Austrian Journal of Statistics
2025, Volume 54, 136–156.
http://www.ajs.or.at/
doi:10.17713/ajs.v54i4.2131

cellKey - An R Package to Perturb Statistical Tables

Bernhard Meindl
Statistics Austria
Vienna

Abstract

National statistical offices (NSIs) routinely publish aggregated data in the form of statistical tables. However, ensuring data privacy is a critical aspect of this process. Anonymization techniques must be applied to these tables to safeguard the privacy of individual data contributors and prevent unauthorized inference about specific units from the published outputs as often required by law. The R package **cellKey** offers a possible solution to this challenge by implementing a post-tabular perturbation method for statistical tables. This method modifies table cell values after aggregation, ensuring that sensitive information is adequately masked. It is versatile, suitable for both frequency tables and magnitude tables. A key feature of the **cellKey** package is its ability to maintain consistency across multiple tables that share identical cells. This ensures that anonymized outputs across different tables remains coherent while still protecting privacy. This approach makes the method especially useful for scenarios involving complex datasets with interrelated tables.

This paper describes the methodological background of the perturbation method for tabular data focusing on differences between frequency and magnitude tables. It also provides a step-by-step guide on applying it using the presented R package in a practical example. Additionally, we evaluate the impact of the perturbation on data utility and privacy protection, offering insights into its effectiveness. The **cellKey** package is user-friendly and can empower NSIs and other data holders to publish statistical outputs that uphold both data utility and privacy, meeting the growing demands for secure and accessible data dissemination.

Keywords: statistical tables, data privacy, anonymization, post-tabular perturbation, data consistency, R.

1. Introduction

Statistical Disclosure Control (SDC) is essential for NSIs to protect the confidentiality of respondents while ensuring that published output remains useful. NSIs collect large amounts of information from individuals and businesses, from which aggregated statistics are computed and subsequently published. However, such aggregated results as statistical tables can create disclosure risks if individual contributors can be identified or if specific information about them can be inferred.

This is a critical issue for NSIs, which must ensure that published data remain both useful and confidential. Without proper protection, sensitive information about individuals or businesses could be exposed, violating privacy regulations and potentially leading to a loss of trust in official statistics.

Statistical tables can generally be divided into two types:

Frequency tables show the number of individuals or entities that fall into specific categories of one or more variables in a data set. However, frequency tables can pose privacy risks if some categories contain very few contributors, as this can make it easier to identify individuals.

Magnitude tables summarize numerical values such as income or turnover for different groups. These tables also require protection because a small number of dominant contributors in a category could make it possible to infer sensitive information about specific units contributing to the table.

By applying SDC techniques such as cell suppression (Fischetti and Salazar-González 2001) or perturbation, NSIs reduce the chances of identifying individual respondents while preserving the quality of statistical outputs. For an overview of disclosure risks for frequency and magnitude tables, see, e.g., Minami and Abe (2019).

This contribution introduces the R (R Core Team 2011) package **cellKey** (Meindl 2023a) which can be used to apply the Cell Key Method (CKM) (Leaver and Marley 2011; Thompson, Broadfoot, and Elazar 2013; Meindl and Enderle 2019) to statistical tables. The main idea of this post-tabular Statistical Disclosure Control (SDC) technique is to perturb table cells with random noise in a consistent and reproducible way. This ensures that any given table cell to which the same units contribute is always perturbed consistently.

The R package **ptable** (Enderle 2023) is used to define probabilities of random noise through perturbation tables (ptables). Such ptables specify the likelihood of transitioning from an initial value, often the original cell counts, to corresponding target values. While primarily designed for integration with the **cellKey** package, ptables generated by package **ptable** can also be utilized in other software tools, such as τ-Argus (Statistics Netherlands (CBS) 2006), which also features an implementation of the Cell Key Method.

The package was initially developed within the framework of the Eurostat-funded project "Open Source tools for perturbative confidentiality methods" (Eurostat 2019). This project involved the practical implementation of the method into an open-source SDC tool that can easily be used in real-world applications.

To simplify testing and development, separate packages were provided that implement the method (cellKey) and the generation of the required perturbation tables (ptable) in R. As suggested in Leaver and Marley (2011) and further enhanced in Giessing (2016), the **ptable** package implements a maximum entropy approach to compute the transition probabilities taking into account certain thresholds of the desired noise distribution. **cellKey** makes use of these perturbation tables and allows to compute perturbed (weighted) frequency and magnitude tables. It also allows for an easy way to specify complex hierarchies using the R package **sdcHierarchies** (Meindl 2024) and proves to be quite fast.

The article is structured as follows. In section 2, the key ideas of the perturbation procedure for both count (2.1) and magnitude tables (2.2) are introduced. In chapter 3, a practical example is given on how the **cellKey** package can be used to consistently perturb a statistical table. The article is finally concluded with a summary and discussion of the results in section 4 which also outlines possible areas of future work.

2. Methodological background

This section summarizes key differences between the approach used in the R package **cellKey** facilitated perturbation parameters provided by the **ptable** package and original proposition of the CKM (Leaver and Marley 2011; Thompson *et al.* 2013). The basic idea of the Cell Key

Method is to provide *"on-the-fly"* protection of tables that can be dynamically generated, e.g., through web interfaces. This makes it for example difficult to protect such data using other protection limitation techniques such as cell suppression, for which all possible tables need to be known in advance.

The main advantage of this post-tabular anonymization technique is its ability to ensure consistent perturbation across all table cells. The method operates on microdata, where each record is assigned a unique, random record key (rkey) once. Whenever a table cell is computed, the rkeys of the contributing units are aggregated to a cell key (ckey). This cell key is then used to retrieve a perturbation value from a perturbation table (ptable), which is subsequently added to the original (weighted) cell value. The result, the perturbed cell value, is then returned as the result of the query. As identical cells, formed by the same contributing records, always lead to the same cell keys, the method ensures that the same perturbation value is consistently applied to those cells across dynamically generated tables.

A significant limitation of the CKM, and a key reason why NSIs are hesitant to adopt the method in practice, is its non-additivity. Because cells are perturbed independently, linear constraints, such as those ensuring consistency between marginal totals and the sums of their contributing inner cells, are typically not preserved. As a result, the perturbed marginal totals often differ from the sum of the perturbed values of the contributing inner cells.

2.1. Frequency table perturbation

One of the main differences between the original method and implementation the the **cellKey** package refers to the generation of record keys. In this implementation, the record keys are assumed to be derived from the standard uniform distribution with minimum 0 and maximum 1. The cell keys are then derived using the following formula:

$$ckey_i = \left(\sum_{i=1}^{n} rkey_i \right) \%\% \, 1 \tag{1}$$

The computation of a cell key for a specific cell i is performed by adding up the record keys of the n contributing units to this cell and using the remainder (modulo operation) of a division by 1 as the cell key. The modulo operation (%%) extracts the fractional part of the sum over all randomly generated record keys.

Another simplification pertains to the format of the perturbation tables that are used from package **ptable**. The tables feature a quite generic format. Table 1 shows the first few rows of a possible perturbation table for frequency tables.

Table 1: Example of a possible perturbation table for a frequency table

i	p	v	p_int_lb	p_int_ub
0	1.0000000	0	0.0000000	1.0000000
1	0.5165283	-1	0.0000000	0.5165283
1	0.4508303	1	0.5165283	0.9673586
1	0.0322262	2	0.9673586	0.9995848
1	0.0004152	3	0.9995848	1.0000000
2	0.1666578	-2	0.0000000	0.1666578

Table 1 features the following information:

- *"i"*: Identifies a *block* within which a specific perturbation value is located.
- *"p"*: Represents the probability of selecting a particular perturbation value for the given block *"i"*.

- *"v"*: Specifies the perturbation value (noise) to be added to the original (weighted) cell value.
- *"p_int_lb"*: Indicates the lower bound of the cumulative probability interval, determined by the value of *"p"*.
- *"p_int_ub"*: Indicates the upper bound of the cumulative probability interval, determined by the value of *"p"*.

The probability intervals defined by *"p_int_lb"* and *"p_int_ub"* are arranged in increasing order based on the values in column *"p"*. Since the probabilities in column *"p"* for each block *"i"* must sum to 1, these intervals collectively span the entire probability range $[0, 1]$ for each block.

The look-up procedure for a specific unperturbed cell frequency f is then as follows:

The first step is to determine the perturbation block where $min(f, max(i)) == i$ from Table 1 holds. Once the appropriate block is identified, the cell key calculated using Equation 1 is used to locate the corresponding row within the block. This is done by finding the perturbation interval, defined by *"p_int_lb"* and *"p_int_ub"*, into which the cell key falls. Since all cell keys, as derived from Equation 1, are constrained within the range $[0, 1]$, they will always fall within one of these intervals. Once a specific row has been identified, the value from column *"v"* is added to the unperturbed count. The resulting perturbed value then can be published.

Examining the first few rows of the perturbation table in Table 1, we can observe that an original cell value of 0 will never be perturbed. This is because there is only one row where $i = 0$, and the corresponding probability interval $[0, 1]$ covers the entire range. As a result, any cell with an unperturbed frequency of 0 will always have a perturbation value of 0 (column *"v"*) and remain unchanged. This behavior is often desirable for NSIs, as cells with counts of 0 can represent structural zeros, values that must remain 0 and should not be perturbed to values $\neq 0$.

For a cell with an original value of 1, the following possible perturbation outcomes exist:

- Case 1: $v = -1$ with a probability $p \sim 51.65\%$
- Case 2: $v = 1$ with a probability $p \sim 45.08\%$
- Case 3: $v = 2$ with a probability $p \sim 3.22\%$
- Case 4: $v = 3$ with a probability $p \sim 0.04\%$

These cases result in perturbed cell values of 0, 2, 3 and 4 for Cases 1, 2, 3 and 4, respectively.

2.2. Magnitude table perturbation

The **cellKey** package also allows for the perturbation of magnitude tables. Cells in such tables represent quantities or totals rather than counts. The general idea for the CKM (Leaver and Marley 2011) was to derive perturbed (weighted) cell value pY based on the following formula

$$pY = Y + \sum_{i=1}^{top_k} f(y_i) \cdot m_i \cdot d_i \cdot v \tag{2}$$

where $Y = \sum_{i=1}^{n} y_i \cdot w_i$ is the weighted, unperturbed cell value consisting of values y_i of observation i with corresponding weights w_i. The second term in Equation 2 is the total amount of perturbation that should be added to cell i based on the top_k largest contributors to the cell. In the implementation it is assumed that the contributions y_i can be ordered by absolute descending values. According to Equation 2, the noise term is computed by adding top_k separate noise components where each of these components consists of three multiplicative components:

- $f(y_i)$: A component that is dependent on the data.
- m_i: Fixed factors defined by the user.
- $d_i \cdot v$: Random components, where d_i specifies the direction, and v_i determines the magnitude of the intended perturbation.

It is important to note that suggesting specific values for parameters m_i is not practical, as these values must be considered together with the corresponding perturbation values (v_i) derived from a specific perturbation table. As seen in Equation 2, m_i acts as a multiplication factor, meaning that higher values result in greater noise being added to the original cell value.

Instead of only using the top_k largest contributors when computing the noise term for cell i, the idea was expanded upon and generalized (Eurostat 2019). In **cellKey** it is therefore also possible to use the following alternatives to the contributions of the largest top_k contributors in deriving the noise term:

The desired variant can be set with argument `type` in function `ck_params_nums()` which is shown in 3.2.2 in more detail. The following choices are possible.

- *"top_contr"*: The default variant where the contributions of the top_k largest units values the cell are used.
- *"mean"*: The arithmetic (weighted) mean over all cell values is used.
- *"range"*: The range of values is used.
- *"sum"*: The original, weighted cell value itself is used.

If any option other than *"top_contr"* is selected, top_k is automatically set to 1.

Furthermore, the fixed factors m_i can also be made dependent on actual cell values $x_i = f(y_i)$, leading to the development of the so-called *flex approach*, which is also available in the **cellKey** package. In this approach, users specify a flexpoint (fp) and two percentage values (σ_0 and σ_1) representing the desired magnitudes for large and small cell values, respectively. The flex function shown in Equation 3 is constructed so that for large values of $x_i => fp$ exceeding the flexpoint fp, the output approaches σ_0. For x_i that are smaller or equal the chosen flexpoint, σ_1 is used as perturbation magnitude.

$$m_i(x_i) = \begin{cases} \sigma_0 \cdot \left(1 + \frac{\sigma_1 \cdot x_i - \sigma_0 \cdot fp}{\sigma_0 \cdot fp} \cdot \left(\frac{2 \cdot fp}{fp + x_i}\right)^q\right) & \forall x_i > fp \\ \sigma_1 & \forall x_i \leq fp \end{cases} \tag{3}$$

There are also adjustments to the third noise-component from Equation 2 in a way that the implementation allows to add a random component $d_i \cdot \mu_c + |v_i|$ that depends on cell keys. $\mu_c > 0$ can be considered as fixed, additional noise that is added to specific cells.

In typical applications it can be useful to specify this parameter to allow extra protection to be applied to sensitive cells that could be identified according to some rules. The idea would be to define $\mu_c > 0$ for all identified sensitive cells. In the current application, the amount is added to the largest cell-contributor for each sensitive cell with a direction d_i derived from $sign(v_i)$ from the perturbation value v_i. This value v_i is obtained through a lookup operation on a predefined perturbation table.

The `ck_params_nums()` function includes additional parameters that enhance flexibility in the computation of cell keys. The argument `use_zero_rkeys` allows users to specify whether record keys of units that do not contribute to a cell in a specific magnitude table should be included or ignored when calculating cell keys. Additionally, the argument `ptab` enables users to define multiple perturbation parameters. These parameters can be configured to apply different perturbation rules based on the characteristics of the cells, such as whether the number of contributing units is even or odd, or if the cell only contains a small number of contributing units.

When defining perturbation tables for magnitude tables, using the same format as for frequency tables is difficult. This is because it is virtually impossible to create blocks for every possible cell value, given the continuous nature of the data in magnitude tables. To address this challenge, a more feasible approach involves computing convex combinations of two different look-up results. This method which is implemented in **cellKey*** allows for greater flexibility and avoids the need for an exhaustive set of blocks in the ptables for continuous variables.

Table 2: Example of a possible perturbation table for a magnitude table

i	j	p	v	p_int_lb	p_int_ub
0	0	1.0000000	0	0.0000000	1.0000000
1	0	0.3725250	-1	0.0000000	0.3725250
1	1	0.3725250	0	0.3725250	0.7450499
1	2	0.1662882	1	0.7450499	0.9113381
1	3	0.0650915	2	0.9113381	0.9764296
1	4	0.0188726	3	0.9764296	0.9953022
1	5	0.0040531	4	0.9953022	0.9993553
1	6	0.0006447	5	0.9993553	1.0000000
5	0	0.0000026	-5	0.0000000	0.0000026
5	1	0.0001912	-4	0.0000026	0.0001938
5	2	0.0053586	-3	0.0001938	0.0055524
5	3	0.0579547	-2	0.0055524	0.0635071
5	4	0.2418291	-1	0.0635071	0.3053362
5	5	0.3893276	0	0.3053362	0.6946638
5	6	0.2418291	1	0.6946638	0.9364929
5	7	0.0579547	2	0.9364929	0.9944476
5	8	0.0053586	3	0.9944476	0.9998062
5	9	0.0001912	4	0.9998062	0.9999974
5	10	0.0000026	5	0.9999974	1.0000000

An example perturbation table for continuous data is shown in Table 2. One can see that this very simple perturbation table features three distinct blocks i with its values being 0, 1, and 5. The lookup-procedure given for an unperturbed, (weighted) cell value x with cell key ck and its largest contribution defined as y_i is now as described.

Let $a = \frac{y1}{x}$ be a function of the largest contribution and the total cell value. a is then used to derive a perturbation parameter from Table 2. The following cases are possible:

- $a == i$: There is an exact match with a block number which can be used to look for a perturbation value
- $a > max(i)$: A perturbation value needs to be found in the block where $i == max(i)$ holds.
- $0 < a < max(i)$: In this case, a linear combination of two perturbation values is used to derive a perturbation value.

The first two cases can easily be resolved and the look-up procedure works just as in the cast for count variables. The cell key (ck) is used to find the matching cumulative perturbation interval defined by variable *"p_int_lb"* and *"p_int_ub"* for block $a == i$ (first case) or $a == max(i)$ in the second case. The third case is a bit more difficult. The main idea is to compute a weighted combination of two different perturbation values as it will now be discussed in an example. We assume that for a specific cell a equals 3.2 and the cell key equals 0.35. We can compute a perturbation value v using the following steps:

- Let a_0 represent the largest value of i less than a, resulting $a_0 = 1$.
- Let a_1 represent the smallest value of i larger than a, resulting $a_1 = 5$.
- Using the values a_0 and a_1 along with the known cell key ($ck = 0.35$) from Table 2, determine two different perturbation values, v_0 and v_1. These are given by $v_0 = -1$ and $v_1 = 0$.
- Calculate a weighted average for the final perturbation value using the formula

$$v = (1 - \lambda) \cdot v_0 + \lambda \cdot v_1$$

where $\lambda = \frac{a-a_0}{a_1-a_0}$. In this example, $\lambda = 0.55$ resulting in a final perturbation value of $v = -0.45$.

Unlike the perturbation table for frequency variables, the column i in a perturbation table for continuous variables is not restricted to integers and can also include real numbers which makes customization even more granular. The implementation in the **cellKey** package also leverages this structure to allow further generalizations, such as supporting different perturbation tables for even and odd numbers.

3. A practical guide to using cellKey

This chapter will demonstrate the usage of the **cellKey** package. It will provide a step-by-step guide for producing perturbed frequency and magnitude tables.

3.1. Setting up a problem instance

To install the **cellKey** package from the Comprehensive R Archive Network (CRAN), one can use the command from Listing 1 in your R console:

```r
install.packages("cellKey")
```

Listing 1: Install the latest version of the **cellKey** package

This command downloads and installs the latest version of the package. Once the installation is complete, you can load the package into your R session and check the version as shown in Listing 2:

```r
library(cellKey)
packageVersion("cellKey")
```

Listing 2: Load package and verify package version

```
## [1] '1.0.2'
```

The initial step in this approach involves generating a statistical table using the `ck_setup()` function. This function creates an object that encompasses all necessary information for perturbing count (and optionally continuously scaled) variables.

Prior to using `ck_setup()`, several inputs must be prepared. These inputs include:

- `x`: A `data.frame` or `data.table` containing the micro-data.
- `rkey`: Specifies how to define record keys.
  - If a number is provided, unique record keys are generated internally by sampling from a uniform distribution and rounding the results to the specified number of digits.
  - If a character string is provided, it is interpreted as the name of a variable within x that already contains record keys.
- `dims`: A named list where each element represents a hierarchical structure created using functions from the **sdcHierarchies** package. Each name in the list corresponds to a variable in the `x` dataset.
- `w`: Specifies the name of a variable in x that contains weights, or `NULL` if no weights are present.
- `countvars`: An optional character vector specifying the names of variables in `x` that hold count values. A special count variable, named *"total"* is generated internally, assigning a value of 1 for each row in `x`.
- `numvars`: An optional character vector specifying the names of variables in x that hold numerical values and can be subsequently perturbed.

The subsequent steps will demonstrate how to generate these required inputs. The first step involves preparing the input data.

Preparing input data

The test data used in this example contains individual-level information, including sampling weights, categorical variables, and continuously scaled variables. Listing 3 shows the relevant code where also a binary variable (*"cnt_highincome"*) is computed for which a perturbed table will also be computed in section 3.

```r
dat <- ck_create_testdata()
dat <- dat[, c("sex", "age", "savings", "income", "sampling_weight")]
dat[, cnt_highincome := ifelse(income >= 9000, 1, 0)]
```

Listing 3: Load test data

Setting up record keys and dimensions

If no suitable record keys exist, they can be added to the dataset using function `ck_generate_rkeys()`, with each record key having 7 digits. These record keys will later be referenced in the `ck_setup()` function. The process is demonstrated in Listing 4.

```r
# Generate record keys with 7 digits and add them to the dataset
dat$rkeys <- ck_generate_rkeys(
  dat = dat,
  nr_digits = 7
)
print(head(dat))
```

Listing 4: Sample record keys

To guarantee that the same record keys are generated consistently for the same input, the `ck_generate_rkeys()` function computes a seed by default based on the `SHA-1` hash (Eastlake and Jones 2001) of the input dataset. This seed is computed prior to sampling the record keys, ensuring reproducibility of the generated keys across runs. By default, this feature is enabled and ensures deterministic behavior, while the optional `seed` argument allows for greater flexibility when required. The first rows of the resulting table data input is shown in Table 3. We note that each record key is bounded in $[0, 1]$.

Table 3: Example micro-data including record keys

sex	age	savings	income	sampling_weight	cnt_highincome	rkeys
male	age_group3	12	5780	87	0	0.8232870
female	age_group3	28	2530	58	0	0.4815691
male	age_group1	550	6920	20	0	0.5541220
male	age_group1	870	7960	53	0	0.3970927
male	age_group4	20	9030	62	1	0.0457354
female	age_group3	102	3290	33	0	0.0822132

The goal of the following example is to create perturbed tables for a) the counts of *"sex"* by *"age"* for all observations and for subgroups where *"cnt_highincome"* is non-zero, and b) for the continuously scaled variables *"savings"* and *"income"*, both by the hierarchical structure defined by variables *"sex"* and *"age"*.

Hierarchies must be defined for each classifying variable in the desired statistical table. Two methods are available for specifying the hierarchical structure of these variables (including subtotals).

The first method uses the `@` value format, similar to the approach used in the **sdcTable** package (Meindl 2023b). However, the recommended method takes advantage of the **sdcHierarchies** (Meindl 2024) package, which enables the dynamic creation and modification of hierarchy objects. In this example, this method is used to define the hierarchies for the variables *"age"* and *"sex"* which is shown in Listings 5.

```r
dim_sex <- sdcHierarchies::hier_create(
  root = "Total",
  nodes = c("male", "female")
)

dim_age <- sdcHierarchies::hier_create(
  root = "Total",
  nodes = paste0("age_group", 1:6)
)
```

Listing 5: Define hierarchies for variables *"sex"* and *"age"*

Both hierarchies are *"hierarchical"* because they include a total value that can be derived from other values within the hierarchy. For the variable *"sex"*, the total code *"Total"* is calculated by summing the contributions of the nodes *"male"* and *"female"*.

The **sdcHierarchies** package provides a framework for creating and managing complex hierarchical structures. The `hier_create()` function is used to create a tree object representing the desired hierarchy. This tree object can then be modified using functions such as `hier_add()` to add elements, `hier_delete()` to remove elements, and `hier_rename()` to rename elements within the hierarchy.

The package includes a vignette, accessible through `sdcHierarchies::hier_vignette()`, which provides examples and demonstrates the usage of these functions. Additionally, the interactive `hier_app()` allows users to modify and visualize hierarchies and convert between different data formats.

Once all dimensions have been defined, these inputs must be combined into a named list. In this list, the names correspond to the variable names in the input dataset, and the elements contain the respective hierarchy specifications created in Listing 5 from above. Therefore, in this example, the list elements of `dims` must be named *"sex"* and *"age"* since the specifications refer to the *"age"* and *"sex"* variables within the input dataset. The relevant code is shown in Listing 6.

```
dims <- list(sex = dim_sex, age = dim_age)
```

Listing 6: Create a named input-list with all relevant dimensional variables

Setup a table

Having prepared the necessary inputs, we can now define a generic statistical table using the `ck_setup()` function as shown in Listing 7:

```
tab <- ck_setup(
  x = dat,
  rkey = "rkeys",
  dims = dims,
  w = "sampling_weight",
  countvars = "cnt_highincome",
  numvars = c("income", "savings")
)
```

Listing 7: Setting up a table object

The `ck_setup()` function returns an `R6` (Chang *et al.* 2024) class object. This object encapsulates both the relevant data and all available methods, enabling a streamlined workflow. Method calls directly modify the object itself, eliminating the need for explicit reassignment. Furthermore, these objects incorporate different methods to obtain current information about the object. For example, it is possible using `$hierarchy_info()` to get information about the dimensional variables spanning the table as demonstrated in Listing 8

```
tab$hierarchy_info()
```

Listing 8: Display information about dimensional variables

```
## $sex
##      code level is_leaf parent
##    <char> <int>  <lgcl> <char>
## 1:  Total     1   FALSE  Total
## 2:   male     2    TRUE  Total
## 3: female     2    TRUE  Total
##
## $age
##          code level is_leaf parent
##        <char> <int>  <lgcl> <char>
## 1:      Total     1   FALSE  Total
## 2: age_group1     2    TRUE  Total
## 3: age_group2     2    TRUE  Total
## 4: age_group3     2    TRUE  Total
## 5: age_group4     2    TRUE  Total
## 6: age_group5     2    TRUE  Total
## 7: age_group6     2    TRUE  Total
```

or by utilizing methods `$cntvars()` or `$numvars()` to show, which variables are available and can be tabulated and/or perturbed. Method `$allvars()` shows both count- and numeric variables as shown in Listing 9:

```
tab$hierarchy_info()
```

Listing 9: Extracting available count- and numeric variables from a table instance

```
## $cntvars
## [1] "total"          "cnt_highincome"
##
## $numvars
## [1] "income"  "savings"
```

As shown in the output, a count variable named *"total"* covering the complete table is automatically generated and does not need to be explicitly specified. A custom print method for displaying general object information is also implemented as shown in Listing 10. The output also indicates variables that already have been perturbed which in this case has not yet been done.

```
tab$hierarchy_info()
```

Listing 10: Display information about the table instance using a custom print method

```
## -- Table Information ------------------------------------------
##    21 cells in 2 dimensions ('sex', 'age')
##    weights: yes
## -- Tabulated / Perturbed countvars ----------------------------
## [ ] 'total'
## [ ] 'cnt_highincome'
## -- Tabulated / Perturbed numvars ------------------------------
## [ ] 'income'
## [ ] 'savings'
```

3.2. Configuring perturbation parameters

After generating a table instance, the next step is to create perturbation parameters and map those to existing variables in the table object.

Perturbation parameters for count variables

The next step involves defining the parameters used to perturb count variables. This is achieved using the `ck_params_cnts()` function. This function requires as input the result of `create_cnt_ptable()` from the **ptable** package. For detailed information on the required parameters, please refer to the **ptable** package documentation.

The next example will feature two different ptables. One will use an exemplary ptable directly provided by the ptable package using `pt_ex_cnts()`, while the other will be created by explicitly specifying parameters in function `create_cnt_ptable()`. The required code is shown in Listing 11.

```r
# Create perturbation parameters suitable for count variables
# using an example table from the ptables package
p_cnts1 <- ptable::pt_ex_cnts() |>
  ck_params_cnts()

# and with customized parameters
p_cnts2 <- ptable::create_cnt_ptable(
  D = 8, V = 3, js = 2, pstay = 0.5
) |> ck_params_cnts()
```

Listing 11: Creating two different perturbation objects suitable for count data

In the above code, we computed two different ptables and also modified the structure using `ck_params_cnts()`. This modification allows the code to use the resulting objects, `p_cnts1` and `p_cnts2`, in subsequent steps. Using `params_cnts_set()`, we have the possibility to map a set of perturbation parameters (like `p_cnts1` or `p_cnts2`) to specific count variables defined with the `v` argument. If `v` is not specified, the perturbation parameter set applies to all count variables.

The following code snippets shows how perturbation parameters `p_cnts1` can be mapped to variable *"total"* (Listing 12) which always exists

```r
tab$params_cnts_set(
  val = p_cnts1,
  v = "total"
)
```

Listing 12: Map perturbation object to count variable *"total"*

and to `p_cnts2` for variable *"cnt_highincome"* (Listing 13).

```r
tab$params_cnts_set(
  val = p_cnts2,
  v = "cnt_highincome"
)
```

Listing 13: Map perturbation object to count variable *"cnt_highincome"*

It should be noted that the output of both methods will indicate if the mapping process was successful. This output was omitted from this article to improve readability.

This implementation allows for the flexibility to use different parameter sets for different count variables. Modifying perturbation parameters for specific variables is also straightforward. Simply re-apply the `params_cnts_set()` method, and it will overwrite any previously defined parameters for specified variables in the table.

Perturbation parameters for continuous variables

The `ck_params_num()` function is used to create input objects for defining perturbation parameters for continuous variables. These input objects are then mapped to continuous variables in a specific table object using the `params_nums_set()` method. As already discussed in 2.2, the package implements a quite broad range of possibilities on how to define the underlying perturbation parameters for continuously scaled variables which can be specified in `ck_params_nums()` using argument `type`. Detailed information about these options is available on the corresponding manual page, which can be accessed by entering `?ck_params_nums` in the R console.

It should be noted that both `ck_params_nums()` and `ck_params_cnts()` functions provide a `path` argument. This allows users to save the defined parameters as a `YAML` file which can be imported again using the `ck_read_yaml()` function. This feature of saving and reusing parameter settings enhances efficiency and reproducibility in the analysis.

To utilize the dynamic multiplier generalization (refer to Equation 3 for details), the output of utility functions `ck_flexparams()` or `ck_simpleparams()` must be passed to the `mult_params` argument in the `ck_params_nums()` function. The argument `mu_c` is used to specify an additional fixed amount of perturbation to be applied to sensitive cells. Meanwhile, the arguments `top_k` and `type` together define the data-dependent components on which the perturbation values should be calculated. Finally, the `ptab` argument allows the specification of one or more perturbation tables. These tables can be applied selectively to all cells, even or odd cells, or small cells. This functionality integrates with the **ptable** package, which provides the `ptable::create_num_ptable()` function for generating appropriate perturbation tables as inputs. For further information of parameters of this function, please refer to the corresponding manual.

In this example, the objective is to create a single perturbation object that can be used for both numerical variables requiring perturbation. However, this object should incorporate different perturbation tables to handle cells with an odd and even number of contributors. Additionally, the flex-approach will be utilized, allowing different magnitudes of noise to be applied to large and small cells.

The first step is to define two distinct perturbation tables, one for odd and one for even numbers of contributors. This can be accomplished using the `ptable::create_nums_ptable()` function, which generates appropriate perturbation tables that can then be used as an input in `ck_params_nums()`. The required code is shown in Listing 14 below.

```r
ptable_even <- ptable::create_nums_ptable(
  D = 5,
  V = 1.1,
  icat = c(1, 3, 5)
)

ptable_odd <- ptable::create_nums_ptable(
  D = 10,
  V = 2,
  icat = c(1, 5, 10)
)
```

Listing 14: Define different perturbation tables suitable for numerical variables

The next step is to define the parameters for the flex function. In this example, the flexpoint `fp` is set to 1000, which determines the point where the noise coefficient function reaches its maximum value. The parameter `p` is a numeric vector of length 2, with the condition that $p[1]$ is greater than $p[2]$. Both elements represent percentages: the first value specifies the desired maximum perturbation percentage for small cells, based on the flexpoint, while the second value specifies the maximum perturbation percentage for large cells. The parameter `epsilon` is a numeric vector with values between 0 and 1 in descending order, with the first element fixed at 1. Its length must match the number of contributors specified in the `top_k` argument of `ck_params_nums()`, which will be explained later. This configuration enables the use of different flex functions for the largest `top_k` contributors. Finally, the parameter `q` corresponds to a value used in Equation 3. The relevant code is displayed in Listing 15.

```r
# Setup parameters for the flex function to be used
p_flex <- ck_flexparams(
  fp = 1000,
  p = c(0.3, 0.03),
  epsilon = c(1, 0.5, 0.2),
  q = 2
)
```

Listing 15: Setup parameters for the *"flex-approach"*

Note that instead of using `ck_flexparams()`, a more basic approach is implemented in `ck_simpleparams()` which simplifies parameter specification by requiring only a single percentage value (`p`) and a vector of `epsilons` (when `top_k` is greater than 1 and applies a constant perturbation magnitude to all cells, independent on their (weighted) values.

Once these prerequisites have been done, it is possible to finally use `ck_params_nums()` to create a suitable perturbation object as it is shown in Listing 16:

```
p_nums <- ck_params_nums(
  type = "top_contr",
  top_k = 3,
  ptab = list("even" = ptable_even, "odd" = ptable_odd),
  mult_params = p_flex,
  mu_c = 2
)
```

Listing 16: Creating a perturbation object suitable for numeric variables

In the application of `ck_params_nums()`, the argument `mu_c` was set to 2, indicating that an additional amount of perturbation is applied to sensitive cells, which will be identified later in the example. Additionally, by specifying the `ptab` argument as a named list with names corresponding to *"even"* and *"odd"*, it is possible to use different perturbation tables for cells with an even or odd number of contributors, achieving the desired level of customization.

Once the perturbation object has been created, it can be applied to the appropriate variables by mapping it accordingly using method `$params_nums_set()` as demonstrated in Listing 17.

```
tab$params_nums_set(
  v = c("income", "savings"),
  val = p_nums
)
```

Listing 17: Map perturbation object to numeric variables *"income"* and *"savings"*

In this example, the same parameters are used for both numerical variables. However, it is also possible, just as with count variables, to assign different parameters to different variables. This is particularly useful when distinguishing between strictly positive variables (such as *"savings"* and *"income"* in this example) and variables that may include negative values. It is also possible to change perturbation parameter for specific numeric variables just as demonstrated for count variables before by calling the `$params_nums_set()` method again and providing a different parameter object.

In order to make use of parameter `mu_c` that allows ab add extra amount of protection to sensitive cells, one may identify sensitive cells according to some rules. The following methods to identify sensitive cells are implemented in package **cellKey**:

- `$supp_p()`: Identifies sensitive cells based on the p%-rule, where a cell is considered sensitive if the cell total minus the two largest contributions is smaller than p % of the largest contribution. Typical values of pp that achieve a balance between data utility and confidentiality generally fall within the 15% to 30% range.

- `$supp_nk()`: Identifies sensitive cells based on *nk*-dominance, where a cell is sensitive if the sum of the n largest values is greater than k% of the cell total. Commonly used values for k in this rule generally range between 60% and 85%.
- `$supp_freq()`: Identifies sensitive cells by checking whether the (weighted) number of contributors in the cell is below a specified minimum frequency.
- `$supp_val()`: Identifies sensitive cells where the (weighted) cell value exceeds or falls below a specified threshold.
- `$supp_cells()`: Identifies sensitive cells based on their specific names, allowing targeted suppression of predefined cells.

These methods provide flexible options for detecting sensitive cells based on various statistical disclosure control rules.

In this example we now want to set all cells for variable *"income"* as sensitive to which less than 15 units contribute which can be achieved using `$supp_freq()` as shown in Listing 18.

```r
tab$supp_freq(
  v = "income",
  n = 15,
  weighted = FALSE
)
```

Listing 18: Identify sensitive cells according to threshold-rule

The output of the method (again omitted from the article for better readability) shows the number of cells identified and marked as sensitive, which in this case refers to 3 cells.

3.3. Generating perturbed outputs

You can perturb variables mapped to appropriate parameter sets using the `$perturb()` method. This functionality allows for the simultaneous perturbation of multiple variables. In this example (see Listing 19), both available count variables are perturbed concurrently to demonstrate this feature.

```r
tab$perturb(
  v = c("total", "cnt_highincome")
)
```

Listing 19: Perturb count variables *"total"* and *"cnt_highincome"*

The `$perturb()` method also prints information about the perturbed variables to the R terminal, which is omitted from this article. After this call, the object `tab` is updated to include perturbed values for the variable *"total"*. It is important to note that no explicit assignment is needed, as the update is applied directly to the object. The results can then be extracted using the `$freqtab()` method (see Listing 20) which returns the results displayed in Table 4.

```r
tab$freqtab(v = "total")
```

Listing 20: Extracting original and perturbed values for variable *"total"*

Table 4: Results of perturbing count variable *"total"*

sex	age	vname	uwc	wc	puwc	pwc
Total	Total	total	4580	275874	4580	275874.0000
Total	age_group1	total	1969	118309	1970	118369.0858
Total	age_group2	total	1143	69498	1144	69558.8031
Total	age_group3	total	864	51866	864	51866.0000
Total	age_group4	total	423	25135	423	25135.0000
Total	age_group5	total	168	10301	169	10362.3155
Total	age_group6	total	13	765	14	823.8462
male	Total	total	2296	137743	2296	137743.0000
male	age_group1	total	1015	60335	1015	60335.0000
male	age_group2	total	571	34449	570	34388.6690
male	age_group3	total	424	25704	424	25704.0000
male	age_group4	total	195	11696	196	11755.9795
male	age_group5	total	84	5144	84	5144.0000
male	age_group6	total	7	415	5	296.4286
female	Total	total	2284	138131	2285	138191.4777
female	age_group1	total	954	57974	954	57974.0000
female	age_group2	total	572	35049	573	35110.2745
female	age_group3	total	440	26162	439	26102.5409
female	age_group4	total	228	13439	228	13439.0000
female	age_group5	total	84	5157	82	5034.2143
female	age_group6	total	6	350	7	408.3333

The `$freqtab()` method returns a `data.table` (Barrett, Dowle *et al.* 2024) containing detailed information for each cell, defined by the dimensional variables (e.g., *"age"* and *"sex"*), and for each perturbed variable (column *"vname"*). The table includes the following columns:

- *"uwc"*: Unperturbed unweighted counts
- *"wc"*: Unperturbed weighted counts
- *"puwc"*: Perturbed unweighted counts
- *"pwc"*: Perturbed weighted counts

It can be observed from Table 4 that the impact of the perturbation is minimal, but it still introduces non-additivity. If greater distortion is required, the existing perturbation results must first be reset. This can be accomplished using the `$reset_cntvars()` method for count variables or the `$reset_numvars()` method for numeric variables. After resetting, a new perturbation parameter object must be created, mapped to the variable, and the `$perturb()` method re-applied. This possibly iterative process allows for adjustments to the perturbation parameters to achieve the desired level of noise.

To add noise to continuous variables, the `$perturb()` method needs to be used as well, specifying the variables to be perturbed as shown in Listing 21.

```
tab$perturb(
  v = c("income", "savings")
)
```

Listing 21: Perturb count variables

After this step, the results can be extracted, as shown in Listing 22, using the `$numtab()` method. In Table 5, the output for the variable *"savings"* is presented.

```
tab$numtab(
  v = "savings"
)
```

Listing 22: Extracting original and perturbed values for variable *"savings"*

Table 5: Results of perturbing numeric variable *"savings"*

sex	age	vname	uws	ws	pws
Total	Total	savings	2273532	137309328	137308234.1
Total	age_group1	savings	982386	59350762	59347303.4
Total	age_group2	savings	552336	33405236	33410476.6
Total	age_group3	savings	437101	26318762	26318146.7
Total	age_group4	savings	214661	12843203	12837435.1
Total	age_group5	savings	80451	4984343	4987596.1
Total	age_group6	savings	6597	407022	412304.2
male	Total	savings	1159816	69435588	69435588.0
male	age_group1	savings	517660	30818165	30815833.3
male	age_group2	savings	280923	16810501	16806397.9
male	age_group3	savings	214970	13006252	13003331.2
male	age_group4	savings	99420	5922654	5926643.2
male	age_group5	savings	43233	2660516	2662494.9
male	age_group6	savings	3610	217500	212688.1
female	Total	savings	1113716	67873740	67872512.5
female	age_group1	savings	464726	28532597	28532968.0
female	age_group2	savings	271413	16594735	16592702.4
female	age_group3	savings	222131	13312510	13313258.4
female	age_group4	savings	115241	6920549	6923327.2
female	age_group5	savings	37218	2323827	2314098.9
female	age_group6	savings	2987	189522	192895.3

The `$numtab()` method also returns a `data.table` containing detailed information for each cell in the table defined by the dimensional variables. The table includes the following columns:

- *"vname"*: Identifies the continuous variable.
- *"uws"*: The unperturbed unweighted values.
- *"ws"*: The unperturbed weighted values.
- *"pws"*: The perturbed weighted values.

The `$numtab()$` method also allows the use of the `mean_before_sum` argument. If set to `TRUE`, the perturbed values are adjusted using the factor $\frac{n}{n+p}$, where n is the original weighted cell value and p the perturbed cell value. This adjustment prioritizes the accuracy of the variable mean over the accuracy of the variable sums. By default, this argument is set to `FALSE`, meaning no adjustment is applied.

It should also be noted that both the `$freqtab()` and the `$numtab()` method include a `path` argument that allows users to save the resulting tables directly to a *.csv* file. This functionality is particularly useful for exporting perturbed frequency and numeric tables for further analysis or for sharing results with collaborators in a widely compatible format.

3.4. Assessing data utility

For count variables, the `$measures_cnt()` method can be used to calculate a variety of utility statistics that evaluate the impact of the perturbation. These statistics provide insights into how the perturbation has affected the data. `$measures_cnt()` returns a list containing various measures. These measures are explained in detail in the package documentation, which can be accessed using `?cellkey_pkg`. One of the key elements in the returned list is *"overview"*, which provides a `data.table` summarizing the distribution of the noise applied during the perturbation process. An example of this output is shown in Table 6. The relevant code is shown in Listing 23.

```
tab$measures_cnts(v = "total")$overview
```

Listing 23: Extracting Utility Measures for perturbed variable *"total"*

Table 6: Overview on perturbation impact on variable 'total'

noise	cnt	pct
-1	8	0.3809524
0	9	0.4285714
1	2	0.0952381
2	2	0.0952381

Table 6 illustrates the impact of perturbation by showing how the noise distribution can be analyzed to evaluate its effects. The overview element provides a `data.table` with the following columns:

- *"noise"*: The amount of noise, calculated as the difference between the original value (orig) and the perturbed value (pert).
- *"cnt"*: The number of cells that were perturbed by the noise value specified in the noise column.
- *"pct"*: The percentage of cells perturbed by the noise value given in the noise column.

This table allows for a clear assessment of the perturbation's impact by summarizing the distribution and prevalence of the applied noise. For numerical variables, currently no quality measures are implemented which is a topic of future development.

4. Results and discussion

The **cellKey** package contains a robust and flexible approach to statistical disclosure control by implementing a post-tabular perturbation algorithm. The methodology ensures a consistent application of noise across tables, preserving coherence while anonymizing data. This is particularly advantageous for NSIs managing interconnected, linked datasets.

The Cell Key Method is especially effective in situations where the statistical tables required to be published are highly detailed or have to be dynamically generated, making traditional methods like cell suppression impractical. Modern data dissemination often involves tables generated on demand, where users customize queries to suit their needs. In such cases, it is impossible to predict all potential table configurations in advance, which is a requirement for cell suppression to identify sensitive cells and maintain consistency. The CKM addresses this challenge by perturbing table cells dynamically and consistently utilizing only the record keys from the underlying micro-data set. This process ensures that identical cells across different and possibly linked tables receive the same perturbation values as the contributing record keys remain the same.

For highly detailed, high-dimensional statistical tables, the method also provides fast solutions. As the number of cells increases, traditional suppression-based methods become more complex and may require significant manual intervention to produce acceptable results.

Another possibility is that the table's complex structure requires extensive suppression to protect contributing units, leading to significant data loss. As a result, the data's utility may be so reduced that publication becomes unfeasible.

The CKM addresses this issue by perturbing each cell independently while preserving coherence across shared cells, even in high-dimensional tables. This approach also maintains high data utility by adding noise rather than removing cells entirely, preventing the large data loss that is often associated with cell suppression. In addition, the CKM supports automation and scalability, making it suitable for large datasets and dynamic table builders where queries do not need to be known in advance by the data holder. Its computational efficiency enables seamless integration into automated workflows, providing a practical alternative to traditional suppression methods.

The method is very flexible and can easily adapt to changing requirements, allowing adjustments for new datasets or shifting priorities without having to reprocess numerous tables. This flexibility is achieved by defining ptables that can be fine-tuned in detail. For example, it is easily possible to create a perturbation table that prevents the appearance of small values within a specified range in the perturbed, safe table.

This adaptability, combined with its ability to provide privacy while maintaining high data utility, makes the CKM a practical and reliable choice for handling detailed or dynamic tables.

One major limitation of the CKM is its non-additive nature, which poses challenges for its use in NSIs. The publication of non-additive results is often considered as a problem because they may undermine the perceived reliability and trustworthiness of the published data. Users, such as policymakers or researchers, typically expect statistical tables to maintain strict additivity, where marginal totals match the sums of their corresponding inner cells. When not-additive results are published however, concerns such as a perception of inaccuracy or challenges to the further usage of data in other software tools can arise.

However, many users might prefer slightly non-additive results with high data utility to strictly additive tables with low utility caused by extensive suppression. Consequently, it is crucial for an NSI to clearly communicate how the CKM differs from suppression-based approaches, that non-additivity is a feature of the perturbation process and not a problem per-se and to highlight its distinct advantages. This would likely facilitate broader acceptance and practical implementation of the CKM in statistical practice.

Future developments of the **cellKey** package could focus on implementing utility measures for numerical variables, expanding documentation with additional case studies, and enhancing compatibility with other statistical software tools. These enhancements would further improve the package and make it an invaluable toolset allowing NSIs to protect outputs and fulfil requirements regarding data privacy.

References

Barrett T, Dowle M, *et al.* (2024). *data.table: Extension of 'data.frame'*. R package version

1.16.4, URL https://CRAN.R-project.org/package=data.table.

Chang W, *et al.* (2024). *R6: Encapsulated Object-Oriented Programming for R.* R package version 2.5.1, URL https://CRAN.R-project.org/package=R6.

Eastlake D, Jones P (2001). "US Secure Hash Algorithm 1 (SHA1)." *RFC 3174*. URL https://www.rfc-editor.org/rfc/rfc3174.

Enderle T (2023). *ptable: Generation of Perturbation Tables for the Cell-Key Method.* R package version 1.0.0, URL https://CRAN.R-project.org/package=ptable.

Eurostat (2019). "Open Source Tools for Perturbative Confidentiality Methods." https://wayback.archive-it.org/12090/20231228131832/https://cros-legacy.ec.europa.eu/content/perturbative-confidentiality-methods_en. Accessed: 2025-01-16.

Fischetti M, Salazar-González JJ (2001). "Solving the Cell Suppression Problem on Tabular Data with Linear Constraints." *Management Science*, **47**(7), 1008–1027.

Giessing S (2016). "Computational Issues in the Design of Transition Probabilities and Disclosure Risk Estimation for Additive Noise." In *Privacy in Statistical Databases*, volume 9867 of *Lecture Notes in Computer Science*, pp. 237–251. Springer.

Leaver V, Marley JK (2011). "A Method for Confidentialising User-Defined Tables: Statistical Properties and a Risk-Utility Analysis." In *Proceedings of 58th World Statistical Congress*, pp. 1072–1081.

Meindl B (2023a). *cellKey: Consistent Perturbation of Statistical Frequency- and Magnitude Tables.* R package version 1.0.2, URL https://CRAN.R-project.org/package=cellKey.

Meindl B (2023b). *sdcTable: Methods for Statistical Disclosure Control in Tabular Data.* R package version 0.32.6, URL https://CRAN.R-project.org/package=sdcTable.

Meindl B (2024). *sdcHierarchies: Create and (Interactively) Modify Nested Hierarchies.* R package version 0.21.0, URL https://CRAN.R-project.org/package=sdcHierarchies.

Meindl B, Enderle T (2019). "CellKey - Consistent Perturbation of Statistical Tables." In *Proceedings of the UNECE Work Session on Statistical Data Confidentiality*. UNECE, United Nations Economic Commission for Europe (UNECE), The Hague, Netherlands. Statistics Austria and Destatis.

Minami K, Abe Y (2019). "Algorithmic Matching Attacks on Optimally Suppressed Tabular Data." *Algorithms*, **12**(8). ISSN 1999-4893. doi:10.3390/a12080165. URL https://www.mdpi.com/1999-4893/12/8/165.

R Core Team (2011). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

Statistics Netherlands (CBS) (2006). *τ-Argus User Manual.* CENEX Project. Software for Statistical Disclosure Control, URL https://www.cbs.nl/en-gb/menu/methoden/onderzoeksmethoden/statistische-methoden/statistical-disclosure-control/tau-argus/.

Thompson G, Broadfoot S, Elazar D (2013). "Methodology for the Automatic Confidentialisation of Statistical Outputs from Remote Servers at the Australian Bureau of Statistics." In *Proceedings of the Joint UNECE/Eurostat Work Session on Statistical Data Confidentiality*. UNECE/Eurostat, Ottawa, Canada. URL https://unece.org/fileadmin/DAM/stats/documents/ece/ces/ge.46/2013/Topic_1_ABS.pdf.

Affiliation:

Bernhard Meindl
Statistics Austria
AT-1110 Vienna, Austria
E-mail: bernhard.meindl@statistik.gv.at

Austrian Journal of Statistics http://www.ajs.or.at/
published by the Austrian Society of Statistics http://www.osg.or.at/
Volume 54 *Submitted:* 2025-01-27
2025 *Accepted:* 2025-02-17