Optimal Random Perturbation at Multiple Privacy Levels

Xiaokui Xiao
Nanyang Technological University
50 Nanyang Avenue, Singapore
xiaokui@ntu.edu.sg

Yufei Tao
Chinese University of Hong Kong
New Territories, Hong Kong
{taoyf@cse, minghua@ie}.cuhk.edu.hk

Minghua Chen
Chinese University of Hong Kong
New Territories, Hong Kong
{taoyf@cse, minghua@ie}.cuhk.edu.hk

ABSTRACT

Random perturbation is a popular method of computing anonymized data for privacy preserving data mining. It is simple to apply, ensures strong privacy protection, and permits effective mining of a large variety of data patterns. However, all the existing studies with good privacy guarantees focus on perturbation *at a single privacy level*. Namely, a *fixed* degree of privacy protection is imposed on *all* anonymized data released by the data holder. This drawback seriously limits the applicability of random perturbation in scenarios where the holder has numerous recipients to which different privacy levels apply.

Motivated by this, we study the problem of *multi-level perturbation*, whose objective is to release multiple versions of a dataset anonymized at different privacy levels. The challenge is that various recipients may collude by sharing their data to infer privacy beyond their permitted levels. Our solution overcomes this obstacle, and achieves two crucial properties. First, collusion is useless, meaning that the colluding recipients cannot learn anything more than what the most trustable recipient (among the colluding recipients) already knows *alone*. Second, the data each recipient receives can be regarded (and hence, analyzed in the same way) as the output of conventional uniform perturbation. Besides its solid theoretical foundation, the proposed technique is both space economical and computationally efficient. It requires $O(n+m)$ expected space, and produces a new anonymized version in $O(n+\log m)$ expected time, where n is the cardinality of the original dataset, and m the number of versions released previously. Both bounds are optimal under the realistic assumption that $n \gg m$.

1. INTRODUCTION

Privacy preserving data mining (PPDM) has received considerable attention from the database community in recent years. Given a dataset D, the objective of PPDM is to discover interesting patterns in D (such as various statistics, decision trees, association rules, and so on) without deriving any private data. *Input perturbation* (IP) [3, 4, 11, 12, 18, 19, 21, 26, 28] is a well-adopted methodology to achieve this goal. Generally speaking, IP converts D to another dataset D^* that permits effective knowledge learning of D, and at the same time, adequately protects the sensitive information[1] in D. Such a D^* can therefore be released to a data miner to perform PPDM.

Random perturbation [3,4,31] is a simple yet effective IP method that provides rigorous privacy guarantees. In this paper, we focus on a specific random perturbation technique [31], which we refer to as *uniform perturbation*. Given a *retention probability* p, uniform perturbation computes D^* from D as follows. First, D^* preserves all the non-sensitive values in D. For each sensitive value x, we toss a coin with head probability p. If the coin heads, x is retained in D^*. Otherwise (the coin tails), we replace x with a random value in its domain. The value of p controls the tradeoff between the effectiveness of data mining, and the strength of privacy protection. When p equals 1, D^* is exactly D, thus maximizing the accuracy of mining, although all private information is revealed. On the other extreme of $p = 0$, all the sensitive values in D^* are completely randomized. In this case, privacy is fully protected, but the precision of mining is the lowest.

1.1 Perturbation at multiple privacy levels

All the previous studies (with some exceptions that are vulnerable in privacy protection; see Section 2) on IP methods focus on perturbation at a single privacy level, namely, the anonymized data released to all recipients provides the same degree of privacy protection. The motivation of this work is that, in practice, it is often necessary to release numerous versions with *different* privacy levels. This is typical when the data holder has recipients that are not equally trustable. For example, it is reasonable to give a less perturbed version to a government agency, and a heavily perturbed version to an unknown organization. Unfortunately, the existing techniques allow us to prepare only one version that either fails to meet the precision requirement of the government agency, or leaks more information than we should to the unknown organization.

Multi-level perturbation is even compulsory if the data holder wants to make profits by charging a cost according to the requested privacy level. This makes sense because various data mining tasks have different precision requirements. For instance, estimating the population of pneumonia patients obviously demands more accurate data than estimating the population of patients having respiratory problems in general. Hence, a recipient is well motivated to pay the minimum cost in exchange of the precision level that just meets her/his needs.

A naive solution to multi-level perturbation is to compute each perturbed version independently. The problem, however, is that

[1] As in the privacy literature, a *sensitive value* refers to a value that its owner may not be willing let others associate with her/him, in order to protect her/his reputation, interests, and so on. The opposite is called a *non-sensitive* value.

VLDB '09, August 24-28, 2009, Lyon, France

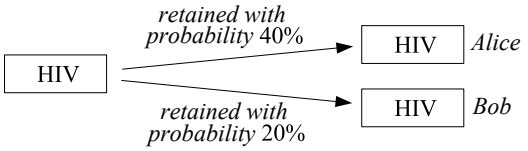


Figure 1: Failure of independent perturbation

this approach fails when two (or more) recipients share their data. To explain, suppose that we employ uniform perturbation to publish data to two recipients Alice and Bob, for whom the retention probability is 40% and 20%, respectively. Let us assume a simple case where D has a single (sensitive) value HIV, and the disease domain has 1,000 distinct values. Figure 1 shows a possible outcome of independent perturbation, where HIV is retained for both Alice and Bob. Note that, without sharing their data, both Alice and Bob have low confidence in believing the real value is HIV. Specifically, to Alice (Bob), the HIV value she (he) receives may have been a random value with 60% (80%) probability. However, once they share their data, they will realize that their received values are unlikely to be random. This is because the chance for both independently perturbed values to be HIV is small (it is less than 1%[2]). Consequently, they will have much higher confidence in believing the original value is HIV. The problem becomes much more severe and complicated when more than two recipients are involved.

1.2 Contributions

We present the first algorithms of multi-level uniform perturbation that are theoretically robust even under the collusion of recipients. Specifically, our algorithms achieve two crucial properties.

1. By sharing data, recipients gain nothing more than having *only* the data of the most trustable recipient whose retention probability is the highest (among all the recipients participating in the sharing). For example, in Figure 1, we ensure that Alice and Bob learn nothing more than Alice *herself* (without sharing). The above property holds regardless of the number of sharing recipients.

2. Each recipient's data can be used in the *same* way as if it was computed via ordinary uniform perturbation. This is important because effective algorithms have been developed to use the output of uniform perturbation to perform analytical tasks such as frequent itemset mining [4, 11, 12], classification by decision trees [8, 29], counting [3], etc. Every recipient, therefore, is able to apply those algorithms *directly*.

In addition to their solid theoretical foundation, the proposed algorithms are both space economical and computationally efficient. Specifically, let n be the cardinality of the underlying dataset D, and m the number of recipients so far. Our algorithms store information that consumes totally $O(n + m)$ expected space, and produces a new perturbed version in $O(n + \log m)$ expected time. For large datasets, n is by far greater than m, in which case the expected space and time complexities are both $O(n)$. This is optimal, noticing that D itself already occupies $\Omega(n)$ space, and writing n records incurs $\Omega(n)$ time. Our technique is *incremental*, because it does not assume that the privacy levels of future releases are known in advance. Instead, a new request can arrive at any time, at any privacy level, no matter how many requests have been handled previously.

The rest of the paper is organized as follows. Section 2 reviews the previous approaches that adopt the IP methodology. Section 3 elaborates the details of uniform perturbation, and its privacy guarantees. Section 4 formally defines the problem of multi-level uniform perturbation. Section 5 gives the details of the proposed algorithm, and analyzes its theoretical properties. Section 6 explains how to implement the algorithm with the minimum space and computational cost. Section 7 evaluates our technique with extensive experiments. Finally, Section 8 concludes the paper with a summary of our results.

2. RELATED WORK

In this section, we discuss the existing input perturbation (IP) approaches, which can be divided into two categories: *partition-based* and *randomization-based*.

Partition-based Approaches. Given a dataset D, a partition-based approach works by first dividing tuples of D into disjoint groups, and then releasing some general information of each group. There have been a plethora of studies on partition-based approaches, such as *generalization* [28], *anatomy* [33, 35], *condensation* [1]. Despite the popularity of these approaches, they are vulnerable to various types of privacy attacks, as pointed out in [16, 26, 32].

With the exceptions of [7, 17, 30, 34], all the existing partition-based approaches consider only *one-time* publication (i.e., only one anonymized version is released); hence, they cannot be applied to multi-level perturbation. The solutions of [7, 17, 30, 34], on the other hand, cannot tackle our problem either. Specifically, [17, 30] focus on anonymizing *marginals* (a marginal is a projection of the dataset on a subset of its attributes), while [7, 34] deal with *dynamic* anonymization, i.e., how to anonymize a dataset in the presence of updates. The meanings of "re-publication" in the above approaches are all different from our work. They still assume a single, universal, privacy level for all the releases, while we allow the publisher to freely choose the amount of anonymization on each release.

Randomization-based Approaches. A randomization-based approach anononymizes a dataset D, by converting D into another dataset D^* through a randomized process. In particular, random perturbation [3, 12, 14, 27, 31] transforms D by replacing some values in D with randomly selected values. The techniques in [2, 5, 6, 10, 24] obtain D^* by adding noise to the numeric values in D. The method in [22] synthesizes D^* based on a model constructed from D.

The random nature of many of these approaches allows them to provide, at least to some extent, protection against collusion. However, to the best of our knowledge, none of them can achieve the same strength of protection (on individual privacy) as described in Section 1.2. The works closest to ours are those on *differential privacy* [9, 10, 13, 24]. They can be employed to publish several anonymized versions which, even put all together by an adversary, still ensure privacy to some extent. Nevertheless, as shown in [24], the combination of any two anonymized versions D_1^* and D_2^* can offer only a privacy guarantee that is strictly worse than that provided by D_1^* or D_2^* alone. This contradicts our goal: the combination of multiple anonymized versions should guarantee the same privacy as the least perturbed version. Li and Chen [20] approach the multi-level problem and guard against collusion with additive Gaussian noise, but their method has limited practical importance because it provides no provable protection on individual privacy; furthermore, it is known that additive Gaussian noise can be com-

[2]It may be deceivingly obvious that this probability ought to be 48%. It is not, because the size of the disease domain plays a crucial role in calculating the probability. The formulae for correct calculation will be given later.

promised by simple denoising methods [15].

In summary, existing IP approaches are insufficient for the multi-level perturbation problem. In Section 5, we will present a new solution to the problem that can be counted as a randomization-based approach. It remains open whether the same degree of privacy can be provided by a partition-based technique.

3. PRELIMINARIES

Random perturbation is type of randomization-based approach that converts a dataset D to its anonymized version D^* by (i) copying the non-sensitive values of D to D^*, and (ii) substituting each sensitive value x of D with a value sampled from a *replacement distribution*. Depending on the choice of replacement distribution, there exist several variations of random perturbation [3, 4, 14, 31]. In this paper, we focus on uniform perturbation [31] because, as shown in [4], it is stronger in privacy protection than other variations.

As explained in Section 1, uniform perturbation processes each sensitive value x of D by tossing a coin with head probability p, where p is a parameter called *retention probability*. If the coin heads, x is retained in D^*; otherwise, it is replaced with a random value in its domain. Algorithm *uni-pert* in Figure 2 summarizes the perturbation process.

More formally, the replacement distribution of uniform perturbation is defined as follows. Let X be a random variable denoting the original value, and Y a random variable denoting the output of perturbation. Both X and Y distribute in a domain DOM with size s. Then, the probability of perturbing a value $x \in DOM$ to $y \in DOM$ is given by:

$$Pr[Y = y|X = x] = \begin{cases} p + (1-p)/s & \text{if } x = y \\ (1-p)/s & \text{if } x \neq y \end{cases} \quad (1)$$

Let us explain the equation by focusing on the case $x = y$, namely, the original value X equals the perturbed value Y (the other case $x \neq y$ is simpler and can be understood similarly). This happens under either of the following disjoint events: (i) X is retained directly, or (ii) X is replaced by a random value from DOM, and this value happens to be x. Event (i) occurs with probability p (the retention probability). The probability of event (ii) equals $(1-p) \cdot (1/s)$, where $(1-p)$ is the probability that we decide not to retain X, and $1/s$ the probability that x is picked randomly from a domain of size s. Hence, the chance that at least one event happens is $p + (1-p)/s$.

Privacy guarantees. Uniform perturbation can ensure several different types of privacy guarantees, such as ρ_1-ρ_2 *privacy* [11] and δ-*growth* [29]. Specifically, both ρ_1-ρ_2 privacy and δ-growth impose constraints on adversaries' prior and posterior beliefs about any sensitive value X in the input data. Let $Q(X)$ be any predicate on X, Y be an anonymized version of X, and $Pr[Q(X)]$ ($Pr\,[Q(X) \mid Y]$) be the adversary's belief in $Q(X)$ before (after) observing Y. ρ_1-ρ_2 privacy requires that

$$Pr[Q(X)] < \rho_1 \Longrightarrow Pr[Q(X) \mid Y] < \rho_2,$$
$$\text{and} \quad Pr[Q(X)] > \rho_2 \Longrightarrow Pr[Q(X) \mid Y] > \rho_1.$$

where ρ_1 and ρ_2 are two constants in $(0, 1]$ such that $\rho_1 < \rho_2$. On the other hand, δ-growth ensures that

$$Pr[Q(X) \mid Y] - Pr[Q(X)] < \delta.$$

In Section 5, we will present a multi-level perturbation algorithm that achieves both ρ_1-ρ_2 privacy and δ-growth.

Algorithm *uni-pert* (x, p)

/* x is the value being perturbed, and p the retention probability */

1. toss a coin with head probability p
2. if the coin heads then return x
3. else return a random value in the domain of x

Figure 2: Algorithm of uniform perturbation

It is noteworthy that, uniform perturbation tends to provide a weaker degree of privacy protection, when there exist correlations among the sensitive values that are known to the adversary [26]. This issue is not specific to uniform perturbation: as pointed out in [25], most existing input perturbation methodologies consider adversaries with no prior knowledge of tuple correlations. In fact, it has been proved in [26] that, if an adversary knows arbitrary correlations among the tuples in D, then no input perturbation algorithm can produce useful anonymized data without enabling the adversary to learn significant information about a predicate $Q(X)$ of some sensitive value X in D. A complete treatment of tuple correlations is beyond the scope of our work.

4. PROBLEM DEFINITION

We assume that the dataset D has a sensitive attribute A with domain DOM, and a set B of non-sensitive attributes. All the tuples of D are considered to have been independently drawn from an identical distribution. Our solutions are applicable, even if there are multiple sensitive attributes A_1, A_2, ..., A_z. In that case, we can simply view all of them as a single attribute with domain $A_1 \times A_2 \times ... \times A_z$, and then, apply the proposed technique in exactly the same way. A similar approach has also been adopted in [26].

We aim at enabling the holder of D to satisfy an *infinite* number of requests for an anonymized version of D. Each request is associated with a retention probability p. This value represents the *trustability level* of the data recipient. That is, the recipient is more (less) trustable if her/his p is higher (lower). The holder must send to the recipient a dataset D^* that can be regarded as the output of uniform perturbation with retention probability p.

Different recipients may collude by sharing their (anonymized) data, in an attempt to infer the original dataset D to a precision beyond their trustability levels. The best way to discourage collusion is to make sure that it will be useless. Notice that, the sharing recipients will obviously glean at least the same knowledge as the most trustable recipient among them. Thus, the best the holder can do is to ensure that nothing more can be derived. Next, we formulate this requirement mathematically.

It suffices to enforce the requirement on individual tuples, because both uniform perturbation and our algorithms anonymize the tuples of D independently. Let t be an arbitrary tuple in D. Consider a set S_{share} of recipients that are colluding to evaluate the chance that the sensitive value X of t satisfies a predicate $Q(X)$. Each recipient has a perturbed copy of X derived using her/his retention probability. Denote by L the set of those perturbed values, among which let $best(L)$ be the one contributed by the most trustable recipient in S_{share}. We will guarantee

$$Pr[Q(X)|L] = Pr[Q(X)|best(L)], \quad (2)$$

where $Pr[Q(X)|L]$ is the confidence that the recipients of S_{share} can derive on conjecture $Q(X)$ by putting their data together, and $Pr[Q(X)|best(L)]$ is their confidence by looking at only $best(L)$ itself. Equation 2, therefore, ensures that collusion yields nothing more about $Q(x)$ than what the most trustable recipient in S_{share} can already tell *alone*. In other words, any ρ_1-ρ_2 privacy or δ-

Symbol	Description
D	The original dataset
A	The sensitive attribute of D
B	The set of non-sensitive attributes of D
DOM	The domain of A
s	The size of DOM
n	The cardinality of D
H	The set of recipients we responded to before
m	The size of H
p_i $(1 \leq i \leq m)$	The i-th highest retention probability of the recipients in H
D_i^* $(1 \leq i \leq m)$	The perturbed version of D returned to the recipient with retention probability p_i
p	The retention probability of the incoming request

Table 1: Frequently used symbols

growth requirement (as discussed in Section 3) we impose on the most trustable recipient still holds under collusion.

Now we are ready to formally define the problem of *multi-level uniform perturbation*. Note that it is trivial to handle the first request, because we can simply return the output of conventional uniform perturbation. The problem kicks in at the second request. In general, let H be the set of all recipients we have already responded to in history, and $|H| \geq 1$. Now, given a new request with retention probability p, we aim at returning a dataset D^* where every tuple t^* corresponds to a tuple $t \in D$ such that

- t^* keeps all the non-sensitive values of t.
- Let X (Y) be the sensitive value t (t^*). The distribution of Y is given in Equation 1, namely, same as uniform perturbation with retention probability p.
- Equation 2 holds for any non-empty subset L of all the perturbed values of t we returned (including the one to the current recipient).

We do not put any constraint on the sequence of recipients, namely, their requests can arrive in any order of their retention probabilities. Furthermore, we deal with the *online* setting where the retention probabilities of future requests are unknown in advance. In analyzing the space and time complexities, we assume that all retention probabilities are at least a small constant $c > 0$. This is reasonable because, in practice, it does not make sense to perform perturbation with an excessively low retention probability. In that case, most sensitive data will be replaced by random values, rendering it impossible to carry out meaningful data mining. In practice, a good choice of c can be 0.001. Table 1 lists the symbols that will be used frequently.

5. MULTI-LEVEL UNIFORM PERTURBATION

Following the notations in Section 4, denote by H the set of recipients we have responded to previously. Let m be the size of H. Recall that each recipient is associated with a retention probability. Denote by p_1, p_2, ..., p_m the retention probabilities associated with the recipients in H, sorted in non-ascending order, namely, $p_1 \geq p_2 \geq ... \geq p_m$. Let D_i^* $(1 \leq i \leq m)$ be the anonymized version of D returned to the recipient with retention probability p_i.

The goal of this section is to design an algorithm for computing a new version D^* for an incoming request with retention probability p. We consider that p is not equivalent to any of $p_1, p_2, ..., p_m$. Otherwise, assume $p = p_i$ for some $i \in [1, m]$; it suffices to return D_i^*. Apparently, in this case, it is not necessary to add the recipient to H. Hence, in the sequel, we consider that all p_1, p_2, ..., p_m are mutually different. Section 5.1 first presents the details of our algorithm, and then, Section 5.2 analyzes its theoretical properties.

5.1 Algorithm

Derivation of D^* must be somehow related to D_1^*, D_2^*, ..., D_m^* (as explained in Section 1.1, independent perturbation may lead to severe privacy breach under collusion). It turns out that, as proved in the next subsection, the correct way to calculate D^* requires at most only two tables from $\{D_1^*, D_2^*, ..., D_m^*\}$. Specifically, let p_l $(1 \leq l \leq m)$ be the smallest probability in $\{p_1, p_2, ..., p_m\}$ that is larger than p, and similarly, p_r $(1 \leq r \leq m)$ be the largest probability in the same set that is smaller than p. Then, the two tables needed are D_l^* and D_r^*.

There are two special cases that must be clarified. First, p may be greater than all the retention probabilities received previously, and hence, p_l does not exist. In this case, we define $p_l = 1$, and D_l^* to be the original dataset D. Second, p may be smaller than all the previous retention probabilities, and hence, p_r does not exist. In this case, we leave p_r and D_r^* undefined.

Our algorithm behaves differently depending on whether p_r exists. Next, we elaborate the details, concentrating on the sensitive values because all the non-sensitive values are directly kept.

- *In case p_r does not exist*, we use only D_l^* to generate D^*. For each tuple $t_l \in D_l^*$, create a sensitive value Y in D^* as follows. Toss a coin with head probability p/p_l. If the coin heads, Y equals the sensitive value of t_l. Otherwise (the coin tails), it is randomly drawn from the domain DOM (of the sensitive attribute).
- *In case p_r exists*, both D_l^* and D_r^* are deployed to derive D^*. Note that every tuple $t_l \in D_l^*$ has a matching tuple $t_r \in D_r^*$ such that t_l and t_r perturb the same tuple in the original dataset D. Denote the sensitive values of t_l and t_r as y_l and y_r, respectively.

  Given t_l and t_r, we create a sensitive value Y in D^* by throwing a coin that heads, tails, or stands with probability u, v, and $1 - u - v$ respectively, where

$$u = \begin{cases} p/p_l & \text{if } y_l = y_r \\ (p - p_r)/(p_l - p_r) & \text{if } y_l \neq y_r \end{cases} \quad (3)$$

$$v = \begin{cases} (1 - \frac{p}{p_l})(1 - \frac{1 - p_r/p}{(s-1)p_r/p_l + 1}) & \text{if } y_l = y_r \\ \frac{p_r(p_l - p)}{p(p_l - p_r)} & \text{if } y_l \neq y_r \end{cases} \quad (4)$$

  where s is the size of DOM. If the coin heads (tails), Y equals y_l (y_r). Otherwise (the coin stands), it is a random value in DOM.

Algorithm *multi-pert* in Figure 3 formally summarizes the above procedures. Next, we will illustrate the intuition behind the algorithm using two examples.

Example 1. Assume that the original dataset D has only a single (sensitive) value x = HIV. Alice issues the first data request with retention probability 40%, which is handled by conventional uniform perturbation. Suppose that HIV is retained, and sent to Alice. So, at this moment, H includes only Alice, and $p_1 = 40\%$.

Now, Bob issues the second request with retention probability $p = 20\%$. As p is smaller than every previous retention probability, p_r does not exist. On the other hand, $p_l = 40\%$, and we will compute the value Y sent to Bob based on *only* the value HIV sent to Alice. Specifically, we do so by tossing a coin with head probability $p/p_l = 50\%$. Y equals HIV if the coin heads, or otherwise a random value from the disease domain DOM.

Algorithm *multi-pert* (p)

/* p is the retention probability of a new request */

1. let $p_1, p_2, ..., p_m$ be the retention probabilities of the previous requests in non-ascending order
2. if p equals p_i for any $i \in [1, m]$, then return D_i^*
3. l = the largest subscript $i \in [1, m]$ such that $p_i > p$
   /* p_l = the lowest of $p_1, p_2, ..., p_m$ greater than p */
4. if p_l does not exist then $p_l = 1$ and $D_l^* = D$
5. r = the smallest subscript $i \in [1, m]$ such that $p_i < p$
   /* p_r = the greatest of $p_1, p_2, ..., p_m$ lower than p */
6. if p_r does not exist
7. for each tuple $t_l \in D_l^*$
8. create a tuple t^* in D^* with $t^*[B] = t_l[B]$
   /* B is the set of non-sensitive attributes */
9. set $t^*[A]$ to $t_l[A]$ with probability p/p_l, or to a random value in DOM with probability $1 - p/p_l$
   /* A is the sensitive attribute */
10. else
11. for each tuple $t_l \in D_l^*$
12. identify its matching tuple $t_r \in D_r^*$
13. create a tuple t^* in D^* with $t^*[B] = t_l[B]$
14. set $t^*[A]$ to $t_l[A]$ with probability u, to $t_r[A]$ with probability v, or to a random value in DOM with probability $1 - u - v$, where u, v are given in Equations 3 and 4, respectively
15. return D^*

Figure 3: Algorithm of multi-level uniform perturbation

To verify the correctness of the above strategy, we need to show that (i) Y follows the same distribution as the output of uniform perturbation, namely, using algorithm *uni-pert* in Figure 2 to perturb HIV with retention probability $p = 20\%$, and (ii) the collusion between Alice and Bob tells no more than what Alice already knows by herself. In fact, fact (ii) is obvious, because Bob's value is computed solely from Alice's value, and therefore, hints no more than what Alice's value already hints. Next, we will explain fact (i).

Let us discuss the probability of Y = HIV, assuming that the disease domain DOM has size $s = 10$. By our algorithm, Y equals HIV under each of the following disjoint events:

I. Alice receives HIV, and the coin we toss for Bob heads.

II. Alice receives HIV, the coin we toss for Bob tails, and the random value we draw from DOM happens to be HIV.

III. Alice does not receive HIV, the coin we toss for Bob tails, and the random value we draw from DOM happens to be HIV.

Since Alice's value is output by *uni-pert* with retention probability 40%, by Equation 1, she gets the value HIV with probability $0.4 + (1 - 0.4)/10 = 0.46$. As a result, Event I occurs with probability $0.46 \times 0.5 = 0.23$ (recall that the coin tossed for Bob has head probability 50%). Event II has probability $0.46 \times 0.5 \times (1/10) = 0.023$, and Event III probability $(1-0.46)\times 0.5 \times (1/10) = 0.027$. Therefore, overall Y has probability $0.23 + 0.023 + 0.027 = 0.28$ to be HIV.

On the other hand, given retention probability $p = 20\%$, algorithm *uni-pert* also has probability $0.2 + (1 - 0.2)/10 = 0.28$ (Equation 1) to perturb value HIV to itself. In a similar fashion, it is not hard to verify that Y has the same probability to take any value as the output of *uni-pert*. This establishes the correctness of fact (i).

Example 2. Let us consider another scenario where everything remains the same with the first recipient Alice, but the second recipient Bob has retention probability $p = 80\%$. In this case, both p_l and p_r exist, and equal 1 and 40%, respectively. Accordingly, we will calculate Y (the value for Bob) using two values: the original value y_l in D, and the value y_r Alice received. Here, y_r happens to coincide with y_l = HIV.

Assuming that DOM has size $s = 10$, we calculate the values of u and v using Equations 3 and 4, obtaining $u = 80\%$ and $v = 17.8\%$. Thus, we throw a coin that heads with probability 80%, tails with probability 17.8%, or stands with probability 2.2%. Y is set to y_l if the coin heads, to y_r if it tails, or drawn randomly from DOM otherwise. As $y_l = y_r$ = HIV, it follows that that Y also equals HIV with 97.8% chance, or a random value with 2.2% probability.

Next, we give the rationale behind the derivation of u and v in Equations 3 and 4 (see Section 5.2 for a rigorous proof). Recall that u (v) is the head (tail) probability of the coin we toss for Bob. For each probability, we need to decide two values for the case $y_l = y_r$ and $y_l \neq y_r$, respectively. Let u_1 (u_2) be the head probability under $y_l = y_r$ ($y_l \neq y_r$), similarly, v_1 and v_2 for tail. Our goal is to find equations from which we can solve u_1, u_2, v_1, and v_2 precisely.

Let random variable X denote the original value in D, and random variable Y_a (Y_b) denote the value Alice (Bob) receives. Remember that Alice and Bob have retention probabilities $p_a = 40\%$ and $p_b = 80\%$, respectively. Our algorithm must ensure two goals. First, collusion is useless, namely,

$$Pr[Q(X)|Y_a = y_a, Y_b = y_b] = Pr[Q(x)|Y_b = y_b] \quad (5)$$

must hold for any values x, y_a, y_b in the domain DOM. Second, Y_b is statistically the same as output perturbation, that is

$$Pr[Y_b = y_b|X = x] = \begin{cases} p_b + (1 - p_b)/s & \text{if } x = y_b \\ (1 - p_b)/s & \text{if } x \neq y_b \end{cases} \quad (6)$$

It turns out that (as shown in the proof of Lemma 1) Equations 5 and 6 both hold when

$$Pr\left[Y_b = y_b|Y_a = y_a, X = x\right] = \begin{cases} \frac{(p_b+\frac{1}{s}(1-p_b))(p_a/p_b+\frac{1}{s}(1-p_a/p_b))}{p_a+\frac{1}{s}(1-p_a)} & \text{if } y_a = y_b = x \\ \frac{(1-p_b)(1-p_a/p_b)}{s^2(p_a+\frac{1}{s}(1-p_a))} & \text{if } y_a = x \neq y_b \\ \frac{1-p_b}{1-p_a}(p_a/p_b + \frac{1}{s}(1-p_a)) & \text{if } y_a = y_b \neq x \\ \frac{1-p_a/p_b}{1-p_a}(p_b + \frac{1}{s}(1-p_b)) & \text{if } x = y_b \neq y_a \\ \frac{(1-p_b)(1-p_a/p_b)}{s(1-p_a)} & \text{otherwise} \end{cases} \quad (7)$$

Four equations can be established based on the first four cases of the above equation set (the last case is redundant as the probabilities of all five cases must sum up to 1). We will explain only the equation arising from the case $y_a = y_b = x$ because the other equations can be formulated similarly.

Notice that $y_a = x$ is exactly the scenario we demonstrated earlier, where the value of Alice is identical to the original value in D (i.e., $y_l = y_r$). In this case, our algorithm gives Bob the same value (i.e., $Y = y_l = y_r$) under either of the following disjoint events:

I. The coin we toss for Bob heads or tails.

II. The coin stands, and the random value we draw from DOM happens to be x.

Event I occurs with probability $u_1 + v_1$, whereas Event II happens with probability $(1 - u_1 - v_1)/s$. Hence, the resulting equation is:

$$u_1+v_1+(1-u_1-v_1)/s = \frac{(p_b + \frac{1}{s}(1 - p_b))(\frac{p_a}{p_b} + \frac{1}{s}(1 - \frac{p_a}{p_b}))}{p_a + \frac{1}{s}(1 - p_a)}.$$

5.2 Theoretical analysis

In this subsection, we will formally prove the correctness of our algorithm by establishing two properties. First, collusion is useless. Second, the anonymized version every recipient obtains can be regarded as the output of conventional uniform perturbation.

As the tuples of the original dataset D are anonymized independently, it suffices to focus on one individual tuple $t \in D$. Let random variable X denote the sensitive value of t. Recall that p_1, p_2, ..., p_m are the retention probabilities of the recipients in H, sorted in descending order. D_i^* $(1 \leq i \leq m)$ is the anonymized version we sent to the recipient with retention probability p_i. Let random variables Y_1, Y_2, ..., Y_m denote the perturbed versions of X in D_1^*, D_2^*, ..., D_m^*, respectively. All of X, Y_1, Y_2, ..., Y_m distribute in domain DOM.

Let L be any non-empty subset of $\{Y_1 = y_1, Y_2 = y_2, ..., Y_m = y_m\}$, where $Y_i = y_i$ represents the observation that the recipient with retention probability p_i obtains a concrete value $y_i \in DOM$. As mentioned in Section 4, proving the futility of collusion is equivalent to verifying the validity of Equation 2 for any L. On the other hand, to convince the recipient with retention probability p_i $(1 \leq i \leq m)$ that the Y_i s/he obtains can be regarded as the outcome of uniform perturbation, we must show that Y_i is statistically the same as the output of *uni-pert* (Figure 2), namely:

$$Pr[Y_i = y_i | X = x] = \begin{cases} p_i + (1 - p_i)/s & \text{if } x = y_i \\ (1 - p_i)/s & \text{if } x \neq y_i \end{cases} \quad (8)$$

for any x, y_1, y_2, ..., y_m in DOM.

For notational convenience, let $p_0 = 1$ and $Y_0 = X$. We first present a key lemma that is useful in the subsequent analysis.

LEMMA 1. *For any $i \in [1, m]$, we have*

$$Pr[Y_i = y_i | Y_0 = y_0, Y_1 = y_1, ..., Y_{i-1} = y_{i-1}] = Pr[Y_i = y_i | Y_{i-1} = y_{i-1}], \quad (9)$$

and

$$Pr[Y_i = y_i | Y_{i-1} = y_{i-1}] = \begin{cases} \frac{p_i}{p_{i-1}} + \left(1 - \frac{p_i}{p_{i-1}}\right)/s & \text{if } y_i = y_{i-1} \\ \left(1 - \frac{p_i}{p_{i-1}}\right)/s & \text{if } y_i \neq y_{i-1} \end{cases} \quad (10)$$

for any y_0, y_1, ..., y_m in DOM.

PROOF. See Appendix 1. □

Intuitively, Equation 9 means that, among X, Y_1, Y_2, ..., Y_{i-1}, it is the last value Y_{i-1} that has the greatest impact on Y_i. In fact, the impact is so heavy that even Y_{i-1} *itself* is already sufficient for analyzing Y_i. Furthermore, Equation 10 tells us another interesting fact: Y_i can actually be regarded as the output of *uni-pert* when the original value is Y_{i-1}, and the retention probability is p_i/p_{i-1}!

Now, we are ready to show that collusion is useless.

THEOREM 1. *Equation 2 holds for any $L \subset \{Y_1 = y_1, Y_2 = y_2, ..., Y_m = y_m\}$.*

PROOF. It suffices to show that the equation is correct when $Q(X)$ takes the form $X = x$ for any $x \in DOM$. Let us introduce I as:

$$I = \{i \mid (Y_i = y_i) \in L, 1 \leq i \leq m\},$$

namely, I collects all the subscripts $i \in [1, m]$ such that Y_i belongs to a colluding recipient. Denote by b (w) as the smallest (largest) value in I. In other words, Y_b (Y_w) is the value contributed by the most (least) trustable recipient among the colluding recipients. Note that the $best(L)$ in Equation 2 is exactly Y_b. Define $\bar{I}$ as the set of integers between b and w that are not captured by I. Namely

$$\bar{I} = \{i \mid b \leq i \leq w, i \notin I\}.$$

Consider the subsequence of variables Y_i, Y_{i+1}, ..., Y_j, where i, j are any integers satisfying $b \leq i \leq j \leq w$. Given concrete values z_i, z_{i+1}, ..., z_j in DOM, let $W_{i,j}(z_i, ..., z_j)$ be the (joint) event where the previous variables are instantiated as

$$Y_i = z_i, Y_{i+1} = z_{i+1}, ..., Y_j = z_j.$$

We are interested in only those events where z_k is fixed to y_k, as long as k belongs to I. In other words, if Y_k is defined for a colluding recipient, we *always* instantiate it to *the* value y_k received by that recipient. Define

$$\bar{L}_{i,j}(z_i, ..., z_j) = W_{i,j}(z_i, ..., z_j) \setminus L, \quad (11)$$

which is the event

$$Y_k = z_k, \forall k \text{ such that } k \in \bar{I} \text{ and } i \leq k \leq j,$$

In the sequel, when z_i, ..., z_j are clear from the context, we simplify $W_{i,j}(z_i, ..., z_j)$ to $W_{i,j}$, and $\bar{L}_{i,j}(z_i, ..., z_j)$ to $\bar{L}_{i,j}$.

Equipped with the above definitions, rewrite $Pr[X = x|L]$ (i.e., the posterior confidence of the colluding recipients) as

$$Pr[X = x|L] = \sum_{z_k \in DOM, \forall k \in \bar{I}} \frac{Pr[X = x, W_{b,w}]}{Pr[W_{b,w}]} Pr[\bar{L}_{b,w}|L]. \quad (12)$$

The rest of the proof will show that, regardless of the values of z_k ($k \in \bar{I}$), it always holds that

$$Pr[X = x, W_{b,w}] = Pr[X = x|Y_b = z_b] \cdot Pr[W_{b,w}]. \quad (13)$$

Thus, Equation 12

$$= \sum_{z_k \in DOM, \forall k \in \bar{I}} Pr[X = x|Y_b = z_b] \cdot Pr[\bar{L}_{b,w}|L]$$
$$= Pr[X = x|Y_b = z_b]$$

which is exactly the right hand side of Equation 2.

It remains to verify Equation 13. First, from Equation 9, it is not hard to derive

$$Pr[W_{b+1,w}|Y_b = z_b, X = x] = Pr[W_{b+1,w}|Y_b = z_b].$$

The above equation can be intuitively understood as follows. Equation 9 indicates that, as long as Y_b is present, we do not need X in analyzing Y_{b+1}. In turn, Y_i $(b + 1 \leq i \leq w - 1)$ allows us to analyze Y_{i+1}. Hence, the chain effect is that, given Y_b, X is not needed in calculating the chance of event $W_{b+1,w}$. Therefore,

$$\begin{aligned} & Pr[X = x, W_{b,w}] \\ = \; & Pr[Y_b = z_b, X = x] \cdot Pr[W_{b+1,w}|Y_b = z_b, X = x] \\ = \; & Pr[Y_b = z_b, X = x] \cdot Pr[W_{b+1,w}|Y_b = z_b] \\ = \; & Pr[X = x|Y_b = z_b] \cdot Pr[W_{b,w}] \end{aligned}$$

validating Equation 13. □

Finally, we prove that every Y_i $(1 \leq i \leq m)$ is statistically the same as the output of algorithm *uni-pert*.

THEOREM 2. *Equation 8 holds for all $1 \leq i \leq m$.*

PROOF. We prove the theorem by induction on i. According to Equation 10 (setting $i = 1$), Equation 8 is correct for $i = 1$. Assuming the correctness of Equation 8 for all $i \leq k \leq m-1$, next we show it must also hold for $i = k+1$.

Equation 9 indicates

$$Pr[Y_{k+1} = y_{k+1}|Y_k = y_k, X = x] = Pr[Y_{k+1} = y_{k+1}|Y_k = y_k].$$

Hence,

$$\begin{aligned} & Pr\left[Y_{k+1} = y_{k+1}|X = x\right] \\ = & \sum_{y_k \in DOM} \Big(Pr[Y_{k+1} = y_{k+1}|Y_k = y_k, X = x] \cdot \\ & \qquad Pr[Y_k = y_k|X = x] \Big) \\ = & \sum_{y_k \in DOM} Pr[Y_{k+1} = y_{k+1}|Y_k = y_k] \cdot Pr[Y_k = y_k|X = x] \end{aligned}$$

By our inductive hypothesis, $Pr[Y_k = y_k|X = x]$ can be solved as in Equation 8. Doing so and solving $Pr[Y_{k+1} = y_{k+1}|Y_k = y_k]$ with Equation 10, the above summation results in:

$$\begin{cases} p_{k+1} + (1 - p_{k+1})/s & \text{if } x = y_{k+1} \\ (1 - p_{k+1})/s & \text{if } x \neq y_{k+1} \end{cases}$$

thus completing the proof. □

Remark. It is worth mentioning that, although in this paper we focus on uniform perturbation, the idea behind *multi-pert* can also be applied to convert some other methods, which currently support only a single level of privacy, to variations that can support multi-level privacy. Examples include perturbation with non-uniform replacement distributions [3] and noise adding methods[3] [2]. Note, however, it does not imply that those variations are able to secure as strong privacy as the techniques proposed in this paper. As mention earlier, for instance, uniformity in general outperforms other replacement distributions in privacy preservation [4], whereas adding Gaussian noise is known to be defective [15].

6. MINIMIZING SPACE AND TIME

In this section, we discuss how to implement the algorithm *multi-pert* in Figure 3 efficiently. As before, let H be the set of recipients we have responded to in history, and $m = |H|$. Let p_1, p_2, ..., p_m be the retention probabilities of the recipients in H, sorted in descending order. Denote by D_i^* $(1 \leq i \leq m)$ the anonymized version of the original dataset D we returned to the recipient with retention probability p_i. As mentioned in Section 4, we consider that all p_1, p_2, ..., p_m are at least a small constant $c > 0$.

Recall that, given a new request with retention probability p, *multi-pert* computes an anonymized version D^* using at most two tables D_l^* and D_r^* from the set $\{D_1^*, D_2^*, ..., D_m^*\}$. Specifically, l (r) is the largest (smallest) subscript $i \in [1, m]$ such that p_i is greater (lower) than p. Once these two tables are identified, *multi-pert* produces D^* by generating at most $2n$ random numbers, where n is the cardinality of D. Assuming that all of $D_1^*, D_2^*, ..., D_m^*$ are materialized, the overall running time of *multi-pert* is $O(n + \log m)$ (where $O(\log m)$ is the time needed to find l and r).

The problem with the above implementation is that storing $D_1^*, D_2^*, ..., D_m^*$ incurs totally $\Omega(nm)$ space, which is expensive. Next, we will reduce the space to $O(n + m)$, without affecting the query complexity $O(n + \log m)$.

Obviously, since we never change any non-sensitive value, the non-sensitive values in the original dataset D need to be stored only once, regardless of the number of recipients. In the sequel, we will concentrate on compressing sensitive values. Furthermore, as tuples are anonymized independently, it suffices to clarify the compression for a single tuple.

A constant-space compression scheme. Consider any tuple $t \in D$. Let random variable Y_i $(1 \leq i \leq m)$ denote its perturbed sensitive value in D_i^*. The crucial observation behind our compression scheme is that, many consecutive values in the list of Y_1, Y_2, ..., Y_m are identical, and therefore, can be represented using less space. Specifically, the m values make $m-1$ consecutive pairs: (Y_1, Y_2), (Y_2, Y_3), ..., (Y_{m-1}, Y_m). Let us say a pair is *disparate* if the two values are different. Denote by $disp(t)$ the number of disparate pairs. Interestingly, the expected value of $disp(t)$ is bounded by a constant independent of m, as shown next.

LEMMA 2. $E[disp(t)] < \ln(1/c)$.

PROOF. For each $i \in [1, m-1]$, let us introduce a random variable V_i which equals 1 if (Y_i, Y_{i+1}) is disparate, or 0 otherwise. Hence,

$$disp(t) = \sum_{i=1}^{m-1} V_i.$$

The expected value $E[V_i]$ of V_i equals the probability $Pr[Y_i \neq Y_{i+1}]$ that Y_i and Y_{i+1} are not identical. In turn, this probability can represented as:

$$Pr[Y_i \neq Y_{i+1}] = \sum_{\forall y \in DOM} Pr[Y_{i+1} \neq y|Y_i = y] \cdot Pr[Y_i = y]$$

where DOM is the domain of Y_i and Y_{i+1}.

Equation 10 implies that $Pr[Y_{i+1} \neq y|Y_i = y] = \frac{s-1}{s}(1 - p_{i+1}/p_i)$, where s is the domain of DOM. Therefore,

$$\begin{aligned} Pr[Y_i \neq Y_{i+1}] &= \sum_{\forall y \in DOM} \frac{s-1}{s}\left(1 - \frac{p_{i+1}}{p_i}\right) Pr[Y_i = y] \\ &= \frac{s-1}{s}\left(1 - \frac{p_{i+1}}{p_i}\right) < 1 - \frac{p_{i+1}}{p_i}. \end{aligned}$$

From the above analysis, we know

$$E[disp(t)] = \sum_{i=1}^{m-1} E[V_i] < \sum_{i=1}^{m-1} 1 - \frac{p_{i+1}}{p_i}.$$

Under the constraint that $1 \geq p_1 \geq p_2 \geq ... \geq p_m \geq c$, $\sum_{i=1}^{m-1}(1 - p_{i+1}/p_i)$ is maximized when $p_1 = 1$ and $p_{i+1}/p_i = c^{1/(m-1)}$ for all $i \in [1, m-1]$, namely:

$$E[disp(t)] < (m-1)(1 - c^{1/(m-1)}).$$

The right hand side of the above inequality increases monotonically with m. Furthermore,

$$\lim_{m \to \infty}(m-1)(1 - c^{1/(m-1)}) = \ln(1/c).$$

Thus, we arrive at $E[disp(t)] < \ln(1/c)$. □

Note that $\ln(1/c)$ is very small. For example, as mentioned in Section 4, in practice a reasonable value for c would be 0.001. In that case, $\ln(1/c) < 7$, namely, on average a tuple has less than 7 disparate pairs regardless of the number m of recipients.

[3]Preliminary effort has been made in [20] to extend Gaussian noise addition to guard against collusion at multiple privacy levels, but the solution thus proposed does not provide provable guarantee on individual privacy.

We are ready to explain how to store Y_1, Y_2, ..., Y_m compactly. In fact, it suffices to build a simple list $history(t)$, where elements have the form $\langle p, Y\rangle$. First, put $\langle p_1, Y_1\rangle$ in $history(t)$. Then, for every disparate consecutive pair (Y_i, Y_{i+1}) $(1 \leq i \leq m-1)$, add an element $\langle p_{i+1}, Y_{i+1}\rangle$ to $history(t)$. By Lemma 2, $history(t)$ occupies $O(1)$ space in expectation.

Given any retention probability p_i $(1 \leq i \leq m)$, $history(t)$ allows us to retrieve the corresponding Y_i in $O(1)$ expected time, even if p_i is not captured by $history(t)$. For this purpose, we find the smallest probability in $history(t)$ that is at least p_i, and return the perturbed value associated with that probability. As an example, assume $m = 5$, and $Y_1 = Y_2$ = HIV, $Y_3 = Y_4$ = pneumonia, Y_5 = HIV. Thus, $history(t)$ consists of $\langle p_1, Y_1\rangle$, $\langle p_3, Y_3\rangle$, and $\langle p_5, Y_5\rangle$. To obtain the perturbed value corresponding to p_2, first identify p_1, which is the lowest probability in $history(t)$ at least p_2 (recall that p_1, p_2, ..., p_5 are in descending order). Hence, the answer is the value Y_1 associated with p_1.

Optimal implementation of *multi-pert*. We need to keep the m retention probabilities of all recipients in H. Furthermore, for each tuple t in the original dataset D, we store its non-sensitive values and the list $history(t)$. The entire storage occupies $O(n + m)$ expected space.

Given an incoming request with retention probability p, *multi-pert* first decides p_l and p_r in $O(\log m)$ time. Then, it constructs D_l^* (D_r^*) in $O(n)$ time by retrieving, for each tuple $t \in D$, its perturbed value at p_l (p_r) from $history(t)$. Finally, the anonymized dataset D^* is derived from D_l^* and D_r^* in $O(n)$ time. The overall computation cost is therefore $O(n + \log m)$.

The $history(t)$ of each tuple $t \in D$ can be updated easily in $O(1)$ time. Specifically, let Y be the perturbed sensitive value of t in D^*. First identify the element $\langle p', Y'\rangle$ where p' is the smallest probability in $history(t)$ larger than p. Add an element $\langle p, Y\rangle$, if p' does not exist (i.e., p is greater than all the probabilities in $history(t)$), or Y is different from Y'. Then, find the element $\langle p'', Y''\rangle$, where p'' is the largest probability in $history(t)$ lower than p. Remove $\langle p'', Y''\rangle$ if Y'' and Y are equivalent. The above description results in the following theorem.

THEOREM 3. *There exists an implementation of algorithm* multi-pert *that consumes $O(n + m)$ expected space, and computes an anonymized version for an incoming request in $O(n + \log m)$ expected time.*

For large datasets, the cardinality n is by far greater than the number m of recipients. In this case, both the expected space and computation complexities are $O(n)$ which is optimal.

7. EXPERIMENTS

This section presents an experimental study of the proposed technique. First, we will verify that our algorithm works fairly well in practice with a pseudo-random generator. Second, we will demonstrate the failure of independent perturbation, thus confirming the motivation in Section 1.1. Finally, we will evaluate the space and computation overhead of our solution. All experiments are performed on a computer with a 4GHz Pentium IV CPU and 2GB memory.

Usefulness of collusion. In Section 5.2, we have rigorously proved that our algorithm *multi-pert* (Figure 3) effectively thwarts collusion. Implicit in the proof is an assumption that we can generate a sequence of numbers that are truly random. As in practice random generation can only be simulated by a pseudo-random generator (such as the *Mersenne twister* algorithm [23] used in our implementation), a natural question is whether *multi-pert* is still effective in that case.

To answer the question, we consider a scenario where the data holder must respond to recipients Alice, Bob, and Charlie (in this order) with retention probabilities p_a = 30%, p_b = 10%, and p_c = 50%, respectively. Note that the ordering ensures that all cases of *multi-pert* (as explained in Section 5.1) will be executed. Let random variable X denote an original sensitive value, in a discrete domain DOM with size s. Let random variables Y_a, Y_b, and Y_c describe the perturbed values sent to Alice, Bob, and Charlie, respectively. If their collusion is useless, equation

$$Pr[X = x|Y_a = y_a, Y_b = y_b, Y_c = y_c] = Pr[X = x|Y_c = y_c] \tag{14}$$

must hold for arbitrary x, y_a, y_b, and y_c in DOM. Remember that the theoretical correctness of Equation 14 has been formally established in Theorem 1. Nevertheless, since our purpose here is practical verification, we must *empirically* calculate both sides, and then check if they are identical. Next, we explain a statistical approach to achieve this.

We start by choosing a probability density function $pdf(x)$ of X, and preparing a 4D array $F[X, Y_a, Y_b, Y_c]$ with all the (totally s^4) cells set to 0. Next, repeat the following *simulation* a huge number of times (10^{10} in our experiments). In each simulation, first generate a sensitive value x following the distribution $pdf(X)$. Then, invoke *multi-pert* to compute the perturbed values y_a, y_b, y_c of x for Alice, Bob, and Charlie, respectively. Finally, increase the cell $F[x, y_a, y_b, y_c]$ by 1. After all simulations, $Pr[X = x|Y_a = y_a, Y_b = y_b, Y_c = y_c]$ can be approximated by

$$\frac{F[x, y_a, y_b, y_c]}{\sum_{\forall x'} F[x', y_a, y_b, y_c]}$$

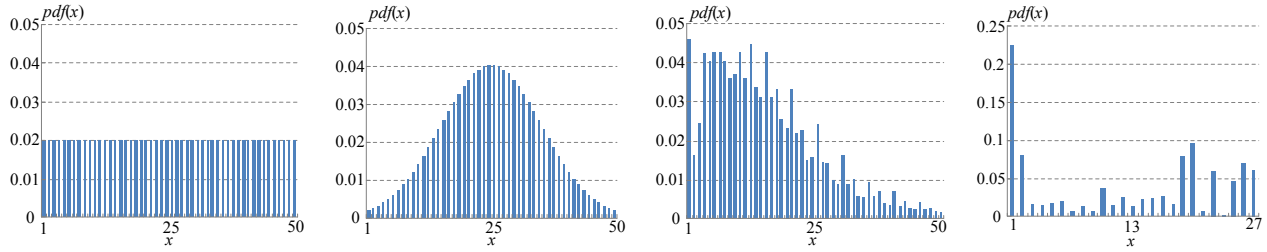
and $Pr[X = x|Y_c = y_c]$ by

$$\frac{\sum_{\forall y'_a, y'_b} F[x, y'_a, y'_b, y_c]}{\sum_{\forall x', y'_a, y'_b} F[x', y'_a, y'_b, y_c]}.$$

We examine 4 different choices of $pdf(x)$. The first two are synthetic distributions: uniform and Gaussian, both in a domain DOM of size $s = 50$, as shown in Figures 4a and 4b respectively. The other two distributions are taken from a real dataset *Adult* that is commonly used in the literature of privacy preservation. This dataset is the product of the Minnesota Population Center (*http://ipums.org*), and contains the information of 600k Americans on 5 attributes: *Age*, *Gender*, *Education*, *Occupation*, and *Salary*. Figure 4c (4d) demonstrates the distribution of *Salary* (*Occupation*), whose domain size is 50 (27). Both distributions are respectively selected as the underlying $pdf(x)$.

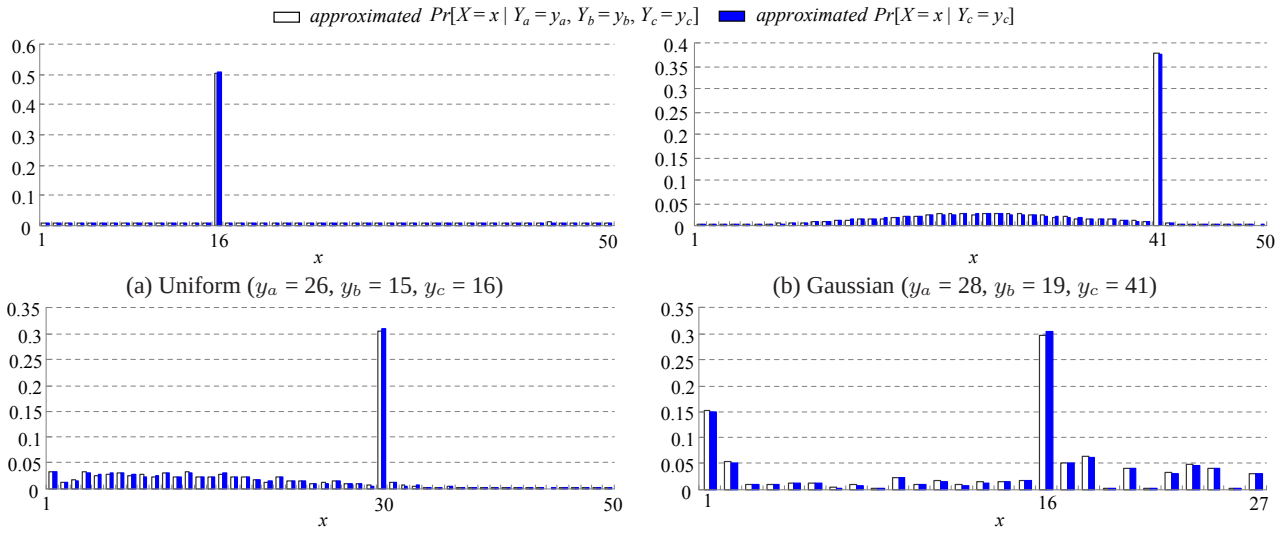
As it is infeasible to visualize a 4D array directly, we instead aim at presenting the cases where the two sides of Equation 14 differ most. First, notice that if we fix y_a, y_b, and y_c but enumerate x through the entire domain DOM, the left hand side of Equation 14 is in fact a distribution $f_{y_a,y_b,y_c}(x)$, which corresponds to the recipients' knowledge about X after collusion. Similarly, the right hand side is also a distribution $f_{y_c}(x)$, which represents the knowledge of Charlie before collusion. Ideally, $f_{y_a,y_b,y_c}(x)$ should be identical to $f_{y_c}(x)$ for all $x \in DOM$, but discrepancy can arise due to pseudo-randomness, precision loss in float calculation, insufficient simulations, etc. We can quantify the dissimilarity of the two distributions using the standard L_2 norm:

$$\sqrt{\sum_{\forall x}(f_{y_a,y_b,y_c}(x) - f_{y_c}(x))^2}. \tag{15}$$



(a) Uniform (b) Gaussian (c) Salary (d) Occupation

Figure 4: Distribution of sensitive values



(a) Uniform (y_a = 26, y_b = 15, y_c = 16) (b) Gaussian (y_a = 28, y_b = 19, y_c = 41)

(c) Salary (y_a = 6, y_b = 46, y_c = 30) (d) Occupation (y_a = 26, y_b = 19, y_c = 16)

Figure 5: Effect of collusion

Among all the combinations of y_a, y_b, y_c, Figure 5a presents the pair of distributions $f_{y_a,y_b,y_c}(x)$ and $f_{y_c}(x)$ that are *most dissimilar*, when $pdf(x)$ is uniform (specifically, the largest dissimilarity is observed at y_a = 26, y_b = 15, y_c = 16). Figures 5b, 5c, 5d give the corresponding results when $pdf(x)$ follows the Gaussian, Salary, and Occupation distributions (see Figure 4), respectively. Clearly, in all cases, both sides of Equation 14 agree with each other very well, indicating that the proposed algorithm *multi-pert* has successfully prevented collusion.

Noteworthily, the fact that there appears a standing-out column in all the distributions in Figure 4, is an intrinsic property of uniform perturbation (which our technique is identical to, as empirically verified shortly) – the perturbed value (in our case, that received by Charlie) always has the highest chance in the reconstructed distribution, even when the value is not equivalent to the original one.

Equivalence to uniform perturbation. Besides anti-collusion, *multi-pert* needs to to make sure that every anonymized version can be regarded as the output of conventional uniform perturbation. In the context of the previous experiments, it implies that equation

$$Pr[Y_i = y_i | X = x] = \begin{cases} p_i + (1-p_i)/s & \text{if } x = y_i \\ (1-p_i)/s & \text{if } x \neq y_i \end{cases} \quad (16)$$

must hold for any $i \in \{a, b, c\}$, and any values of x and y_i in DOM. Theorem 2 has proved the equation in theory. Next, we provide empirical evidence of its correctness in practice.

We follow a strategy similar to Figure 5. Specifically, given $i = a$ (the cases of $i = b$ and c are similar), $Pr[Y_a = y_a | X = x]$ can be approximated as

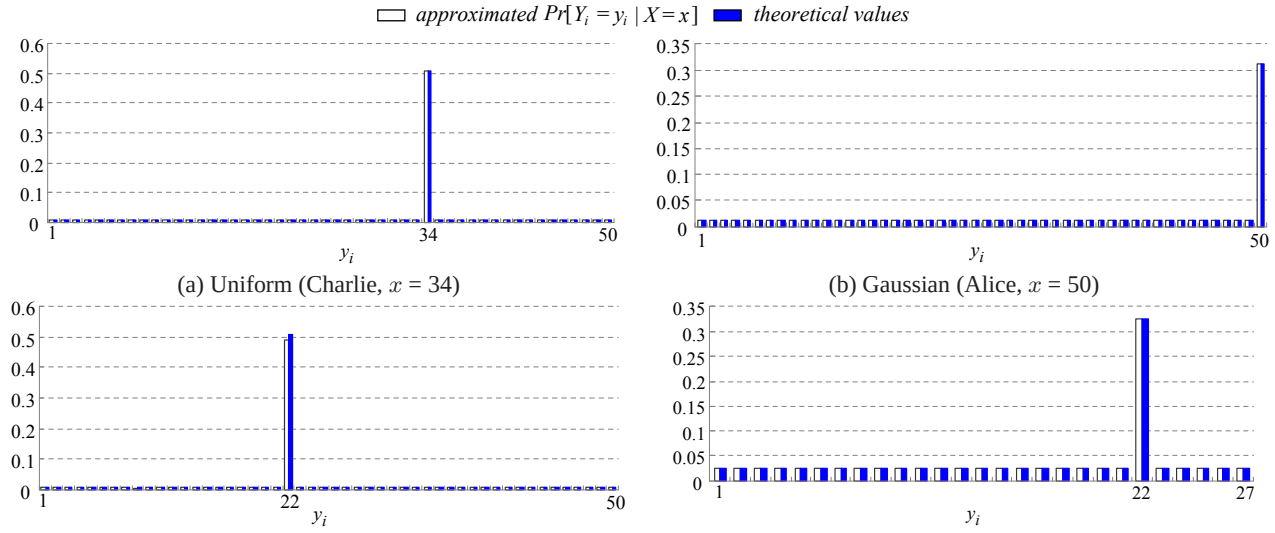
$$\frac{\sum_{\forall y'_b, y'_c} F[x, y_a, y'_b, y'_c]}{\sum_{\forall y'_a, y'_b, y'_c} F[x, y'_a, y'_b, y'_c]}.$$

where F is the array produced by the simulations described earlier. By fixing x and enumerating y_i in DOM, the left hand side of Equation 16 can be regarded as a distribution $f_x(y_i)$ of Y_i. Given the theoretical values on the right hand side of Equation 16, the error of $f_x(y_i)$ can again be quantified by the L_2 norm:

$$\sqrt{\left(f_x(x) - \left(p_i + \frac{1-p_i}{s}\right)\right)^2 + \sum_{\forall y_i, y_i \neq x} \left(f_x(y_i) - \frac{1-p_i}{s}\right)^2}.$$

Among all combinations of $i \in \{a, b, c\}$ and $x \in DOM$, Figure 6a shows the most erroneous distribution $f_x(y_i)$ when $pdf(x)$ is uniform, as well as the corresponding theoretical values (specifically, the highest error is observed in the distribution with $i = c$ and $x = 34$). Figures 6b, 6c, 6d illustrate the results for the other distributions. In all cases, the empirical values perfectly match the theoretical ones, confirming that the output of *multi-pert* is statistically equivalent to that of uniform perturbation.

Failure of independent perturbation. The remaining experiments will use all the attributes of the *Adult* dataset, treating *Salary* as the only sensitive attribute. In Section 1.1, we argue that independent perturbation may lead to severe privacy breach. The next experiment will validate the argument. Specifically, we will show that,



(a) Uniform (Charlie, x = 34)

(b) Gaussian (Alice, x = 50)

(c) Salary (Charlie, x = 22)

(d) Occupation (Alice, x = 22)

Figure 6: Equivalence to uniform perturbation

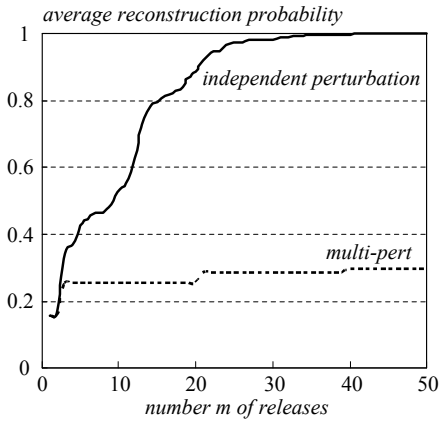
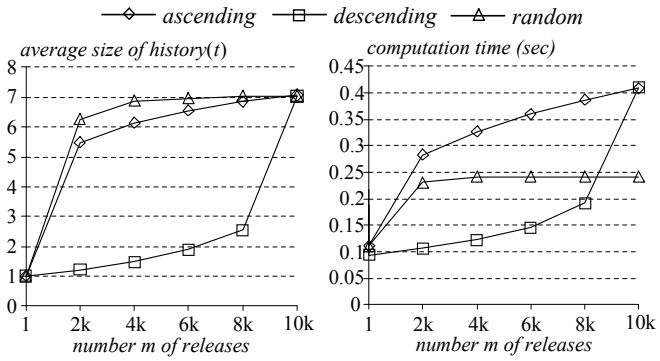


Figure 7: Vulnerability of independent perturbation



(a) Space vs. m

(b) Time vs. m

Figure 8: Overhead of *multi-pert*

when the number m of recipients is large, their collusion may reconstruct the whole dataset with high probability.

We generate all retention probabilities uniformly in the range $[0.001, 0.5]$. For each tuple of *Adult*, define its *reconstruction probability* as the chance that the collusion of all m recipients successfully restores its actual sensitive value (see Appendix 2 for how to derive this probability). We continuously monitor the average reconstruction probability of all tuples as m increases. Figure 7 shows the results of independent perturbation and our algorithm *multi-pert*. Independent perturbation is clearly a vulnerable approach, noticing that its average reconstruction probability quickly surges to nearly 1 after only 30 releases, implying that at this time almost all the tuples are precisely revealed. In contrast, *multi-pert* is much more robust, by retaining an average reconstruction probability of around 30%.

Space and computation cost. We examine three sequences of retention probabilities: *random*, *ascending*, and *descending*. A *random* sequence consists of 10,000 probabilities drawn uniformly from the interval $[0.001, 0.5]$. The *ascending* (*descending*) sequence sorts those probabilities in the *random* sequence in ascending (descending) order.

Multi-pert stores all the non-sensitive values only once. Hence, compared to the original dataset, the extra space overhead is determined by how many (perturbed) sensitive values we must keep for each tuple t, namely, the size of its structure $history(t)$ (see its definition in Section 6). Figure 8a plots the average size of $history(t)$ as a function of m. As predicted by Lemma 2, the size $history(t)$ is bounded by a constant (here, 7) for all arrival orders. In fact, by slightly modifying the proof of Lemma 2, we can tighten the upper bound to $1 + \ln(p_{max}/p_{min})$, where p_{max} (p_{min}) is the highest (lowest) retention probability seen so far. This explains why *random* (*descending*) demands the most (least) space for small m, noticing that it causes the fastest (slowest) growth of p_{max}/p_{min}.

The last experiment inspects the computational efficiency of *multi-pert*. As the whole dataset is small (around 10 mega bytes), we load everything into memory, and measure the CPU time. Figure 8b demonstrates the performance as m increases. Evidently, the algorithm is very efficient. It computes an anonymized dataset in less than half a second in all cases.

8. CONCLUSIONS

Random perturbation is a classic data anonymization technique that has significant importance in practice. It is easy to apply, ensures strong privacy protection, and enables effective data mining. Unfortunately, all the existing studies are restricted to perturbation

at a single privacy level. This is a serious problem, because a data holder often needs to perform anonymization for multiple recipients that are not equally trustable, and hence, should be assigned different privacy levels. In this paper, we have settled the problem with a novel algorithm for randomly perturbing a dataset at an infinite number of privacy levels. Our solution is robust even when recipients attempt to infer extra sensitive information by sharing their data together. Furthermore, the algorithm works in the online setting where the privacy levels of future data requests are unknown in advance, and can arrive at an arbitrary order. Finally, in addition to its rigorous and strong privacy guarantees, the proposed technique is also highly efficient, as its expected space and time complexities are asymptotically optimal.

Acknowledgements

This work was supported by GRF 4173/08, 4161/07, and 1202/06 from HKRGC.

9. REFERENCES

[1] C. C. Aggarwal and P. S. Yu. A condensation approach to privacy preserving data mining. In *EDBT*, pages 183–199, 2004.
[2] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *SIGMOD*, pages 439–450, 2000.
[3] R. Agrawal, R. Srikant, and D. Thomas. Privacy preserving OLAP. In *SIGMOD*, pages 251–262, 2005.
[4] S. Agrawal and J. R. Haritsa. A framework for high-accuracy privacy-preserving mining. In *ICDE*, pages 193–204, 2005.
[5] B. Barak, K. Chaudhuri, C. Dwork, S. Kale, F. McSherry, and K. Talwar. Privacy, accuracy, and consistency too: a holistic solution to contingency table release. In *PODS*, pages 273–282, 2007.
[6] A. Blum, K. Ligett, and A. Roth. A learning theory approach to non-interactive database privacy. In *STOC*, pages 609–618, 2008.
[7] J.-W. Byun, Y. Sohn, E. Bertino, and N. Li. Secure anonymization for incremental datasets. In *SDM*, pages 48–63, 2006.
[8] W. Du and Z. Zhan. Using randomized response techniques for privacy-preserving data mining. In *SIGKDD*, pages 505–510, 2003.
[9] C. Dwork. Differential privacy. In *ICALP*, pages 1–12, 2006.
[10] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, pages 265–284, 2006.
[11] A. V. Evfimievski, J. Gehrke, and R. Srikant. Limiting privacy breaches in privacy preserving data mining. In *PODS*, pages 211–222, 2003.
[12] A. V. Evfimievski, R. Srikant, R. Agrawal, and J. Gehrke. Privacy preserving mining of association rules. In *SIGKDD*, pages 217–228, 2002.
[13] S. R. Ganta, S. P. Kasiviswanathan, and A. Smith. Composition attacks and auxiliary information in data privacy. In *SIGKDD*, pages 265–273, 2008.
[14] Z. Huang and W. Du. OptRR: Optimizing randomized response schemes for privacy-preserving data mining. In *ICDE*, pages 705–714, 2008.
[15] Z. Huang, W. Du, and B. Chen. Deriving private information from randomized data. In *SIGMOD*, pages 37–48, 2005.
[16] D. Kifer. Attacks on privacy and de finetti's theorem. In *SIGMOD*, 2009.
[17] D. Kifer and J. Gehrke. Injecting utility into anonymized datasets. In *SIGMOD*, pages 217–228, 2006.
[18] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan. Incognito: Efficient full-domain k-anonymity. In *SIGMOD*, pages 49–60, 2005.
[19] N. Li, T. Li, and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and ℓ-diversity. In *ICDE*, pages 106–115, 2007.
[20] Y. Li and M. Chen. Enabling multi-level trust in privacy preserving data mining. Technical Report UCB/EECS-2008-156, EECS Department, University of California, Berkeley, Dec 2008.
[21] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. ℓ-diversity: Privacy beyond k-anonymity. In *ICDE*, page 24, 2006.
[22] A. Machanavajjhala, D. Kifer, J. M. Abowd, J. Gehrke, and L. Vilhuber. Privacy: Theory meets practice on the map. In *ICDE*, pages 277–286, 2008.
[23] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *TMCS*, 8(1):3–30, 1998.
[24] F. McSherry and K. Talwar. Mechanism design via differential privacy. In *FOCS*, pages 94–103, 2007.
[25] V. Rastogi, M. Hay, G. Miklau, and D. Suciu. Relationship privacy: Output perturbation for queries with joins. In *PODS*, 2009.
[26] V. Rastogi, S. Hong, and D. Suciu. The boundary between privacy and utility in data publishing. In *VLDB*, pages 531–542, 2007.
[27] S. Rizvi and J. R. Haritsa. Maintaining data privacy in association rule mining. In *VLDB*, pages 682–693, 2002.
[28] P. Samarati. Protecting respondents' identities in microdata release. *TKDE*, 13(6):1010–1027, 2001.
[29] Y. Tao, X. Xiao, J. Li, and D. Zhang. On anti-corruption privacy preserving publication. In *ICDE*, pages 725–734, 2008.
[30] K. Wang and B. C. M. Fung. Anonymizing sequential releases. In *SIGKDD*, pages 414–423, 2006.
[31] S. L. Warner. Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, 60(309):63–69, 1965.
[32] R. C.-W. Wong, A. W.-C. Fu, K. Wang, and J. Pei. Minimality attack in privacy preserving data publishing. In *VLDB*, pages 543–554, 2007.
[33] X. Xiao and Y. Tao. Anatomy: Simple and effective privacy preservation. In *VLDB*, pages 139–150, 2006.
[34] X. Xiao and Y. Tao. m-invariance: Towards privacy preserving re-publication of dynamic datasets. In *SIGMOD*, pages 689–700, 2007.
[35] Q. Zhang, N. Koudas, D. Srivastava, and T. Yu. Aggregate query answering on anonymized tables. In *ICDE*, pages 116–125, 2007.

Appendix 1. Proof of Lemma 1

We prove the lemma by induction on m. For $m = 1$, there is only one possible choice for i (i.e., 1), in which case the lemma is vacuously true. Assuming the lemma's correctness for all $m \leq k$, next we show that it also holds for $m = k + 1$.

Assume, without loss of generality, that the last, $(k+1)$-st, request has retention probability p_j (recall that the past retention probabilities p_1, p_2, ..., p_{m+1} have been sorted in descending order). The proof is simple if $j = k + 1$. Specifically, in this scenario, Y_j is computed only from Y_k. In other words, Y_j is not related to Y_0, Y_1, ..., Y_{k-1}, and hence, Equation 9 is correct. Furthermore, our algorithm *multi-pert* (Figure 3) sets Y_j to Y_k with probability p_k/p_j, or to a random value from DOM with probability $1 - p_k/p_j$. Therefore, Equation 10 is also correct.

For $j < k + 1$, *multi-pert* computes Y_j from Y_{j+1} and Y_{j-1}. Given $i \in [0, j-1]$, Equations 9 and 10 trivially hold from the inductive hypothesis, because they are not affected by Y_j. The subsequent analysis concentrates on $i \geq j$. To simplify notations, let E_z ($1 \leq z \leq m$) denote the event

$$\{Y_0 = y_0, Y_1 = y_1, ..., Y_z = y_z\} \setminus \{Y_j = y_j\}.$$

In other words, if $z < j$, then E_z is the joint event that variables Y_0, Y_1, ..., Y_z are instantiated as y_0, y_1, ..., y_z, respectively. Otherwise ($z \geq j$), the event corresponds to the instantiation of Y_0, ..., Y_{j-1}, Y_{j+1}, ..., Y_z.

Case 1: $i = j$. To verify Equation 9, derive

$$Pr[Y_j = y_j | E_{j-1}] = \sum_{y_{j+1} \in DOM} \Big(Pr[Y_j = y_j | E_{j+1}] \cdot Pr[Y_{j+1} = y_{j+1} | E_{j-1}] \Big) \quad (17)$$

Observe that $Pr[Y_j = y_j|E_{j+1}]$ equals $Pr[Y_j = y_j|Y_{j-1} = y_{j-1}, Y_{j+1} = y_{j+1}]$, because our algorithm picks only Y_{j-1} and Y_{j+1} to calculate Y_j even though all of Y_1, Y_2, ..., Y_{j-2} are also present. Furthermore, by our inductive hypothesis that Equation 10 holds before Y_j exists, we have $Pr[Y_{j+1} = y_{j+1}|E_{j-1}] = Pr[Y_{j+1} = y_{j+1}|Y_{j-1} = y_{j-1}]$. Thus, Equation 17 evolves into

$$\begin{aligned} &= \sum_{y_{j+1}\in DOM} \Big(Pr[Y_j = y_j|Y_{j+1} = y_{j+1}, Y_{j-1} = y_{j-1}] \cdot \\ &\qquad Pr[Y_{j+1} = y_{j+1}|Y_{j-1} = y_{j-1}] \Big) \\ &= Pr[Y_j = y_j|Y_{j-1} = y_{j-1}]. \end{aligned}$$

To verify Equation 10, we first write down

$$\begin{aligned} &Pr[Y_j = y_j|Y_{j-1} = y_{j-1}] \\ &= \sum_{y_{j+1}\in DOM} \Big(Pr[Y_j = y_j|Y_{j+1} = y_{j+1}, Y_{j-1} = y_{j-1}] \cdot \\ &\qquad Pr[Y_{j+1} = y_{j+1}|Y_{j-1} = y_{j-1}] \Big). \end{aligned} \tag{18}$$

As Equation 8 holds before Y_j exists, we know

$$\begin{aligned} &Pr[Y_{j+1} = y_{j+1}|Y_{j-1} = y_{j-1}] \\ &= \begin{cases} \frac{p_{j+1}}{p_{j-1}} + (1 - \frac{p_{j+1}}{p_{j-1}})/s & \text{if } y_{j+1} = y_{j-1} \\ (1 - \frac{p_{j+1}}{p_{j-1}})/s & \text{otherwise} \end{cases} \end{aligned} \tag{19}$$

Meanwhile, according to how *multi-pert* works, we also have

$$\begin{aligned} &Pr[Y_j = y_j|Y_{j+1} = y_{j+1}, Y_{j-1} = y_{j-1}] \\ &= \begin{cases} \frac{(\rho_1 + \frac{1}{s}(1-\rho_1))(\rho_0 + \frac{1}{s}(1-\rho_0))}{\rho_2 + \frac{1}{s}(1-\rho_2)} & \text{if } y_j = y_{j+1} = y_{j-1} \\ \frac{(1-\rho_1)(1-\rho_0)}{s^2(\rho_2 + \frac{1}{s}(1-\rho_2))} & \text{if } y_j \neq y_{j+1} = y_{j-1} \\ \frac{1-\rho_1}{1-\rho_2}(\rho_0 + \frac{1}{s}(1-\rho_0)) & \text{if } y_j = y_{j+1} \neq y_{j-1} \\ \frac{1-\rho_0}{1-\rho_2}(\rho_1 + \frac{1}{s}(1-\rho_1)) & \text{if } y_j = y_{j-1} \neq y_{j+1} \\ \frac{(1-\rho_1)(1-\rho_0)}{s(1-\rho_2)} & \text{otherwise} \end{cases} \end{aligned} \tag{20}$$

where $\rho_1 = p_j/p_{j-1}$, $\rho_2 = p_{j+1}/p_{j-1}$, $\rho_0 = p_{j+1}/p_j$. Plugging Equations 19 and 20 into Equation 18, it is easy to verify that Equation 10 is correct.

Case 2: $i = j + 1$.

$$\begin{aligned} &Pr[Y_{j+1} = y_{j+1}|E_{j-1}, Y_j = y_j] \\ &= Pr[Y_j = y_j|E_{j+1}] \cdot \frac{Pr[Y_{j+1} = y_{j+1}|E_{j-1}]}{Pr[Y_j = y_j|E_{j-1}]} \\ &= Pr[Y_j = y_j|Y_{j+1} = y_{j+1}, Y_{j-1} = y_{j-1}] \cdot \\ &\qquad \frac{Pr[Y_{j+1} = y_{j+1}|Y_{j-1} = y_{j-1}]}{Pr[Y_j = y_j|Y_{j-1} = y_{j-1}]}. \end{aligned} \tag{21}$$

Plugging Equations 18, 19 and 20 into Equation 21, we obtain

$$\begin{aligned} &Pr[Y_{j+1} = y_{j+1}|E_{j-1}, Y_j = y_j] \\ &= \begin{cases} \frac{p_{j+1}}{p_j} + (1 - \frac{p_{j+1}}{p_j})/s & \text{if } y_{j+1} = y_j \\ (1 - \frac{p_{j+1}}{p_j})/s & \text{if } y_{j+1} \neq y_j \end{cases} \end{aligned} \tag{22}$$

We now show $Pr[Y_{j+1} = y_{j+1}|Y_j = y_j]$ has the same expression as in Equation 22, which will validate both Equations 9 and 10. First,

$$\begin{aligned} &Pr[Y_{j+1} = y_{j+1}|Y_j = y_j] \\ &= \sum_{y_{j-1}\in DOM} \Big(Pr[Y_j = y_j|Y_{j+1} = y_{j+1}, Y_{j-1} = y_{j-1}] \cdot \\ &\qquad Pr[Y_{j+1} = y_{j+1}|Y_{j-1} = y_{j-1}] \frac{Pr[Y_{j-1} = y_{j-1}]}{Pr[Y_j = y_j]} \Big). \end{aligned} \tag{23}$$

On the other hand,

$$\begin{aligned} &Pr[Y_j = y_j] \\ &= \sum_{y_{j-1}\in DOM} Pr[Y = y_j|Y_{j-1} = y_{j-1}] Pr[Y_{j-1} = y_{j-1}] \\ &= (p_j/p_{j-1}) \cdot Pr[Y_{j-1} = y_j] + (1 - p_j/p_{j-1})/s. \end{aligned} \tag{24}$$

To compute $Pr[Y_{j+1} = y_{j+1}|Y_j = y_j]$, we fit Equations 19, 20 and 24 into Equation 23, and simplify the resulting formulae as much as possible. During the derivation, the unfriendly term $Pr[Y_{j-1} = y_j]$ cancels nicely. At the end, we arrive an expression that is exactly the one shown in Equation 22.

Case 3: $i \in [j + 2, k + 1]$. Equation 10 holds directly from the inductive hypothesis, because it is not related to Y_j. To verify Equation 9, derive

$$\begin{aligned} &Pr[Y_i = y_i|E_{i-1}, Y_j = y_j] \\ &= Pr[Y_j = y_j|E_i] \frac{Pr[E_i]}{Pr[E_{i-1}, Y_j = y_j]} \\ &= Pr[Y_j = y_j|Y_{j+1} = y_{j+1}, Y_{j-1} = y_{j-1}] \cdot \\ &\qquad \frac{Pr[Y_i = y_i|E_{i-1}]}{Pr[Y_j = y_j|E_{i-1}]} \\ &= Pr[Y_j = y_j|Y_{j+1} = y_{j+1}, Y_{j-1} = y_{j-1}] \cdot \\ &\qquad \frac{Pr[Y_i = y_i|Y_{i-1} = y_{i-1}]}{Pr[Y_j = y_j|Y_{j+1} = y_{j+1}, Y_{j-1} = y_{j-1}]} \\ &= Pr[Y_i = y_i|Y_{i-1} = y_{i-1}]. \end{aligned}$$

Appendix 2. Reconstruction probability

We borrow the notations D, t, m, p_i ($1 \leq i \leq m$), X, Y_i, y_i defined in Section 5.2. The reconstruction probability of t equals:

$$Pr[X = t[A]|Y_1 = y_1, ..., Y_m = y_m], \tag{25}$$

namely, the probability that the collusion of all m recipients reveals the actual sensitive value $t[A]$ of t. Under independent perturbation, Equation 25 evolves into

$$\begin{aligned} &\frac{Pr[Y_1 = y_1, ..., Y_m = y_m|X = t[A]] \cdot Pr[X = t[A]]}{Pr[Y_1 = y_1, ..., Y_m = y_m]} \\ &= \frac{Pr[X = t[A]] \cdot \prod_{i=1}^{m} Pr[Y_i = y_i|X = t[A]]}{\sum_{\forall x} \left(Pr[X = x] \cdot \prod_{i=1}^{m} Pr[Y_i = y_i|X = x] \right)}, \end{aligned}$$

where $Pr[Y_i = y_i|X = x]$ is given in Equation 1. Under the proposed algorithm *multi-pert*, Equation 25 equals $Pr[X = t[A]|Y_1 = y_1]$, which, in turn, can be solved as

$$\frac{Pr[X = t[A]] \cdot Pr[Y_1 = y_1|X = t[A]]}{\sum_{\forall x} \left(Pr[X = x] \cdot Pr[Y_1 = y_1|X = x] \right)}.$$