

# In-Memory Perturbation Optimizer Algorithm for Data Privacy on an Anonymous Server

Jaroonsak Chaiprasitjinda<sup>1</sup>, Chetneti Srisaan<sup>1</sup>

<sup>1</sup>*School of Digital Innovation Technology, Rangsit University, Pathumthani, Thailand*

**Abstract** – The emergence of data controllers as a novel term within data privacy laws, such as the General Data Protection Regulation (GDPR), has ushered in significant responsibilities. Stricter regulations prohibit the intentional sharing of personal records on the Internet. This research focuses on safeguarding data privacy, specifically in ubiquitous tabular formats across numerous websites. A novel approach employing a cell-key perturbation method is proposed, demonstrating efficacy in tabular formats. Addressing this challenge, we introduce the in-memory perturbation optimizer (IMPO) algorithm as a novel solution. The primary objective is to create and develop a platform that secures all personal data through a dispenser server, operating in near real-time. Also, it emphasizes the importance of balancing data utility with privacy protection to maintain the integrity and quality of the dataset. Experimental results reveal that the IMPO algorithm outperforms in terms of data accuracy. Additionally, the algorithm introduces an average time delay of 2 seconds, ensuring optimal time service for real-time datasets.

**Keywords** – Data anonymization, privacy-preserving, outliers, privacy, violation.

## 1. Introduction

In 2018, Thailand introduced its inaugural data privacy law, the Personal Data Protection Act (PDPA), which subsequently became enforceable in 2022. The PDPA mandates that all data service industries, including the government sector, uphold the secure preservation of personal data. The terms "data controller" and "data processor" were initially defined by the GDPR laws in 2016 and applicable in various organizations. Due to the inherent nature of data ownership, there is a reluctance to disclose individual personal records.

According to legal provisions, organizations are prohibited from sharing personal records on the Internet, particularly sensitive data such as medical records and racial information. While most internet data is anonymized, there exists a distinction in protecting a user's personal data, allowing for modification through random perturbing methods like the cell-key perturbation method [5].

The primary challenge in data privacy lies in data usage breaches, with numerous instances of intentionally illegal data usage by hackers and marketing firms. Our platform aims to prevent unintentional data breaches through the safeguarding of unpublished datasets and protection against intentional hacking techniques such as reidentification and different methods.

Although attackers may employ various tactics to deceive users directly, such approaches are considered less worthwhile in contemporary times. Recent trends in hacking indicate a shift towards fewer attacks with greater impact, particularly within data centers. Despite advancements in hardware technology capable of processing substantial data transactions, a bottleneck in processing occurs due to central databases designed for regular queries rather than aggregated data queries, such as the group-by-command.

The random perturbing method necessitates a basic group-by function to identify potential risk records. The proposed IMPO platform comprises an IMPO computation server and an IMPO dispenser, as illustrated in Figure 1.

---

DOI: 10.18421/TEM133-16

<https://doi.org/10.18421/TEM133-16>


**Corresponding author:** Jaroonsak Chaiprasitjinda,  
School of Digital Innovation Technology,  
Rangsit University, Pathumthani, Thailand  
Email: [jaroonsak.c65@rsu.ac.th](mailto:jaroonsak.c65@rsu.ac.th)

Received: 05 March 2024.

Revised: 17 June 2024.

Accepted: 01 July 2024.

Published: 27 August 2024.

 © 2024 Jaroonsak Chaiprasitjinda & Chetneti Srisaan; published by UIKTEN. This work is licensed under the Creative Commons Attribution-NonCommercial-NoDeriv 4.0 License.

The article is published with Open Access at <https://www.temjournal.com/>

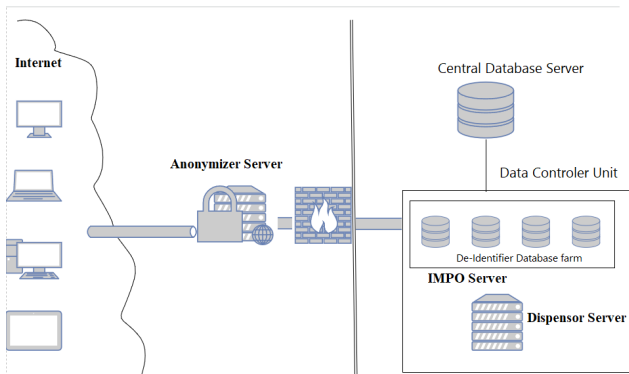


Figure 1. IMPO platform

Figure 1 depicts our newly designed architecture for personalized online anonymized data. Each user receives identical results when querying the same dataset, ensuring compliance with GDPR laws where data controllers are entities responsible for holding other individuals' personal data. In our scenario, the data controller unit comprises two servers and one database, as illustrated in Figure 1.

Netflix previously anonymized a dataset by eliminating all personal identification and replacing it with random identification. Despite this anonymization effort, researchers were able to combine this dataset with an IMDB dataset containing names, linking entries, and successfully de-anonymize individuals [3]. The terms "de-identified data" and "quasi-identifier" were introduced by L. Sweeney [4]. A de-identified data dataset involves removing explicit identifiers while still allowing references back to a single record, as shown in Figure 2. On the other hand, a quasi-identifier is a group of non-direct identifiers that, when combined, can uniquely identify a personal record.

```
df.groupby(['marital-status', 'education', 'sex', 'income']).size().reset_index(name='count')
```

	marital-status	education	sex	income	count
0	Divorced	10th	Female	<=50K	59
1	Divorced	10th	Male	<=50K	45
2	Divorced	10th	Male	>50K	2
3	Divorced	11th	Female	<=50K	69
4	Divorced	11th	Female	>50K	3
...	...	...	...	...	...
310	Widowed	Prof-school	Male	>50K	1
311	Widowed	Some-college	Female	<=50K	126
312	Widowed	Some-college	Female	>50K	7
313	Widowed	Some-college	Male	<=50K	13
314	Widowed	Some-college	Male	>50K	5

315 rows x 5 columns

Figure 2. Deidentified data

Figure 2 illustrates an instance of de-identified data, where only one male graduate from a professional school within a specific age range earns an annual income exceeding \$50,000. The "count" field denotes the frequency of each corresponding row. Figure 2 represents a dataset comprising 315 individuals, characterized by marital status, education, and gender. There exists a potential privacy risk, as hackers may attempt to identify individuals by cross-referencing this data with another dataset, employing a technique commonly known as "reidentification."

Balancing data utility and privacy involves making trade-offs. Achieving perfect data utility is unattainable when datasets undergo extensive modifications. The primary technique for safeguarding privacy is k-anonymity which employs the concepts of generalization and suppression to alter the dataset before release [6]. Numerous research papers indicate that k-anonymity involves the permanent removal of direct identifiers and modification of quasi-identifiers to compromise data utility. Another recent approach is data perturbation, which does not alter permanent attributes. In this research, data perturbation aims to restrict access to datasets or databases to only authorized users. While not an encryption method, data perturbation necessitates a decryption mechanism. The process involves initially modifying and publishing the data, then "decrypting" it back to its original form upon reception.

The primary advantage of the cell-key perturbation method lies in its ability to perturb data at a cell level. This method allows the data controller to introduce noise to specific records while retaining traceability.

Web and database servers have long been the main sources of data delivery to customers. Web pages or websites often serve as common platforms for distributing information in a tabular format. This research opts for the cell-key perturbation method, considering its suitability for tabular formats rather than broader datasets. In the context of tabular format services, another significant challenge is the differing techniques.

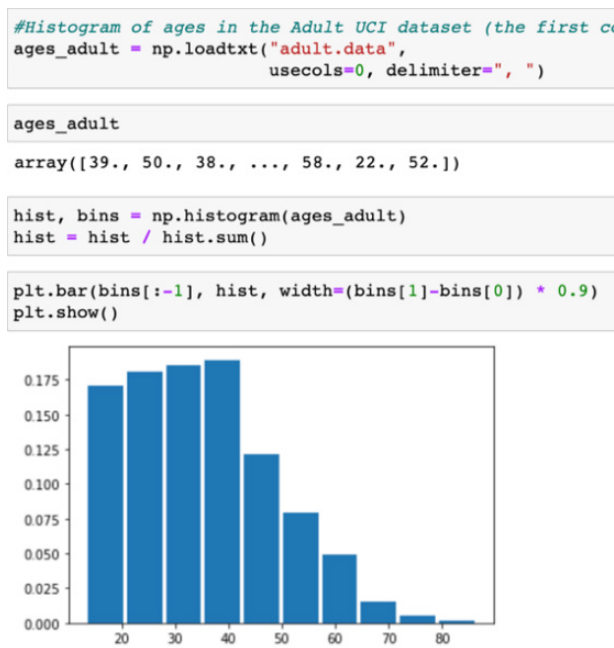


Figure 3. Histogram of age attributes

Figure 3 presents a histogram depicting age attributes with a skewed distribution and high sensitivity. Additionally, a notable outlier is observed within the age attributes, constituting a significant percentage. Winkler [7] asserted in his research that outliers in distributions can contribute to reidentification problems. Addressing this issue is a focal point of Section 4 in this paper.

### 1.1. Perturbation Method

The implementation of a cell-key perturbation method necessitates the computation of tabular structures, particularly in the context of multidimensional tabular data. This method involves creating a singular, perturbed large dataset from which various distinct tables can be derived. Multidimensional tabular data adopts a matrix structure, requiring a high-speed computing processor and a substantial memory capacity. The number of dimensions contributes to the creation of a larger memory space.

For example, consider the renowned adult dataset sourced from the UCI machine learning repository, containing census information from 42 countries. In Figure 4, the dataset displays 73 unique values for age, 42 for native country, and 2 for sex. The matrix formed by these three attributes is of dimensions 73 x 42 x 2. Notably, the dataset encompasses nine categorical attributes, including age, work class, education, marital status, occupation, relationship, race, sex, and native country. The total values resulting from the Cartesian product of these nine attributes amount to discrete values numbering 1,944,069,120.

```
df3=df[['age','native-country','sex']]
p=df['age']
q=df['sex']
r=df['native-country']
pd.crosstab(p, [q, r], rownames=['p'],
           , colnames=['q', 'r'])
```

q Female		r					
		Cambodia	Canada	China	Columbia	Cuba	Dc Re
p							
17	0	0	0	0	0	0	
18	0	0	0	0	0	0	
19	0	0	0	0	0	0	
20	0	1	0	0	0	0	
21	0	0	0	0	1	2	
...	...	...	...	...	...	...	
84	0	0	0	0	0	0	
85	0	0	0	0	0	0	
86	0	0	0	0	0	0	
88	0	0	0	0	0	0	
90	0	0	0	0	0	0	

72 rows x 81 columns

Figure 4. The matrix of quasi-identifier attributes

Figure 4 illustrates an example of a matrix containing three quasi-identifier attributes. In this research paper, there are a total of nine distinct categorical attributes designated as quasi-identifiers. Our algorithm employs a memory-based matrix construction approach, where all conceivable combinations of the matrix are technically possible. However, for this study, we specifically opt for the inclusion of three attributes in each matrix, as detailed in Section 3.1.

### 1.2. In-Memory Architecture

An in-memory database is a database that holds all its data in the memory (RAM) of a server. MongoDB, Redis, Memcached, Cassandra, and H2 are examples of commercial products in the market.

Besides, several freely available databases, such as McObject, SQLite, and CSQL [19], [20], [21], [22], provide in-memory capabilities. For this research, eXtremeDB is selected as the HPC hybrid persistent and in-memory database.

The structure of this paper is as follows: Section 1 is an introduction. Section 2 reviews previous literature. Section 3 outlines the methodology employed in this research. Section 4 presents the experimental demonstrations. Finally, in Section 5, the paper concludes by summarizing the experiment and discussing the obtained results.

## 2. Literature Review

Privacy-preserving terminology was introduced Agrawal and Srikant in 2000 [1], proposing the addition of random transactions to perturb databases. Dwork [2] highlighted differencing techniques as a potential hacking method, emphasizing the risk of revealing medical records when large queries are combined with prior knowledge. Bell, and Koren [3] demonstrated that reidentification techniques could reveal original member records even after removing obvious personal information from Netflix data records.

Sweeney [4] conducted a linking attack involving group health insurance, where she successfully combined information from public voter registration records to identify a significant portion of individuals. In a notable instance, she managed to uncover all the medical records of the governor of Massachusetts, William Weld. She emphasized that a combination of ZIP code, gender, and date of birth is sufficient to identify most people in the U.S.A., prompting global data industries to prioritize carefully protecting their content.

Winkler's [7] research pointed out that outliers in synthetic data distributions may lead to reidentifications, underscoring the importance of addressing outliers.

Kabakus, A. T., & Kara, R. [8] conducted an experiment that tested NoSQL and RDBMS in an in-memory database to see who has better memory management. The in-memory databases used in the experiment were MongoDB, Redis, Memcached, Cassandra, and H2. Experimental results show that NoSQL outperforms RDBMS in all data operations, even though RDBMS runs on an in-memory database.

The UK Office for National Statistics acknowledged the efficacy of cell-key perturbation in protecting against disclosure risk in their open data on previous censuses by Dove *et al.* [9]. Mivule [10] advocated for data perturbation using noise addition as a suitable methodology for safeguarding confidentiality in published datasets.

Sarah Giessing and Reinhard Tent [11] experimented with the cell-key perturbation method on continuous variables using the  $\tau$ -Argus and R package cell-key software. The results show that an extended version of this package can work well on non-integer-valued data.

Bailie and Chien-Hung [12] showcased the advantages of cell-key perturbation over differential privacy, while Karwa *et al.* [13] proposed a new method for safely sharing open datasets using graph data and differential privacy.

Narayanan and Shmatikov [14] demonstrated that even after removing personal identification from the Netflix dataset, researchers could de-anonymize individuals by linking it with an IMDb dataset. Holohan *et al.* [15] presented the IBM Differential Privacy Library, an open-source tool written in Python.

Dankar and Emam [16] reviewed the application of Laplace noise differential privacy in the healthcare industry, emphasizing its suitability for sensitive data with real numbers. Roth [17] explored predicate queries and introduced the concept of distributional privacy, highlighting tradeoffs for statistical queries available both offline and online.

Nukrongsin and Srisa-An [18] demonstrated the effectiveness of cell-key perturbation using a data-swapping concept, where data is swapped per query to return answers without modifying the original data.

## 3. Methodology

This research endeavors to introduce the in-memory concept for online computation in a web server, necessitating sufficient memory capacity.

### 3.1. Proof of In-Memory Space for Matrix Calculation

Instead of employing the group-by function in the database server, matrix computation is adopted in the RAM to analyze a risk group, as discussed in Section 4. The initial step in this research is to provide evidence of in-memory space adequacy for matrix calculations. The study involves nine different categorical attributes serving as quasi-identifiers.

To illustrate a matrix formation, we use a `crosstab()` function in python to construct a matrix starting from  $m=3$  (figure5),  $m=4$  (figure6), and  $m=5$  (figure7).

#### 3.1.1. Case 1: Three Quasi-Identifiers in One Matrix

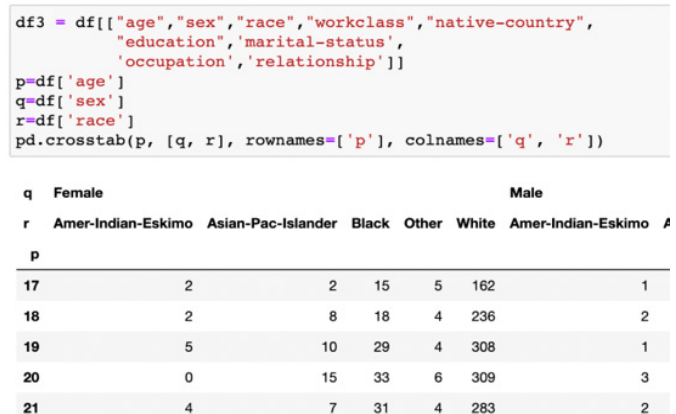


Figure 5. Part of three quasi-identifier attributes matrix ( $m=3$ )



Figure 5 shows a snapshot of the crosstab() function and depicts the construction of a 3-dimensional matrix to illustrate the construction of a 3-dimensional matrix involving three quasi-identifier attributes, resulting in a matrix with a size of 73 rows × 10 columns. The unique record counts for sex, race, and age are 2, 5, and 73, respectively. This specific case is denoted as m=3

In the scenario of m=3, the largest matrix dimensions are 42 (native-country), 73 (age), and 16 (education), containing a total of 40,059 integer numbers. The size of this matrix in memory at a given time per query is 392,448 bits (40,056 × 8).

Considering there are  $n\binom{9}{3}=168$  queries from nine different quasi-identifiers, the total memory size required for one round of computation is 66 Mbits (168 × 392,448 bits) or approximately 9 Mbytes per round, continuously stored in memory. With this size, the computation can effectively be performed online.

### 3.1.2. Case 2: Four Quasi-Identifiers in One Matrix

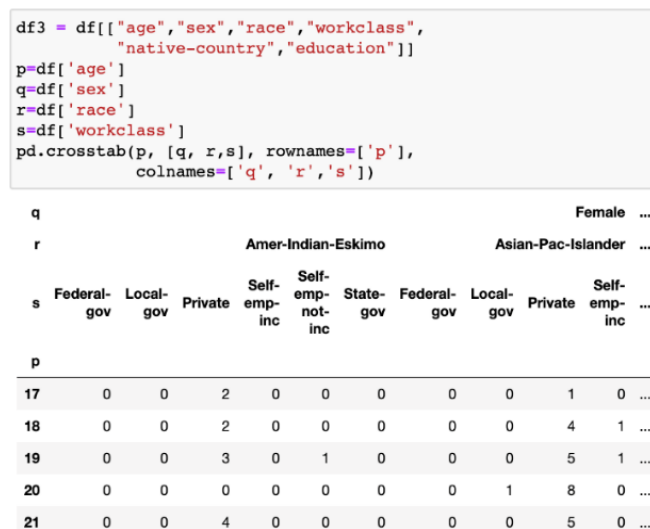


Figure 6. The snapshot of crosstab() function in Python creates a matrix (m=4)

Figure 6 shows a snapshot of the crosstab() function and depicts the construction of a 4-dimensional matrix involving four quasi-identifier attributes, resulting in a matrix with a size of Age (73 rows) × Sex (2) × Race (5) × Work class (9) columns, [Age (73 rows) × {Sex, Race, Work class} (2x5x9=90 columns)] matrix (m=4). The unique record counts for sex, race, work class, and age are 2, 5, 9, and 73, respectively. This specific case is denoted as m=4.

In this m=4 scenario, the largest matrix dimensions are 42 (native-country), 73 (age), occupation (15), and 16 (education), containing a total of 40,059 integer numbers. The size of this matrix in memory at a given time per query is 735,840 bytes.

Considering there are  $n\binom{9}{5}=126$  queries from nine different quasi-identifiers, the total memory size required for one round of computation is 100 Mbytes (126 × 735,840 bytes) per round, continually stored in memory.

### 3.1.3. Case 3: Five Quasi-Identifiers in One Matrix

```
df3 = df[["age", "sex", "race", "workclass",
         "native-country", "education"]]
p=df['age']
q=df['sex']
r=df['race']
s=df['workclass']
t=df['education']
pd.crosstab(p, [q, r,s,t], rownames=['p'],
           colnames=['q', 'r', 's', 't'])
```

p	q Female		r Amer-Indian-Eskimo				s ?		Federal-gov	
	10th	9th	Assoc-voc	Bachelors	HS-grad	Some-college	10th	Assoc-voc		
17	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	1	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0

Figure 7. The snapshot of crosstab() function in Python creates a matrix (m=5)

Figure 7 illustrates the construction of a 5-dimensional matrix using crosstab() function involving five quasi-identifier attributes, resulting in a matrix with a size of 73 rows × 630 columns. The unique record counts for sex, race, work class, education, and age are 2, 5, 9, 16, and 73, respectively. This specific case is denoted as m=5.

In this m=5 scenario, the largest matrix dimensions are 42 (native-country), 73 (age), occupation (15), work class (9), and 16 (education), containing a total of 6,622,560 integer numbers. In this case (m=5), the matrix size is 6,622,560 bytes in memory at a given time per query (worst case).

Considering there are  $n\binom{9}{5}=126$  queries from nine different quasi-identifiers, the total memory size required for one round of computation is 835 Mbytes (126 × 6,622,560 bytes) per round, continually stored in memory. However, with this size, the calculation cannot be completed online in memory.

### 3.1.4. Case 4: Worst case (m=9)

The dataset encompasses nine categorical attributes: 1. age, 2. work class, 3. education, 4. marital status, 5. occupation, 6. relationship, 7. race, 8. sex, and 9. native country.

The total values resulting from the Cartesian product of these nine attributes amount to discrete values numbering 1,944,069,120.

However, the sheer magnitude of values, equivalent to approximately 1,944 Mbytes (1.9 GBytes) in memory, renders online computation impossible. The authors recognize the impracticality of processing such large datasets online, with a noteworthy difference in the number of quasi-identifiers between  $n=3$  and  $n=5$ , representing an approximately 92-fold increase in complexity.

Acknowledging the limitations of memory capacities, the authors have set a pragmatic upper limit of 10 gigabytes for integer values per query in one round of computation. This decision is influenced by the common constraint of laptops having 8 GB of RAM, and for web servers, a maximum of 10 GB of RAM consumption is deemed feasible.

### 3.2. Our Methodology

Our algorithm comprises five essential steps:

Step 1: Identification of all possible outliers in the dataset.

Step 2: Computation of the frequency table matrix utilized to search a risk group.

Step 3: Combination of results obtained from Steps 1 and 2 to form a risk group.

Step 4: Perturbation of the records within the identified risk group using the cell-key perturbation method.

Step 5: Compilation of all anonymized records into a single microdata and subsequent publication through an anonymizer server.

The implementation details are presented in Section 4. For demonstration purposes, we utilize the adult dataset, which is particularly suitable due to its abundance of quasi-identifier attributes and large number of records. The dataset consists of 32,561 records with 15 columns. It should be noted that the dataset was published prior to the announcement of GDPR laws and lacks direct identifiers but does contain quasi-identifiers such as sex, marital status, education, income, etc.

## 4. Experiment

To illustrate the functionality of the IMPO algorithm, a sample Python code is presented herein. It is imperative to note that in actual software development scenarios, development teams extensively employ various tools such as React, PHP, and PostgreSQL databases. However, for the sake of simplicity, these tools are not explicitly mentioned in this paper.

### 4.1. Step 1: Identification of Potential Outliers in a Dataset

Initiating the algorithm involves assigning a unique row ID to each data point. For this demonstration, a row ID in Python is utilized. This unique row ID serves as the key for decryption in the final step of the algorithm.

The dataset under consideration contains outliers, and the following steps outline the process for extracting these outliers. The interquartile rule is employed for outlier identification, as demonstrated in Figure 8, where the range calculation is presented.



Figure 8. Interquartile rule (IQR) method

The age range in the dataset is bounded from zero as the lower limit to 75 as the upper limit, as depicted in Figure 9. Consequently, any records with an age exceeding 75 are extracted and saved into the CSV file illustrated in Figure 10. This extraction results in a total of 3,856 rows within the exported file.

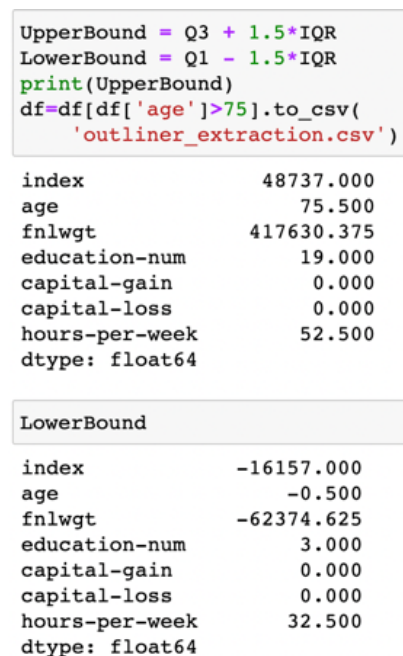


Figure 9. Outliner file

To substantiate the research findings of Winkler and Williams [7], a straightforward group-by function is applied to the extracted dataset, using the attributes {marital-status, 'race', 'sex', 'age', 'native-country'}.

The outcome reveals that all records fall within a high-risk group, warranting the need for perturbation.

```
risk_g=df2.groupby(['marital-status','race','sex','age',
                    'native-country']).size().reset_index(name='count')
risk_g[risk_g['count']==1]
```

	marital-status	race	sex	age	native-country	count
0	Divorced	Black	Female	90	United-States	1
2	Divorced	White	Female	81	?	1
4	Divorced	White	Female	88	United-States	1
5	Divorced	White	Male	77	United-States	1
6	Divorced	White	Male	83	United-States	1
7	Married-civ-spouse	Asian-Pac-Islander	Male	78	United-States	1
8	Married-civ-spouse	Asian-Pac-Islander	Male	90	Philippines	1
9	Married-civ-spouse	Black	Female	77	United-States	1
10	Married-civ-spouse	Black	Male	77	United-States	1
11	Married-civ-spouse	Black	Male	84	United-States	1
12	Married-civ-spouse	Other	Male	77	United-States	1
13	Married-civ-spouse	White	Female	79	United-States	1

Figure 10. Excerpt of a risk group from Outliner

Figure 10 delineates that a high-risk group is derived from the outliers. Among the 3,856 records, 91 belong to this risk group and necessitate protection through data swapping or masking.

#### 4.2. Step 2: Computation of the Frequency Table Matrix for Risk Group Identification

To alleviate the load on the central database server, the search for a risk group within the dataset commences by constructing a sizable matrix in memory. A cross-table matrix formulation on a web server proves more memory-efficient compared to a group-by function on the database server. In an effort to conserve memory, the frequency table matrix is created with three quasi-identifier attributes at a time, as expounded in Section 3.

Despite isolating the outliers into a risk group during step 2, there remain personal records that require protection. An illustrative example of this scenario is presented in Figure 11.

```
df.groupby(['marital-status','race','sex','age',
            'native-country']).size().reset_index(name='count')
```

	marital-status	race	sex	age	native-country	count
0	Divorced	Amer-Indian-Eskimo	Female	21	United-States	1
1	Divorced	Amer-Indian-Eskimo	Female	24	United-States	2
2	Divorced	Amer-Indian-Eskimo	Female	25	United-States	1
3	Divorced	Amer-Indian-Eskimo	Female	29	United-States	1
4	Divorced	Amer-Indian-Eskimo	Female	30	United-States	2
...	...	...	...	...	...	...
3786	Widowed	White	Male	72	United-States	1
3787	Widowed	White	Male	73	United-States	2
3788	Widowed	White	Male	74	?	1
3789	Widowed	White	Male	74	Germany	1
3790	Widowed	White	Male	74	United-States	3

3791 rows x 6 columns

Figure 11. Proof of a risk group

Figure 11 underscores the persistence of issues despite the progress made in the previous step. At this juncture, the avoidance of the group-by-command is motivated by performance concerns. Instead, a crosstab matrix is meticulously constructed to identify records with a frequency of one (f=1). The intricacy lies in creating crosstab matrices ranging from three attributes up to seven (maximum). Each matrix is scrutinized for instances of a singular frequency (f=1). The resultant lists are then amalgamated with those obtained in Step 2, culminating in the identification of the final risk group.

```
df = df[['age','sex','race']]
df = df[df.age.notna() & df.sex.notna()]
p=df['age']
q=df['sex']
r=df['race']
pd.crosstab([q, r], rownames=['p'],
            colnames=['q', 'r'])
```

p	Female					Male				
	Amer-Indian-Eskimo	Asian-Pac-Islander	Black	Other	White	Amer-Indian-Eskimo	Asian-Pac-Islander	Black	Other	White
17	2	2	15	5	162	1	0	19	2	187
18	2	8	18	4	236	2	3	28	2	247
19	5	10	29	4	308	1	5	27	1	322
20	0	15	33	6	309	3	5	30	8	344
21	4	7	31	4	283	2	9	34	2	344

Figure 12. Excerpt of crosstab of three attributes

Figure 12 depicts a singular instance: a female of Amer-Indian-Eskimo ethnicity, aged 17, among the entire dataset of 484,125 records.

#### 4.3. Step 3: Integration of Results from Steps 1 and 2 to Form a Risk Group

By combining the lists obtained in step 2 with those from step 3, we compile a comprehensive set of records requiring protection. The subsequent steps involve determining the method for perturbing the identified records.

#### 4.4. Step 4: Perturbation of Records in the Risk Group using Cell-Key Perturbation Method

In Step 4, records within the risk group are subjected to perturbation through the application of noise, effectively reversing the results of sensitive data (such as income) to opposite values. In contrast to the cell-key perturbation method, our algorithm opts to modify all records within the risk group for the sake of simplicity, thereby accelerating the algorithm's runtime to less than 2 seconds. To facilitate the decryption process back to the original data, all record IDs are transmitted in a separate file.

#### 4.5. Step 5: Compilation of Anonymized Records and Publication via an Anonymous Server

The final step involves aggregating all anonymized records into a microdata set, which is subsequently published through an anonymous server.

## 5. Conclusion

In the real business sector, both data controllers and data processors encounter challenges related to data privacy and security. The majority of data services offered by these entities are often presented in a web page format. This research endeavors to address the safeguarding of data privacy, specifically in tabular formats, which are ubiquitously employed across various websites. Our proposed approach introduces a novel method utilizing cell-key perturbation tailored to handle tabular data effectively.

The in-memory concept is integral to optimizing matrix calculations within random access memory (RAM). While many queries are typically executed on central databases, security considerations necessitate the execution of some queries on a web server. Using a group-by query in a central database can potentially strain or even overload the server, making in-memory matrix calculations on a web server a more prudent approach. The decision to utilize three quasi-identifiers in one matrix, as discussed in Section 3, is rooted in consideration of RAM limitations, a topic explored in Sections 3 and 4.

The in-memory perturbation optimizer (IMPO) algorithm is purposefully designed for online services facilitated by web servers. In contrast to traditional dataset-centric approaches, our web server enables users to request data services through an interactive UX/UI web page, eliminating the need to download the entire dataset. The end-to-end service time is impressively minimized to less than 2 seconds.

### References:

- [1]. Agrawal, R., & Srikant, R. (2000). Privacy-preserving data mining. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, 439-450.
- [2]. Dwork, C. (2011). A firm foundation for private data analysis. *Communications of the ACM*, 54(1), 86-95.
- [3]. Bell, R. M., & Koren, Y. (2007). Lessons from the Netflix prize challenge. *Acm Sigkdd Explorations Newsletter*, 9(2), 75-79.
- [4]. Sweeney, L. (2000). Simple demographics often identify people uniquely. *Health (San Francisco)*, 671(2000), 1-34.
- [5]. Turgay, S., & İlter, İ. (2023). Perturbation methods for protecting data privacy: A review of techniques and applications. *Automation and Machine Learning*, 4(2), 31-41.
- [6]. Kanokngamwitroj, K., Srisa-An, C., & Kasemsawasdi, S. (2022). The effect of data anonymization on a data science project. In *2022 6th International Conference on Information Technology (InCIT)*, 201-206. IEEE.
- [7]. Winkler, W. E. (2004). Re-identification methods for masked microdata. In *International Workshop on Privacy in Statistical*, 216-230. Berlin, Heidelberg: Springer Berlin Heidelberg.
- [8]. Kabakus, A. T., & Kara, R. (2017). A performance evaluation of in-memory databases. *Journal of King Saud University-Computer and Information Sciences*, 29(4), 520-525.
- [9]. Dove, I., Ntoumos, C., & Spicer, K. (2018). Protecting Census 2021 Origin-Destination Data Using a Combination of Cell-Key Perturbation and Suppression. In *Privacy in Statistical Databases: UNESCO Chair in Data Privacy, International Conference, PSD 2018, Valencia, Spain, September 26-28, 2018, Proceedings*, 43-55. Springer International Publishing.
- [10]. Mivule, K. (2012). Utilizing Noise Addition for Data Privacy, an Overview. In *Proceedings of the International Conference on Information and Knowledge Engineering (IKE 2012)*, 65-71
- [11]. Giessing, S., & Tent, R. (2019). Concepts for generalising tools implementing the cell key method to the case of continuous variables. *Joint UNECE/Eurostat Work Session on Statistical Data Confidentiality (The Hague, 29-31 October 2019)*.
- [12]. Bailie, J., & Chien, C. H. (2019). ABS perturbation methodology through the lens of differential privacy. *Joint UNECE/Eurostat work session on statistical data confidentiality*.
- [13]. Karwa, V., Raskhodnikova, S., Smith, A., & Yaroslavtsev, G. (2014). Private analysis of graph structure. *ACM Transactions on Database Systems (TODS)*, 39(3), 1-33.
- [14]. Narayanan, A., & Shmatikov, V. (2008). Robust de-anonymization of large sparse datasets. In *2008 IEEE Symposium on Security and Privacy (sp 2008)*, 111-125. IEEE.
- [15]. Holohan, N., Braghin, S., Aonghusa, P.M., and Levacher, K. (2019). Diffprivlib: The IBM Differential Privacy Library. *arXiv: Cryptography and Security*, 1-5
- [16]. Dankar, F. K., & Emam, K. E. (2013). Practicing differential privacy in health care: A review. *Transactions on Data Privacy*, 6(1), 35-67.
- [17]. Roth, A. (2010). *New algorithms for preserving differential privacy*. Carnegie Mellon University.
- [18]. Nukrongsin, S., & Srisa-An, C. (2023), Cell-key Perturbation Data Privacy Procedure for Security Operations Center Team. *2023 20th International Joint Conference on Computer Science and Software Engineering (JCSSE2023)*, 190-195.
- [19]. SQLite. (n.d.). *Most Widely Deployed and Used Database Engine*. SQLite. Retrieved from: <http://www.sqlite.org/mostdeployed.html> [accessed: 01 March 2024].
- [20]. CSQL. (n.d.). *CSQL Product Family*. CSQL. Retrieved from: <http://csql.sourceforge.net/> [accessed: 02 March 2024].
- [21]. MonetDB. (n.d.). *Analyze and manage your data with standard SQL*. MonetDB. Retrieved from: <http://www.monetdb.org/Home>. [accessed: 02 March 2024].
- [22]. McObject. (n.d.). *The eXtremeDB Database Management Family - McObject LLC*. McObject. Retrieved from: <http://www.mcobject.com/extremedbfamily.shtml>. [accessed: 03 March 2024].