayesian adversary, and RR Bound in terms of ASR and GIR under varying $|\mathcal{D}|$. $\varepsilon = 2$ and $n = 5$ parameters are used throughout the table.

		ASR				GIR			
	$\lvert\mathcal{D}\rvert$	Random $\mathcal{A}$	RR Bound	Bayesian $\mathcal{A}$	$\frac{\text{Bayesian } \mathcal{A}}{\text{Random } \mathcal{A}}$	Random $\mathcal{A}$	RR Bound	Bayesian $\mathcal{A}$	$\frac{\text{Bayesian } \mathcal{A}}{\text{Random } \mathcal{A}}$
GRR	10	0.100	0.451	0.709	7.09	0.100	0.451	0.713	7.13
	30	0.033	0.203	0.326	9.88	0.100	0.258	0.372	3.72
	50	0.020	0.131	0.192	9.60	0.100	0.202	0.255	2.55
	70	0.014	0.097	0.134	9.57	0.100	0.175	0.205	2.05
	90	0.011	0.077	0.102	9.27	0.100	0.159	0.185	1.85
BLH	10	0.100	0.451	0.595	5.95	0.100	0.451	0.601	6.01
	30	0.033	0.203	0.377	11.42	0.100	0.258	0.417	4.17
	50	0.020	0.131	0.281	14.05	0.100	0.202	0.338	3.38
	70	0.014	0.097	0.220	15.71	0.100	0.175	0.297	2.97
	90	0.011	0.077	0.178	16.18	0.100	0.159	0.249	2.49
OLH	10	0.100	0.451	0.676	6.76	0.100	0.451	0.679	6.79
	30	0.033	0.203	0.511	15.48	0.100	0.258	0.533	5.33
	50	0.020	0.131	0.440	22.00	0.100	0.202	0.483	4.83
	70	0.014	0.097	0.398	28.42	0.100	0.175	0.456	4.56
	90	0.011	0.077	0.361	32.82	0.100	0.159	0.418	4.18
RAPPOR	10	0.100	0.451	0.715	7.15	0.100	0.451	0.721	7.21
	30	0.033	0.203	0.534	16.18	0.100	0.258	0.562	5.62
	50	0.020	0.131	0.452	22.60	0.100	0.202	0.499	4.99
	70	0.014	0.097	0.397	28.36	0.100	0.175	0.450	4.50
	90	0.011	0.077	0.362	32.91	0.100	0.159	0.411	4.11
OUE	10	0.100	0.451	0.672	6.72	0.100	0.451	0.679	6.79
	30	0.033	0.203	0.507	15.36	0.100	0.258	0.540	5.40
	50	0.020	0.131	0.435	21.75	0.100	0.202	0.479	4.79
	70	0.014	0.097	0.393	28.07	0.100	0.175	0.445	4.45
	90	0.011	0.077	0.362	32.91	0.100	0.159	0.423	4.23
SS	10	0.100	0.451	0.710	7.10	0.100	0.451	0.709	7.09
	30	0.033	0.203	0.541	5.41	0.100	0.258	0.571	5.71
	50	0.020	0.131	0.451	4.51	0.100	0.202	0.489	4.89
	70	0.014	0.097	0.399	3.99	0.100	0.175	0.447	4.47
	90	0.011	0.077	0.374	3.74	0.100	0.159	0.434	4.34

4.5. Impact of consistency in users' true values

Recall from Section 3.4 that users' true values may remain constant or change over time, and in case of changing values, the adversary aims to find: $v_u^p = \text{mode}(v_u^1, v_u^2, ..., v_u^n)$. In this section, we perform experiments regarding the impact of the rate of change in users' true values. We define *consistency* to denote how consistent the user's true value remains during the time of data collection. More formally:

$$Consistency = \frac{\#\text{ of times } i \in [1,n] \text{ such that } v_u^i = \text{mode}(v_u^1, v_u^2, ..., v_u^n)}{n} \tag{27}$$

The higher the consistency, the more constant the user's true value remains throughout multiple iterations of data collection. We fix $n = 10$ and simulate varying levels of consistency $= \{0.1, 0.2, 0.4, 0.6, 0.8, 1.0\}$. On one extreme, consistency $= 1.0$ implies that the user's true value always remains constant. On the other extreme, consistency $= 0.1$ implies that the user's true value changes every timestamp. We measure the ASR values for different consistency levels, ε values, LDP protocols, and datasets. The results are shown in Figure 4.

Results in Figure 4 show that our attacks are most successful when consistency $= 1.0$, i.e. users' true values remain constant. As consistency decreases (i.e. users' values become more random), ASR decreases. These observations hold for a variety of ε values ($\varepsilon = 0.5$, 1.0, 2.0, 4.0, 6.0), LDP protocols (GRR, OLH, RAPPOR, OUE) and datasets. Intuitively, these results agree with our intuition in Section 3.4 that inferring highly changing (i.e. more random) values is more difficult since this problem becomes similar to inferring an LDP protocol's input from its output – as long as the protocol is correct, there is a theoretical limit (enforced by the definition of LDP) regarding the effectiveness of achieving this. On the other hand, our longitudinal attacks are much more effective when the users' data remains mostly static.

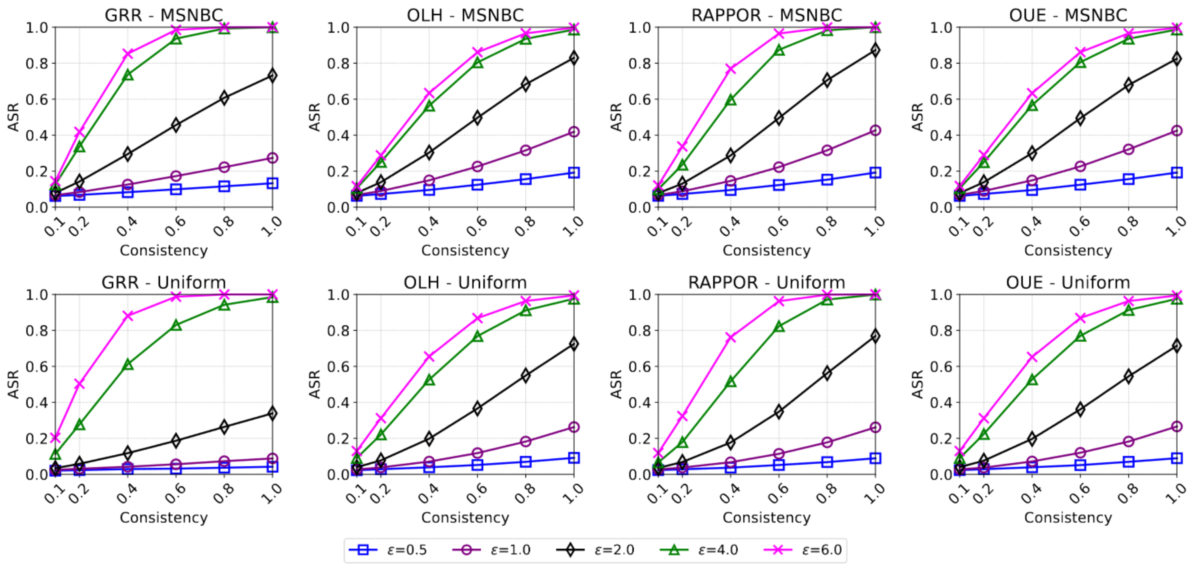

Figure 4. Impact of consistency on attack effectiveness for four protocols (GRR, OLH, RAPPOR, OUE), five different ε values, and two datasets (MSNBC and Uniform).

5. Conclusion

In this paper, we proposed longitudinal attacks against iterative data collection with LDP. We first proposed a general Bayesian attack formulation and then showed how to apply the attack to six commonly used LDP protocols from the literature. Experiments conducted using three datasets and various parameters demonstrate that the longitudinal attacks are effective. The effectiveness of our longitudinal attacks motivates the need for countermeasures, i.e. strategies that can be used together with existing LDP protocols to reduce or restrict the risks of longitudinal attacks.

In future work, we plan to extend our current work in several directions. First, we will explore potential countermeasures such as memoization. In memoization, the user's true value is first perturbed with ε_1 and stored (memoized) on the user's device. Then, at each iteration, the memoized value is reperturbed with a different budget (say ε_2) and sent to the data collector. Hence, the privacy protection at each iteration is achieved by a combination of ε_1 and ε_2; in addition, longitudinal privacy is achieved with ε_1-LDP despite arbitrarily many data collections because the adversary's attack recovers the memoized value. Yet, memoization suffers from the key shortcoming of reduced data utility due to the perturbation applied twice. Second, we will explore the application of w-event privacy. w-event privacy is a useful notion for ensuring the privacy of continuously computed aggregate queries over streaming data. Its idea is to protect a sliding window of at most w timestamps rather than protecting the full stream, which is more difficult and requires more noise [40, 41]. Although w-event privacy was originally proposed for centralized DP, we will investigate its formulation and application to LDP. Finally, we will investigate the application of our attack in contexts where the user's data changes in correlated fashion. For example, in the context of location data, although the user's location may change at each timestamp, the next location is highly correlated with the previous location. In such cases, an attack inspired by the attack proposed in this paper may be leveraged to exploit the correlations between consecutive locations.

Acknowledgment

The authors gratefully acknowledge the support from The Scientific and Technological Research Council of Türkiye (TÜBİTAK) under project number 121E303.

References

[1] Cormode G, Jha S, Kulkarni T, Li N, Srivastava D et al. Privacy at scale: local differential privacy in practice. In: Proceedings of the 2018 International Conference on Management of Data; Houston, TX, USA; 2018. pp. 1655-1658.

[2] Wang T, Blocki J, Li N, Jha S. Locally differentially private protocols for frequency estimation. In: Proceedings of the 26th USENIX Security Symposium; Vancouver, BC, Canada; 2017. pp. 729-745.

[3] Xiong X, Liu S, Li D, Cai Z, Niu X. A comprehensive survey on local differential privacy. Security and Communication Networks 2020; 2020: 1-29. https://doi.org/10.1155/2020/8829523

[4] Cunningham T, Cormode G, Ferhatosmanoglu H, Srivastava D. Real-world trajectory sharing with local differential privacy. Proceedings of the VLDB Endowment 2021; 14 (11): 2283-2295. https://doi.org/10.14778/3476249.3476280

[5] Wang H, Hong H, Xiong L, Qin Z, Hong Y. L-srr: Local differential privacy for location-based services with staircase randomized response. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security; New York, NY, USA; 2022. pp. 2809-2823.

[6] Kim JW, Jang B. Workload-aware indoor positioning data collection via local differential privacy. IEEE Communications Letters 2019; 23 (8): 1352-1356. https://doi.org/10.1109/LCOMM.2019.2922963

[7] Navidan H, Moghtadaiee V, Nazaran N, Alishahi M. Hide me behind the noise: local differential privacy for indoor location privacy. In: 2022 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW); Genoa, Italy; 2022. pp. 514-523.

[8] Marchioro T, Kazlouski A, Markatos EP. Practical crowdsourcing of wearable IoT data with local differential privacy. In: Proceedings of the 8th ACM/IEEE Conference on Internet of Things Design and Implementation; San Antonio, TX, USA; 2023. pp. 275-287.

[9] Wu X, Khosravi MR, Qi L, Ji G, Dou W et al. Locally private frequency estimation of physical symptoms for infectious disease analysis in internet of medical things. Computer Communications 2020; 162: 139-151. https://doi.org/10.1016/j.comcom.2020.08.015

[10] Kim JW, Lim JH, Moon SM, Jang B. Collecting health lifelog data from smartwatch users in a privacy-preserving manner. IEEE Transactions on Consumer Electronics 2019; 65 (3): 369-378. https://doi.org/10.1109/TCE.2019.2924466

[11] Zhao P, Zhang S, Wan S, Liu G, Umer T. A survey of local differential privacy for securing internet of vehicles. The Journal of Supercomputing 2020; 76 (11): 8391-8412. https://doi.org/10.1007/s11227-019-03104-0

[12] Gursoy ME, Tamersoy A, Truex S, Wei W, Liu L. Secure and utility-aware data collection with condensed local differential privacy. IEEE Transactions on Dependable and Secure Computing 2021; 18 (5): 2365-2378. https://doi.org/10.1109/TDSC.2019.2949041

[13] Ou L, Qin Z, Liao S, Li T, Zhang D. Singular spectrum analysis for local differential privacy of classifications in the smart grid. IEEE Internet of Things Journal 2020; 7 (6): 5246-5255. https://doi.org/10.1109/JIOT.2020.2977220

[14] Gai N, Xue K, Zhu B, Yang J, Liu J et al. An efficient data aggregation scheme with local differential privacy in smart grid. Digital Communications and Networks 2022; 8 (3): 333-342. https://doi.org/10.1016/j.dcan.2022.01.004

[15] Erlingsson U, Pihur V, Korolova A. Rappor: randomized aggregatable privacy-preserving ordinal response. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security; Scottsdale, AZ, USA; 2014. pp. 1054-1067.

[16] Differential Privacy Team, Apple. Learning with privacy at scale. 2020. https://docs-assets.developer.apple.com/ml-research/papers/learning-with-privacy-at-scale.pdf

[17] Thakurta AG, Vyrros AH, Vaishampayan US, Kapoor G, Freudinger J et al. Emoji frequency detection and deep link frequency. 2017. US Patent App. 15/640,266.

[18] Ding B, Kulkarni J, Yekhanin S. Collecting telemetry data privately. In: Advances in Neural Information Processing Systems; Long Beach, CA, USA; 2017. pp. 3571-3580.

[19] Apple. Apple differential privacy technical overview. 2017. https://www.apple.com/privacy/docs/Differential-_Privacy_Overview.pdf.

[20] Li T, Sahu AK, Talwalkar A, Smith V. Federated learning: challenges, methods, and future directions. IEEE Signal Processing Magazine 2020; 37 (3): 50-60. https://doi.org/10.1109/MSP.2020.2975749

[21] Truex S, Liu L, Chow KH, Gursoy ME, Wei W. LDP-Fed: Federated learning with local differential privacy. In: Proceedings of the Third ACM International Workshop on Edge Systems, Analytics and Networking; Crete, Greece; 2020. pp. 61-66.

[22] Arcolezi HH, Couchot JF, Al Bouna B, Xiao X. Improving the utility of locally differentially private protocols for longitudinal and multidimensional frequency estimates. Digital Communications and Networks 2022; in press. https://doi.org/10.1016/j.dcan.2022.07.003

[23] Arcolezi HH, Pinzon C, Palamidessi C, Gambs S. Frequency estimation of evolving data under local differential privacy. In: 26th International Conference on Extending Database Technology; Ioannina, Greece; 2023. pp. 512-525.

[24] Bassily R, Smith A. Local, private, efficient protocols for succinct histograms. In: Proceedings of the 47th Annual ACM Symposium on Theory of Computing; Portland, OR, USA; 2015. pp. 127-135.

[25] Fanti G, Pihur V, Erlingsson U. Building a RAPPOR with the unknown: privacy-preserving learning of associations and data dictionaries. Proceedings on Privacy Enhancing Technologies 2016; 2016 (3): 41-61. https://doi.org/10.1515/popets-2016-0015

[26] Gu X, Li M, Cheng Y, Xiong L, Cao Y. Pckv: locally differentially private correlated key-value data collection with optimized utility. In: Proceedings of the 29th USENIX Security Symposium; online; 2020. pp. 967–984.

[27] Wang T, Li N, Jha S. Locally differentially private frequent itemset mining. In: IEEE Symposium on Security and Privacy (S&P); San Francisco, CA, USA; 2018. pp. 127-143.

[28] Jia J, Gong NZ. Calibrate: frequency estimation and heavy hitter identification with local differential privacy via incorporating prior knowledge. In: IEEE International Conference on Computer Communications (INFOCOM); Paris, France; 2019. pp. 2008-2016.

[29] Wang T, Li N, Jha S. Locally differentially private heavy hitter identification. IEEE Transactions on Dependable and Secure Computing 2019; 18 (2): 982-993. https://doi.org/10.1109/TDSC.2019.2927695

[30] Wang T, Lopuhaa-Zwakenberg M, Li Z, Skoric B, Li N. Locally differentially private frequency estimation with consistency. In: Network and Distributed System Security Symposium (NDSS); San Diego, CA, USA; 2020.

[31] Qin Z, Yang Y, Yu T, Khalil I, Xiao X et al. Heavy hitter estimation over set-valued data with local differential privacy. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security; Vienna, Austria; 2016. pp. 192–203.

[32] Qin Z, Yu T, Yang Y, Khalil I, Xiao X et al. Generating synthetic decentralized social graphs with local differential privacy. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security; Dallas, TX, USA; 2017. pp. 425-438.

[33] Wang S, Huang L, Wang P, Nie Y, Hu X et al. Mutual information optimally local private discrete distribution estimation. https://arxiv.org/pdf/1607.08025.pdf

[34] Murakami T, Takahashi K. Toward evaluating re-identification risks in the local privacy model. Transactions on Data Privacy 2021; 14 (3): 79-116.

[35] Gursoy ME, Liu L, Chow KH, Truex S, Wei W. An adversarial approach to protocol analysis and selection in local differential privacy. IEEE Transactions on Information Forensics and Security 2022; 17: 1785-1799. https://doi.org/10.1109/TIFS.2022.3170242

[36] Gadotti A, Houssiau F, Annamalai MSMS, de Montjoye YA. Pool inference attacks on local differential privacy: quantifying the privacy guarantees of Apple's count mean sketch in practice. In: Proceedings of the 31st USENIX Security Symposium; Boston, MA, USA; 2022. pp. 501-518.

[37] Arcolezi HH, Gambs S, Couchot JF, Palamidessi C. On the risks of collecting multidimensional data under local differential privacy. Proceedings of the VLDB Endowment 2023; 16 (5): 1126-1139. https://doi.org/10.14778/3579075.3579086

[38] Arcolezi HH, Couchot JF, Al Bouna B, Xiao X. Random sampling plus fake data: multidimensional frequency estimates with local differential privacy. In: Proceedings of the 30th ACM International Conference on Information and Knowledge Management (CIKM); Queensland, Australia; 2021. pp. 47-57.

[39] Chatzikokolakis K, Cherubin G, Palamidessi C, Troncoso C. Bayes security: a not so average metric. In: IEEE 36th Computer Security Foundations Symposium (CSF); Dubrovnik, Croatia; 2023. pp. 388-406.

[40] Kellaris G, Papadopoulos S, Xiao X, Papadias D. Differentially private event sequences over infinite streams. Proceedings of the VLDB Endowment 2014; 7 (12): 1155-1166. https://doi.org/10.14778/2732977.2732989

[41] Schäler C, Hütter T, Schäler M. Benchmarking the utility of w-event differential privacy mechanisms – when baselines become mighty competitors. Proceedings of the VLDB Endowment 2023; 16 (8): 1830-1842. https://doi.org/10.14778/3594512.3594515