Rmind: a tool for cryptographically secure statistical analysis

Dan Bogdanov[1], Liina Kamm[1,2], Sven Laur[2] and Ville Sokk[1]

[1] Cybernetica, Tartu, Estonia
{dan, liina, ville.sokk}@cyber.ee
[2] University of Tartu, Institute of Computer Science, Tartu, Estonia
swen@math.ut.ee

Abstract. Secure multi-party computation platforms are becoming more and more practical. This has paved the way for privacy-preserving statistical analysis using secure multi-party computation. Simple statistical analysis functions have been emerging here and there in literature, but no comprehensive system has been compiled. We describe and implement the most used statistical analysis functions in the privacy-preserving setting including simple statistics, t-test, χ^2 test, Wilcoxon tests and linear regression. We give descriptions of the privacy-preserving algorithms and benchmark results that show the feasibility of our solution.

Keywords: Privacy, statistical analysis, hypothesis testing, predictive modelling, cryptography

1 Introduction

Digital databases exist wherever modern computing technology is used. These databases often contain private data, e.g., the behaviour of customers or citizens. Today's data analysis technologies require that the analyst have a direct access to the records in the database, thus creating a fundamental risk to privacy. Even if the data analyst is diligent and keeps secrets, the digital data in his or her possession can leak in an internal or external attack.

Years of information security research has given us strong encryption schemes that can protect databases at rest, i.e., while they are stored before processing. However, today's tools require us to decrypt data before it can be processed, bringing us back to the privacy problem. Emerging cryptographic technologies like secure multi-party computation are a solution to this problem, allowing data to be processed in their encrypted form. Essentially, they create a new kind of computer that does not need to see individual data values, but can still manipulate them—much like a blind craftsman sculpting a work of art.

Secure multi-party computation technologies were considered impractical for years, but clever protocol design and determined engineering have created the first real-world applications, for example, see [11, 10]. This and a wide range of

prototypes mean that we can now build data analysis tools with significantly better protection against both internal and external attacks.

In 2014, the Estonian Data Protection Agency reviewed our proposal for linking and analyzing two personal databases using secure multi-party computation [4]. They concluded that, according to the data protection legislation, secure multi-party computation is not personal information processing and no permit is needed. While this is not the prevailing legal position, it marks a potential paradigm shift in data protection.

In interviews conducted with potential end users of secure multi-party computation technology, they showed great interest, but also had concerns [4]. First, statisticians are used to seeing individual values and are unsure if they can find inconsistencies and ensure analytical quality without such access. Secure multi-party computation takes away control from the analyst so that only query results will be disclosed.

The second concern is the availability of user-friendly environments for performing cryptographically secure data analysis. The steps in the analysis depend on the data and, thus, statisticians use computational environments, like SAS, SPSS or R, to interactively determine the workflow. Hence, potential users expect to see a feature complete drop-in replacement of the environment with additional privacy guarantees.

In this paper, we overcome these challenges. We describe a suite of privacy-preserving algorithms for filtering, descriptive statistics, outlier detection, statistical testing and modelling. We implement these algorithms in Rmind, a privacy-preserving statistical analysis tool designed to provide an experience similar to existing scriptable tools such as the R environment[1]. Rmind provides tools to support all stages of statistical analysis—data collection, exploration, preparation and analysis.

Our contribution. This paper builds on our earlier work in [4, 34] where we report on our first attempts on implementing privacy-preserving statistics with secure multi-party computation. In this paper, we have made significant improvements. First, we provide all the algorithms with their privacy analysis. This includes new algorithms for privacy-preserving false discovery rate control, Mann-Whitney U test, Gaussian elimination with backsubstitution, LU decomposition and its use in solving a set of linear equations, and the conjugate gradient method.

Second, we present the Rmind privacy-preserving statistical tool that is designed to resolve user acceptance issues. This is achieved by implementing privacy-preserving tools needed for complex data analysis processes where data is collected from various sources, linked and statistically analysed. Third, we have implemented the new algorithms, optimised the previous prototype implementation, and provide new performance results that prove the feasibility of the system.

Related work. The closest system to what we propose has been introduced by Chida et al. in [17]. They are using the popular statistics environment R to

[1] The R Project for Statistical Computing. `http://www.r-project.org`

create a user interface to a secure multi-party computation backend. They have implemented descriptive statistics, filtering, cross-tabulation and a version of the t-test and χ^2 test. Their protocols combine public and private calculations and provide impressive performance. However, their implementation is limited in the kinds of analyses they can perform due to their lack of support for real numbers. Their implementation also does not support linking databases.

Another recent implementation with similar goals is by El Emam et al [22]. They provide protocols for only linking and the computation of χ^2-tests, odds ratio and relative risk. Other published results have focused on individual components in our statistics suite, e.g, mean, variance, frequency analysis and regression [14, 19, 20, 38, 37], filtered sums, weighted sums and scalar products [51, 54, 33].

Related papers on private data aggregation have targeted streaming data, for example consider solutions by Shi et al. and Li et al. that provide differential privacy [48, 43].

2 A tool for cryptographically secure statistical analysis

2.1 Preliminaries

Secure multi-party computation (SMC) is a cryptographic technology for computing a function with multiple parties so that only the party who provided a particular input can see that input and every party only gets the output specifically intended for them. More specifically, *input parties* provide the inputs to the computation and expect that nobody else learns them. *Computing parties* store inputs, participate in secure multi-party computation protocols and give outputs to the *output parties*.

There are various techniques for performing SMC, including homomorphic encryption [45, 25], garbled circuits [55] and secret sharing [16, 16]. All of these can provide a set of cryptographic primitives that implement the secure arithmetic we need for the statistical operations. We model such sets of primitives as *protection domain kinds*, following the definition from [5].

Definition 1 (Protection domain kind). *A protection domain kind (PDK) is a set of data representations, algorithms and protocols for storing and computing on protected data.*

The statistical tool presented in this paper is designed on a protection domain kind based on *additive secret sharing* [8]. If an input party wants to provide a secret value $s \in Z$ (where Z is a quotient ring) as a private input to n computing parties, it first uniformly generates *shares* $s_1, \ldots, s_{n-1} \leftarrow Z$ and calculates the final share $s_n \leftarrow s - s_1 - \cdots - s_{n-1}$. Each computing party receives one share s_i. No computing party is capable of recovering s, because each share of it is just a uniformly distributed value.

The parties can use secure operations defined in the protection domain kind to process the shares without recovering the secret. For example, if each computing party has shares u_i and v_i of secrets u and v, they can calculate $w_i \leftarrow u_i + v_i$

to get the shares of $w = u + v$. Further operations in this protection domain kind require more complex protocols, as described in [8, 9].

For statistical analysis, we need a PDK with composable operations for secure integer arithmetic, secure floating point arithmetic, secure linking, sorting and shuffling. The arithmetic operations are required for implementing statistical functions with cryptographic privacy. Secure linking and sorting are required for preparing data tables for analysis.

In this paper, we also require that the PDK has a cryptographically private shuffling protocol. This protocol must be capable of randomly rearranging the values in a vector or rows in a matrix without leaking the elements in the vector or matrix. By shuffling, we can rearrange inputs and, thus, break any relations between the order of the values and their sources. Thus, it is an important tool for reducing privacy leaks.

Many PDK implementations provide integer arithmetic [44, 13, 29, 24, 18, 3, 39], floating point arithmetic [15, 23], shuffling and sorting [26] and linking [22], but, to our knowledge, only the SHAREMIND framework provides all the operations we need in a single implementation [9, 42, 40, 35, 6].

Therefore, the statistical tool in this paper is built on the SHAREMIND framework and its protection domain kind implementation based on additive secret sharing among three parties with honest-but-curious security. This security model is acceptable for us in the statistical application, as we are mainly interested in providing privacy. However, the algorithms presented in this paper are not dependent on the particular protection domain kind and can be implemented on other frameworks as well.

2.2 The design of a secure statistical tool

In this paper, we propose the complete design for a statistical analysis tool that collects and analyses data in an encrypted form without decrypting it. The tool is designed to achieve the following goals:

1. **Support for data collection and sharing.** Based on end-user interviews in [4], the scenarios that most need privacy-preserving data analysis are ones where different data owners provide data to be statistically analysed by one or more analysts. The tool must support such an analysis process.
2. **Similarity to existing tools.** Based on [4], the end users prefer tools with familiar user interfaces.
3. **Clear tagging of public and private data.** The data uploaded into the tool may contain both public and private values. The tool must support data tagging with privacy types.
4. **Seamless use of cryptographic technology.** The interfaces for analysing public and private data must be the same and cryptography is applied automatically and seamlessly when private data is processed.
5. **Features chosen according to real-world needs.** The statisticians we interviewed stressed that data transformation and preparation is as important as the final analysis and a good tool supports both.

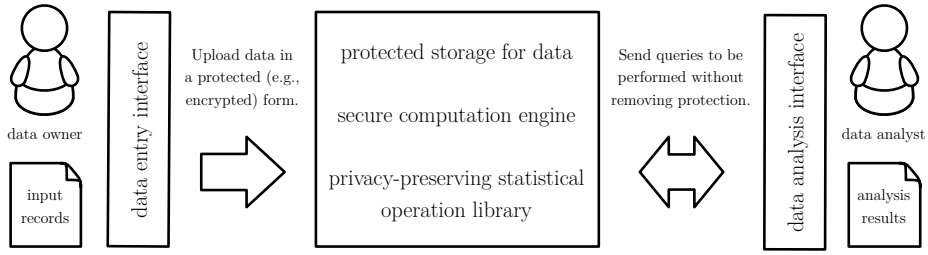

Fig. 1. Design of a privacy-preserving statistical analysis tool and interfaces based on secure computation

Based on these goals, we propose the design described in Figure 1. The main components of the tool are the data entry interface, the data analysis interface and the storage and computation backend. Standalone data entry or analysis interfaces can be implemented on any programming platform supported by the secure multi-party computation framework. Data is tagged public or private during the upload to allow the statistical tool to apply secure multi-party computation on private data.

What differentiates this design from other client-server designs, is that private data remains protected throughout the analysis process with secure multi-party computation techniques. This efficiently ensures that the data owner is the only party with access to the private inputs and only the results of the analysis are disclosed to the analyst. We will now elaborate further on the exact privacy guarantees of the tool.

2.3 Privacy goals

In our setting, data owners who use data upload tools, fill the roles of input parties. The people or organisations hosting the privacy-preserving storage and analysis systems are the computing parties. The statistical analyst who wants to see the results of the analysis is the result party. A single real-world entity can fill the role of multiple parties, e.g. both provide and receive information.

Our privacy goal in this setting is to ensure that the private inputs of the input parties cannot be accessed by other parties. Ideally, we would like to assure that no information about the private inputs is revealed during the computations. However, such a security goal is impossible to achieve, as the expected outputs contain some information about the data. On the other hand, simple cryptographic privacy is not enough, as it does not ensure that the outputs do not directly leak a significant amount of information about private inputs. Hence, we have the following four privacy properties that the Rmind tool, and the secure multi-party computation algorithms that make up its analytical capability, must satisfy.

1. **Source privacy.** During the evaluation of the algorithm, computing parties must not be capable of associating a particular computation result with

the input of a certain input party. We achieve this goal in the proposed algorithms by using the secure shuffling operation that removes such associations.

2. **Cryptographic privacy.** During the evaluation of the algorithm, computing parties must not learn anything about the intermediate values used to compute results, including the individual values in the inputs of input parties, unless this information can be deduced form the desired output of the algorithm. This goal is achieved by processing all private inputs using secure operations in the PDK.
3. **Query auditing.** During the evaluation of the algorithm, the result parties learn the intended results only when no computing party objects to the publication of the results. In our tool, query restrictions are enforced by the computing parties who can refuse to respond to a query, if they have reason to believe that this query would break privacy. Restrictions on queries can be based on the query auditing algorithms that attempt to detect malicious behaviour from statisticians performing the analysis. For example, a computing party can refuse to give the results to an aggregation query, when it is performed over less than five inputs. Note that, at this point, we are focusing on privacy and are less worried of denial of service attacks by computing parties.
4. **Output privacy.** The outputs of the algorithm must not leak significant parts of the private inputs. In this paper, we consider output privacy separately for each algorithm. In some algorithm we allow limited leakage from certain operations for efficiency reasons. For example, we may reveal the number of values that remain after filtering. Each algorithm description is followed by a description of the allowed leakage.
   We adopt this approach, because the proper formalisation of the output privacy is elusive for statistical analysis. On one hand, k-anonymity [52] is too weak as it does not consider potential background information an attacker can have. On the other hand, differential privacy [21] is too strong, as it assumes perfect background knowledge which is unrealistic. In this paper, we are focusing on algorithm designs that do not add noise to the inputs or outputs.

All privacy-preserving algorithms proposed in this paper are designed to follow these goals. As we allow both public and private values in our algorithms, we use the following notation in the algorithms to distinguish between them. Let $[\![x]\!]$ denote a private value x, let $[\![\boldsymbol{a}]\!]$ denote a private value vector $\boldsymbol{a}$, and let $[\![\mathbf{M}]\!]$ denote a private matrix $\mathbf{M}$. Let binary operations between private values denote the respective privacy-preserving protocols in the used protection domain kind. We consider the sizes of all vectors and matrices to be public.

2.4 Achieving efficiency

Protection domain kinds may have specific performance profiles. For example, PDKs based on secret sharing, are significantly more efficient when the algo-

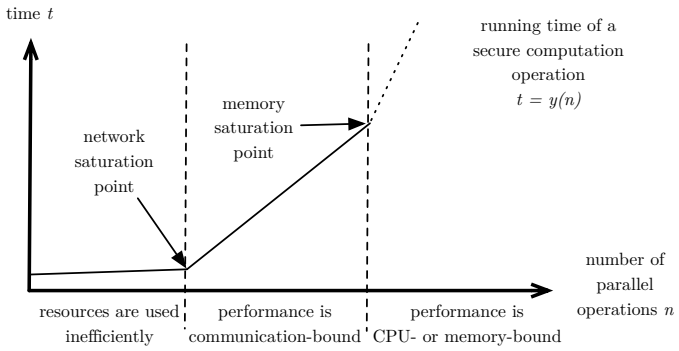

Fig. 2. Performance model of the secure computation protocols based on secret sharing.

rithms perform many parallel operations together. Figure 2 shows a generic running time profile for a secure multi-party computation operation [9].

The vertical axis shows the running time t of a secure multi-party computation protocol based on secret sharing and the horizontal axis shows n, the number of simultaneous parallel operations. In the function $t = y(n)$, y characterises the running time of the protocol based on the network setting it is deployed in. A thorough study of function y has been performed in [47].

On Figure 2, we can distinguish three different stages. In the first stage, the running time does not grow quickly in the number of inputs. This is because we can fit the protocols for many parallel operations in the network channel at the same time. Once the network channel becomes saturated, each further input starts increasing the number of round trips for messages in the protocol. This causes the running time to grow much faster with each parallel input. At some point, all the computers resources are used up and processing more values in parallel will either be impossible or very slow. Similar effects have been noted in other PDKs, see e.g. [49]. This gives us reason to use parallel operations as much as possible in our algorithm design.

3 Data import and filtering

We now present a collection of algorithms for performing privacy-preserving statistical analysis in the application model described in the previous sections. We begin with the first steps in the statistical analysis process—acquiring and preparing the data.

Data are crucial ingredients of statistical analyses and they can be collected for a specific designed study, such as a clinical trial, or, alternatively, existing datasets can be used, e.g. tax data can be used to analyse the financial situation of a country. We look at these two different methods separately, as they entail special requirements in a privacy-preserving setting.

First, let us look at the case where data are collected for a specific study. In a privacy-preserving setting we can assume that data are entered by a data

collector (e.g. national census or a clinical trial) or by data donors themselves (e.g. an online survey). In this case, the joint database is considered to be *vertically partitioned*. With secure multi-party computation, the data are encrypted or secret-shared immediately at the source and stored in a privacy-preserving manner.

Second, consider the case where datasets previously exist and analysts wish to perform a study by combining data from several different databases that cannot be joined publicly (forming a *horizontally partitioned* database). Then, data can be imported from these databases by encrypting or secret-sharing them and later merging them in a privacy-preserving way.

For generality, we look at the data importing stage as one abstract operation. However, we keep in mind that when dealing with existing databases, data can be validated and filtered by the database owners and managers before they are imported into the privacy-preserving database. In addition to automatic checks, the data manager can also look at the data and see if there are questionable values that need to be checked or removed.

3.1 Availability mask vectors

It is likely that in a dataset, values are missing for some data donors. There are two options for dealing with missing values in a privacy-preserving dataset: a special value in the data domain can be used for denoting these values; or an extra attribute can be added for each attribute to store this information. The first option is not practical in the encrypted domain, as the use of a special value adds an expensive private comparison operation to nearly every operation.

The solution with an extra attribute is much more practical, as only one private bit of extra data needs to be stored per entry. The latter uses extra storage space of $N \cdot k \cdot b$ bits, where N is the number of entries, k is the number of attributes, and b is the smallest data unit that can be stored in the database.

In our work, we concentrate on the latter, as this allows us to perform faster computations on private data. Let the availability mask $[\![\boldsymbol{m}]\!]$ of vector $[\![\boldsymbol{a}]\!]$ contain 0 if the corresponding value in the attribute $[\![\boldsymbol{a}]\!]$ is missing and 1 otherwise. It is clear, that it is not possible to discern which and how many values are missing from the value vector by looking at the private availability vector. However, the count of available elements can be computed by summing the values in the availability mask.

3.2 Input validation

By input validation and data correction, we mean operations that can be done without interacting with the user, such as range and type checks. The values for acceptable ranges and types are specified by the user but they are applied to the data automatically. Input validation and data correction can be performed at two stages—during data import on entered but not yet encrypted data, or afterwards, in the privacy-preserving database on encrypted data. It is, of course, faster, more sensible and straightforward to do this on data before encryption.

Algorithm 1: Privacy-preserving function **cut** for cutting the dataset according to a given filter.

Data: Data vector $[\![\boldsymbol{a}]\!]$ of size N and corresponding mask vector $[\![\boldsymbol{m}]\!]$.
Result: Data vector $[\![\boldsymbol{x}]\!]$ of size n that contains only elements of $[\![\boldsymbol{a}]\!]$ corresponding to the mask $[\![\boldsymbol{m}]\!]$

1 Shuffle the value pairs in vectors $([\![\boldsymbol{a}]\!], [\![\boldsymbol{m}]\!])$ into $([\![\boldsymbol{a'}]\!], [\![\boldsymbol{m'}]\!])$
2 $\boldsymbol{s} \leftarrow$ **declassify**$([\![\boldsymbol{m'}]\!])$
3 $[\![\boldsymbol{x}]\!] \leftarrow ([\![a'_i]\!] \mid s_i = 1, i \in \{1, \ldots, N\})$
4 **return** $[\![\boldsymbol{x}]\!]$

If the data cannot be checked before encryption for some reason, the validation has to be performed in the privacy-preserving database.

To perform a range check on a private vector $[\![\boldsymbol{a}]\!]$ of values, we construct a vector $[\![\boldsymbol{b}]\!]$ containing the constant with which we want to compare the values. We then compare the two vectors point-wise and get a private vector $[\![\boldsymbol{m}]\!]$ of comparison results, where $[\![m_i]\!] \leftarrow [\![a_i]\!] \odot [\![b_i]\!]$, where $\odot$ is a comparison operation, $i \in \{1, \ldots, n\}$, and n is the size of vectors $[\![\boldsymbol{a}]\!]$, $[\![\boldsymbol{b}]\!]$ and $[\![\boldsymbol{m}]\!]$. Unlike the public range check operation, the privacy-preserving version does not receive as a result a private vector of values that are within range. Instead, it receives a mask vector $[\![\boldsymbol{m}]\!]$ that can be used in further computations. To find out how many values are within range, it suffices to sum the values in vector $[\![\boldsymbol{m}]\!]$.

3.3 Evaluating filters and isolating filtered data

Similarly to range checks, filtering can be performed by comparing a vector $[\![\boldsymbol{a}]\!]$ of values point-wise to a vector $[\![\boldsymbol{b}]\!]$ containing filter values. As a result, we obtain a mask vector $[\![\boldsymbol{m}]\!]$ that contains 1 if the condition holds and 0 otherwise. For a more complex filter, the comparisons are done separately and the resulting mask vectors are combined using conjunction and disjunction. For example, to find from a dataset all women who have had a degree in statistical analysis for more than 5 years, we will first make three comparisons $[\![\boldsymbol{p}]\!] \leftarrow ([\![\boldsymbol{a}]\!] = \text{"}F\text{"})$, $[\![\boldsymbol{q}]\!] \leftarrow ([\![\boldsymbol{b}]\!] = \text{"}\textit{statistics}\text{"})$, and $[\![\boldsymbol{r}]\!] \leftarrow ([\![\boldsymbol{c}]\!] > 5)$, where $[\![\boldsymbol{a}]\!]$ contains values for gender, $[\![\boldsymbol{b}]\!]$ contains values for a person's specialty, and $[\![\boldsymbol{c}]\!]$ contains values for years since graduation. As the filters themselves are privacy-preserving, it is not possible to distinguish which records correspond to the filter conditions. To get the combined filter, we need to conjunct the filters together $[\![\boldsymbol{m}]\!] \leftarrow [\![\boldsymbol{p}]\!] \wedge [\![\boldsymbol{q}]\!] \wedge [\![\boldsymbol{r}]\!]$.

Most of the algorithms presented in this paper are designed so that filter information is taken into account during computations. Thus, in general, there is no need to extract from the original data vector a subset of values that correspond to the filter. On the other hand, some algorithms require that a subset vector containing only the filtered data be built. We use Algorithm 1 for cutting the dataset based on a given filter in a privacy-preserving way.

First the value and mask vector pairs are shuffled in a privacy-preserving way, retaining the correspondence of the elements. Next, the mask vector is

declassified and values for which the mask vector contains 0 are removed from the value vector. The obtained subset vector is then returned to the user. It is also possible to cut matrices in a similar fashion. This process leaks the number of values that correspond to the filter that the mask vector represents. If the number of records in the filter is published anyway, the function **cut** does not reveal any new information, as oblivious shuffling ensures that no other information about the private input vector and mask vector is leaked [42].

The availability of filtering, predicate evaluation and summing elements of a vector, allows us to implement privacy-preserving versions of data mining algorithms such as frequent itemset mining (FIM). As FIM is often used for mining sparse datasets, a significant speedup can be achieved if the infrequent itemsets are pruned using the **cut** function.

4 Data quality assurance

When databases are encrypted or secret-shared, the privacy of the data donor is protected and no users, including system administrators can see individual values. However, this also makes it impossible for the data analyst to see the data. Statistical analysts often detect patterns and anomalies, and formulate hypotheses when looking at the data values. Our solution is to provide privacy-preserving algorithms for a range of descriptive statistics that can give a feel of the data, while protecting the individual records.

Given access to these aggregate values and the possibility to eliminate outliers, it is possible to ensure data quality without compromising the privacy of individual data owners. Even though descriptive statistics leak some information about inputs, the leakage is small and strictly limited to aggregations permitted on the current database.

4.1 Five-number summary

Box-plots are a simple tool for giving a visual overview of the data and for effectively drawing attention to outliers. These diagrams are based on the five-number summary—a set of descriptive statistics that includes the minimum, lower quartile, median, upper quartile and maximum of an attribute. All of these values are, in essence, quantiles and can, therefore, be computed by using a formula for the appropriate quantiles. As no one method for quantile estimation has been widely agreed upon in the statistics community, we use algorithm $\mathbf{Q_7}$ from [32] used by the R software. Let p be the percentile we want to find and let $[\![\boldsymbol{a}]\!]$ be a vector of values sorted in ascending order. Then the quantile is computed using the following function:

$$\mathbf{Q}(p, [\![\boldsymbol{a}]\!]) = (1-\gamma) \cdot [\![a_j]\!] + \gamma \cdot [\![a_{j+1}]\!] \ ,$$

where $j = \lfloor (n-1)p \rfloor + 1$, n is the size of vector $[\![\boldsymbol{a}]\!]$, and $\gamma = np - \lfloor (n-1)p \rfloor - p$. For finding the j-th element in a private vector of values, we can either use privacy-preserving versions of vector lookup or sorting.

Algorithm 2: Privacy-preserving algorithm for finding the five-number summary of a vector.

Data: Input data vector $[\![\boldsymbol{a}]\!]$ and corresponding mask vector $[\![\boldsymbol{m}]\!]$.
Result: Minimum $[\![min]\!]$, lower quartile $[\![lq]\!]$, median $[\![me]\!]$, upper quartile $[\![uq]\!]$, and maximum $[\![max]\!]$ of $[\![\boldsymbol{a}]\!]$ based on the mask vector $[\![\boldsymbol{m}]\!]$

1 $[\![\boldsymbol{x}]\!] \leftarrow \mathbf{cut}([\![\boldsymbol{a}]\!], [\![\boldsymbol{m}]\!])$
2 $[\![\boldsymbol{b}]\!] \leftarrow \mathbf{sort}([\![\boldsymbol{x}]\!])$
3 $[\![min]\!] \leftarrow [\![b_1]\!]$
4 $[\![max]\!] \leftarrow [\![b_n]\!]$
5 $[\![lq]\!] \leftarrow \mathbf{Q}(0.25, [\![\boldsymbol{b}]\!])$
6 $[\![me]\!] \leftarrow \mathbf{Q}(0.5, [\![\boldsymbol{b}]\!])$
7 $[\![uq]\!] \leftarrow \mathbf{Q}(0.75, [\![\boldsymbol{b}]\!])$
8 **return** $([\![min]\!], [\![lq]\!], [\![me]\!], [\![uq]\!], [\![max]\!])$

Algorithm 3: Privacy-preserving algorithm for finding the five-number summary of a vector.

Data: Input data vector $[\![\boldsymbol{a}]\!]$ of size N and corresponding mask vector $[\![\boldsymbol{m}]\!]$.
Result: Minimum $[\![min]\!]$, lower quartile $[\![lq]\!]$, median $[\![me]\!]$, upper quartile $[\![uq]\!]$, and maximum $[\![max]\!]$ of $[\![\boldsymbol{a}]\!]$ based on the mask vector $[\![\boldsymbol{m}]\!]$

1 $([\![\boldsymbol{b}]\!], [\![\boldsymbol{m'}]\!]) \leftarrow \mathbf{sort}([\![\boldsymbol{a}]\!], [\![\boldsymbol{m}]\!])$
2 $[\![n]\!] \leftarrow \mathbf{sum}([\![\boldsymbol{m}]\!])$
3 $[\![os]\!] \leftarrow N - [\![n]\!]$
4 $[\![min]\!] \leftarrow [\![b_{[\![1+os]\!]}]\!]$
5 $[\![max]\!] \leftarrow [\![b_N]\!]$
6 $[\![lq]\!] \leftarrow \mathbf{Q}(0.25, [\![\boldsymbol{a}]\!], [\![os]\!])$
7 $[\![me]\!] \leftarrow \mathbf{Q}(0.5, [\![\boldsymbol{a}]\!], [\![os]\!])$
8 $[\![uq]\!] \leftarrow \mathbf{Q}(0.75, [\![\boldsymbol{a}]\!], [\![os]\!])$
9 **return** $([\![min]\!], [\![lq]\!], [\![me]\!], [\![uq]\!], [\![max]\!])$

Using the quantile computation formula $\mathbf{Q}$, the five-number summary can be computed as given in Algorithm 2. The algorithm uses Algorithm 1 to remove the unwanted or missing values from the data vector. The subset vector is sorted in a privacy-preserving way and then the summary elements are computed. As mentioned before, function **cut** leaks the count of elements n that correspond to the filter signified by the mask vector $[\![\boldsymbol{m}]\!]$. If we want to keep n secret, we can use Algorithm 3 which hides n, but is slower than Algorithm 2.

Algorithm 3 starts by sorting the data vector first by the data values and then by the mask values (line 1) to ensure that missing values will be at the beginning of the vector. Next, the offset is computed privately on line 3. The formula $\mathbf{Q}$ for computing the quantiles works as before with the exception that, as the number of elements $[\![n]\!]$ is kept private, the indices $[\![j]\!]$ are also not revealed. The computed offset $[\![os]\!]$ is added to the private index $[\![j]\!]$ to account for the values at the beginning of the sorted vector that do not belong to the filtered subset. In addition, vector lookup on line 4 and during the computations of the

Algorithm 4: Privacy-preserving algorithm for finding the frequency table of a data vector.

Data: Input data vector $[\![\boldsymbol{a}]\!]$ and corresponding mask vector $[\![\boldsymbol{m}]\!]$.
Result: Vector $[\![\boldsymbol{b}]\!]$ containing breaks against which frequency is computed, and vector $[\![\boldsymbol{c}]\!]$ containing counts of elements

1 $[\![\boldsymbol{x}]\!] \leftarrow \mathbf{cut}([\![\boldsymbol{a}]\!], [\![\boldsymbol{m}]\!])$
2 $n \leftarrow \mathbf{declassify}(\mathbf{sum}([\![\boldsymbol{m}]\!]))$
3 $k \leftarrow \lceil \log_2(n) + 1 \rceil$
4 $[\![min]\!] \leftarrow \mathbf{min}([\![\boldsymbol{x}]\!])$, $[\![max]\!] \leftarrow \mathbf{max}([\![\boldsymbol{x}]\!])$
5 Compute breaks according to $[\![min]\!]$, $[\![max]\!]$ and k, assign result to $[\![\boldsymbol{b}]\!]$
6 $[\![c_i]\!] \leftarrow (\mathbf{sum}([\![x_i]\!] \leq [\![b_{i+1}]\!]), i \in \{1, \ldots, n\})$
7 **for** $i \in \{k, \ldots, 2\}$ **do**
8 $\quad [\![c_i]\!] \leftarrow [\![c_i]\!] - [\![c_{i-1}]\!]$
9 **end**
10 **return** $([\![\boldsymbol{b}]\!], [\![\boldsymbol{c}]\!])$

quantiles are now performed in a privacy-preserving way. As a result, the value is retrieved while the the index remains secret for all computing parties

4.2 Data distribution

Distribution of an attribute also gives an insight into the data. For categorical attributes, the distribution can be discerned by counting the occurrences of different values. For numerical attributes, we must split the range into bins specified by breaks and compute the corresponding frequencies. The resulting frequency tables can be visualised as a histogram.

Algorithm 4 computes a frequency table for a vector of numerical values similarly to a public frequency computation algorithm. Missing or filtered values are removed on line 1. The number of bins k is computed according to Sturges' formula [50]. Bins are created based on the minimum, maximum and k. Finally, on lines 6 - 9, the elements belonging in each bin are counted using secure multiparty computation. These comparisons can be done in parallel to speed up the algorithm.

The process of creating a frequency table for discrete or categorical attributes is similar, but instead of checking whether an element belongs to an interval, it is compared to each bin value and the last cycle is omitted.

4.3 Simple statistical measures

Statistical algorithms use common operations for computing means, variance and covariance. These statistical measures also provide important insights about the attributes and their correspondence. We show, how these measures can be computed in a privacy-preserving manner over various samples.

To compute means, variances and covariances, we first multiply point-wise the value vector $[\![\boldsymbol{a}]\!]$ with the mask vector $[\![\boldsymbol{m}]\!]$. Let us denote the result by $[\![\boldsymbol{x}]\!]$.

This way, the values that do not correspond to the filter do not interfere with the computations. The number of subjects n is computed by summing the elements in the mask vector. The arithmetic mean, and the unbiased estimates of variance and standard deviation can be computed as follows

$$\begin{aligned}
\mathbf{mean}([\![\boldsymbol{x}]\!]) &= \frac{1}{n} \cdot \sum_{i=1}^{n} [\![x_i]\!] \ , \\
\mathbf{var}([\![\boldsymbol{x}]\!]) &= \frac{1}{n-1} \left(\sum_{i=1}^{n} [\![x_i]\!]^2 - \frac{1}{n} \cdot \left(\sum_{i=1}^{n} [\![x_i]\!] \right)^2 \right) \ , \\
\mathbf{sdev}([\![\boldsymbol{x}]\!]) &= \sqrt{\mathbf{var}([\![\boldsymbol{x}]\!])} \ .
\end{aligned}$$

The computation of these values is straightforward, if the privacy-preserving platform supports addition, multiplication, division and square root. Furthermore, if n is public, we can use division with a public divisor and public square root instead, as they are faster than the private versions.

Trimmed mean is a version of mean where the upper and lower parts of the sorted data vector $[\![\boldsymbol{a}]\!]$ are not included in the computation. The analyst specifies the percentage of data that he or she wants to trim off the data. Then the corresponding quantiles are computed, data is compared to these values and the mask vector $[\![\boldsymbol{m}]\!]$ is updated with the results acquired from the comparison operation. This ensures that only values that fall between the given percentages remain in the filtered result $[\![\boldsymbol{x}]\!]$.

Covariance shows whether two attributes change together. The unbiased estimate of covariance between filtered vectors $[\![\boldsymbol{x}]\!]$ and $[\![\boldsymbol{y}]\!]$ can be computed as

$$\mathbf{cov}([\![\boldsymbol{x}]\!], [\![\boldsymbol{y}]\!]) = \frac{1}{n-1} \left(\sum_{i=1}^{n} [\![x_i]\!][\![y_i]\!] - \frac{1}{n} \cdot \sum_{i=1}^{n} [\![x_i]\!] \sum_{i=1}^{n} [\![y_i]\!] \right) \ .$$

4.4 Unidimensional outlier detection

Datasets often contain errors or extreme values that should be excluded from the analysis. Although there are many elaborate outlier detection algorithms like [12], outliers are often detected using quantiles.

It is common to mark values in a data vector $[\![\boldsymbol{a}]\!]$ smaller than the 5% quantile or larger than 95% quantile as outliers. The corresponding mask vector is computed by comparing all elements of $[\![\boldsymbol{a}]\!]$ to $\mathbf{Q}(0.05, [\![\boldsymbol{a}]\!])$ and $\mathbf{Q}(0.95, [\![\boldsymbol{a}]\!])$, and then conjuncting the resulting index vectors. The values of the quantiles need not be published for outlier detection purposes and data is filtered to exclude outliers from further analysis. Furthermore, it is possible to combine the mask vector with the availability mask $[\![\boldsymbol{m}]\!]$ and cache it as an updated availability mask to reduce the filtering load.

Another generic measure of eliminating outliers from a dataset is using median absolute deviation [27, 28] (MAD). Element $[\![x]\!]$ is considered an outlier in a value vector $[\![\boldsymbol{a}]\!]$ of length n if

$$|\mathbf{Q}(0.5, [\![\boldsymbol{a}]\!]) - [\![x]\!]| > \lambda \cdot \mathsf{MAD}, \tag{1}$$

where

$$\mathsf{MAD} = \mathbf{Q}(0.5, |[\![a_i]\!] - \mathbf{Q}(0.5, [\![\boldsymbol{a}]\!])|) \ \ , i \in \{1, \ldots, n\}$$

and λ is a constant. The exact value of λ is generally between 3 to 5, but it can be specified depending on the dataset.

These simple statistical measures are also useful for analyzing simple privacy-preserving statistical surveys. For example, answers to single choice questions can be analyzed with privacy-preserving frequency tables and visualized with bar charts.

5 Statistical tests

5.1 The principles of statistical testing

Two-sample statistical tests compare two different populations. In such cases, we first extract two groups—the case and control populations. There are two ways to approach this.

Firstly, we can select the appropriate subjects into one group and assume all the rest are in the other group. Alternatively, we can choose subjects into both groups. These selection categories yield either one or two mask vectors. In the former case, we compute the second mask vector by flipping all the bits in the existing mask vector. Hence, we can always consider the version where case and control groups are determined by two mask vectors.

In the following, let $[\![\boldsymbol{a}]\!]$ be the value vector we are testing and let $[\![\boldsymbol{ca}]\!]$ and $[\![\boldsymbol{co}]\!]$ be mask vectors for case and control groups, respectively. Then $[\![n_{ca}]\!] = \mathbf{sum}([\![\boldsymbol{ca}]\!])$ and $[\![n_{co}]\!] = \mathbf{sum}([\![\boldsymbol{co}]\!])$ are the counts of subjects in the corresponding populations.

Figure 3 gives an overview of what the different steps are for statistical testing in the private and public setting. In a normal statistical testing procedure, we first compute the test statistic based on the data. Next, we compute the p-value based on the obtained value and the size of the sample. Finally, we compare the p-value to a significance threshold set by the analyst.

In the privacy-preserving setting, we have two options of how to carry out this procedure. The choice depends on how much data we are willing to publish. The first option is similar to the public setting, with the difference that the test statistic is computed in a privacy-preserving manner. It is then published along with the sample sizes, the p-value is computed publicly and compared to the given threshold value. Computing the p-value in public reveals sample sizes but does not reveal any additional information as the function is invertible if the sample sizes are known and the test statistic can be computed based on the

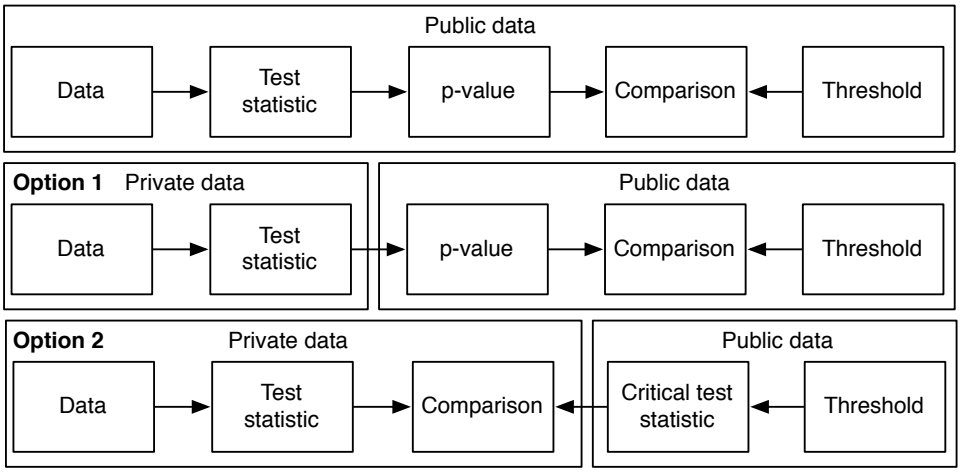

Fig. 3. Statistical testing procedure in the public and private setting

published p-value. However, revealing the sizes of the case and control groups might sometimes not be acceptable.

The first option is intuitive to the analyst as he or she receives the same type of result as in the case of public computations. However, in case of, for instance, genome-wide association studies (GWAS) and multiple testing, revealing the p-value can reveal too much information. For this occasion, we propose another way of doing privacy-preserving hypothesis testing.

In this option, the test statistic is computed based on the data in a privacy-preserving manner. The data analyst determines the threshold and the critical p-value and the corresponding test statistic are publicly determined based on this threshold. Finally, the privately computed test statistic is privately compared to the critical test statistic. The only thing that is published is the decision whether the alternative hypothesis is supported by the data. Note, that the first option must always be chosen if the analyst needs the p-value itself and not the accept/reject decision.

We discuss how to perform Student's t-test, paired t-test, Wilcoxon rank sum and signed-rank tests, and the χ^2-test in a privacy-preserving manner. These test algorithms return the test statistic value that has to be combined with the sizes of the compared populations to determine the significance of the difference.

5.2 Student's t-tests

The two-sample Student's t-test is the simplest statistical tests that allows us to determine whether the difference of group means is significant or not compared to variability in groups. There are two common flavours of this test [36] depending on whether the variability of the populations is equal. Let $[\![\boldsymbol{x}]\!] \leftarrow [\![\boldsymbol{a}]\!] \cdot [\![\boldsymbol{ca}]\!]$ and $[\![\boldsymbol{y}]\!] \leftarrow [\![\boldsymbol{a}]\!] \cdot [\![\boldsymbol{co}]\!]$, then $[\![\boldsymbol{x}]\!]$ is the data of the case population and $[\![\boldsymbol{y}]\!]$ is the data

of the control population. For equal variance, the t-test statistic is computed as:

$$[\![t]\!] = \frac{\mathbf{mean}([\![\boldsymbol{x}]\!]) - \mathbf{mean}([\![\boldsymbol{y}]\!])}{\mathbf{sdev}([\![\boldsymbol{x}]\!], [\![\boldsymbol{y}]\!]) \cdot \sqrt{\frac{1}{[\![n_{ca}]\!]} + \frac{1}{[\![n_{co}]\!]}}} \ ,$$

where $\mathbf{sdev}([\![\boldsymbol{x}]\!], [\![\boldsymbol{y}]\!])$ estimates the common standard deviation of the two samples and is computed as follows

$$\mathbf{sdev}([\![\boldsymbol{x}]\!], [\![\boldsymbol{y}]\!]) = \sqrt{\frac{([\![n_{ca}]\!] - 1) \cdot \mathbf{var}([\![\boldsymbol{x}]\!]) + ([\![n_{co}]\!] - 1) \cdot \mathbf{var}([\![\boldsymbol{y}]\!])}{[\![n_{ca}]\!] + [\![n_{co}]\!] - 2}} \ .$$

The t-test for unequal variances is also known as the Welch t-test. The test statistic is computed as follows

$$[\![t]\!] = \frac{\mathbf{mean}([\![\boldsymbol{x}]\!]) - \mathbf{mean}([\![\boldsymbol{y}]\!])}{\sqrt{\frac{\mathbf{var}([\![\boldsymbol{x}]\!])}{[\![n_{ca}]\!]} + \frac{\mathbf{var}([\![\boldsymbol{y}]\!])}{[\![n_{co}]\!]}}} \ .$$

A paired t-test [36] is used to detect whether a significant change has taken place in cases where there is a direct one-to-one dependence between case and control group elements, for example, the data consists of measurements from the same subject. Let $[\![\boldsymbol{b}]\!]$ and $[\![\boldsymbol{c}]\!]$ be the paired measurements, and let n be the count of these measurements. The test statistic for the paired t-test is computed in the following way

$$[\![t]\!] = \frac{(\mathbf{mean}([\![\boldsymbol{b}]\!]) - \mathbf{mean}([\![\boldsymbol{c}]\!])) \cdot \sqrt{[\![n]\!]}}{\mathbf{sdev}([\![\boldsymbol{b}]\!] - [\![\boldsymbol{c}]\!])} \ .$$

The algorithms for computing both t-tests are straightforward evaluations of the respective formulae using privacy-preserving computations. To compute these, we need the availability of privacy-preserving addition, multiplication, division and square root. As mentioned earlier, we can either publish the test statistic and the population sizes or, based on a user-given threshold, publish only whether the hypothesis was significant or not.

5.3 Wilcoxon rank sum test and signed rank test

T-test results have meaning only if the distribution of values in case and control groups follows the normal distribution. If this assumption does not hold, non-parametric Wilcoxon tests provide an alternative. The Wilcoxon rank sum test and its improvement Mann-Whitney U test [31] work on the assumption that the distribution of data in one group significantly differs from that in the other. Algorithm 5 gives an overview of how we compute the test statistic $[\![w]\!]$ using the Mann-Whitney U test.

For this algorithm to work, we need to cut the database similarly to what was done for the five-number summary, keeping in mind that here we need the dataset to retain elements from both groups—cases and controls. On line 1, we

Algorithm 5: Privacy-preserving Mann-Whitney U test

Data: Value vector $[\![\boldsymbol{a}]\!]$ and corresponding mask vectors $[\![\boldsymbol{ca}]\!]$ and $[\![\boldsymbol{co}]\!]$
Result: Test statistic $[\![w]\!]$
1 $[\![\boldsymbol{m}]\!] \leftarrow [\![\boldsymbol{ca}]\!] \vee [\![\boldsymbol{co}]\!]$
2 $[\![n_{ca}]\!] \leftarrow \mathbf{sum}([\![\boldsymbol{ca}]\!])$ and $[\![n_{co}]\!] \leftarrow \mathbf{sum}([\![\boldsymbol{co}]\!])$
3 $([\![\boldsymbol{x}]\!], [\![\boldsymbol{u}]\!], [\![\boldsymbol{v}]\!]) \leftarrow \mathbf{cut}(([\![\boldsymbol{a}]\!], [\![\boldsymbol{ca}]\!], [\![\boldsymbol{co}]\!]), [\![\boldsymbol{m}]\!])$
4 $([\![\boldsymbol{x}]\!], [\![\boldsymbol{u}]\!], [\![\boldsymbol{v}]\!]) \leftarrow \mathbf{sort}([\![\boldsymbol{x}]\!], [\![\boldsymbol{u}]\!], [\![\boldsymbol{v}]\!])$
5 $[\![\boldsymbol{r}]\!] \leftarrow \mathbf{rank}([\![\boldsymbol{x}]\!])$
6 $[\![\boldsymbol{r}_{ca}]\!] \leftarrow [\![\boldsymbol{r}]\!] \cdot [\![\boldsymbol{u}]\!]$ and $[\![\boldsymbol{r}_{co}]\!] \leftarrow [\![\boldsymbol{r}]\!] \cdot [\![\boldsymbol{v}]\!]$
7 $[\![R_{ca}]\!] \leftarrow \mathbf{sum}([\![\boldsymbol{r}_{ca}]\!])$ and $[\![R_{co}]\!] \leftarrow \mathbf{sum}([\![\boldsymbol{r}_{co}]\!])$
8 $[\![u_{ca}]\!] \leftarrow [\![R_{ca}]\!] - \frac{1}{2}([\![n_{ca}]\!] \cdot ([\![n_{ca}]\!] + 1))$ and $[\![u_{co}]\!] \leftarrow [\![n_{ca}]\!] \cdot [\![n_{co}]\!] - [\![u_{ca}]\!]$
9 **return** $[\![w]\!] \leftarrow \mathbf{min}([\![u_{ca}]\!], [\![u_{co}]\!])$

Algorithm 6: Privacy-preserving Wilcoxon signed-rank test

Data: Paired value vectors $[\![\boldsymbol{a}]\!]$ and $[\![\boldsymbol{b}]\!]$ for n subjects, mask vector $[\![\boldsymbol{m}]\!]$
Result: Test statistic $[\![w]\!]$
1 $([\![\boldsymbol{x}]\!], [\![\boldsymbol{y}]\!]) \leftarrow \mathbf{cut}(([\![\boldsymbol{a}]\!], [\![\boldsymbol{b}]\!]), [\![\boldsymbol{m}]\!])$
2 $[\![\boldsymbol{d}]\!] \leftarrow [\![\boldsymbol{x}]\!] - [\![\boldsymbol{y}]\!]$
3 Let $[\![\boldsymbol{d'}]\!]$ be the absolute values and $[\![\boldsymbol{s}]\!]$ be the signs of elements of $[\![\boldsymbol{d}]\!]$
4 $[\![\boldsymbol{s}]\!] \leftarrow \mathbf{sort}(([\![\boldsymbol{d'}]\!], [\![\boldsymbol{s}]\!]))$
5 $[\![\boldsymbol{r}]\!] \leftarrow \mathbf{rank_0}([\![\boldsymbol{s}]\!])$
6 **return** $[\![w]\!] \leftarrow \mathbf{sum}([\![\boldsymbol{s}]\!] \cdot [\![\boldsymbol{r}]\!])$

combine the two input mask vectors into one. The function **cut** differs from its previous usage in that several vectors are cut at once based on the combined filter $[\![\boldsymbol{m}]\!]$. Next, the value and mask vectors are sorted based on the values of $[\![\boldsymbol{x}]\!]$ so that the relation between the values and mask elements is retained.

The **rank** function on line 5 shares and assigns an integer $i \in \{1, \ldots, n\}$ to all values in the sorted vector based on the location of the value. If some values in the sorted vector are equal, all of these elements are assigned the average of their ranks. This is done using oblivious comparison and oblivious division. This correction makes the algorithm significantly slower as this requires us to keep all the ranks as floating point values instead of integers. We do allow our algorithms to be called without this correction which makes the test give a stricter bound and might not accept borderline hypotheses, but the algorithms will work faster.

On line 6, the rank vector $[\![\boldsymbol{r}]\!]$ is multiplied with the case and control masks to find the ranks belonging to the case and control groups.

Similarly to Student's paired t-test, the Wilcoxon signed-rank test [53] is a paired difference test. Our version, given in Algorithm 6, takes into account Pratt's correction [31] for when the values are equal and their difference is 0.

First, both data vectors are cut based on the mask vector similarly to what was done in Algorithm 5. Next, the difference $[\![\boldsymbol{d}]\!]$ between the two data samples is found, followed by the computation of the absolute value and sign of $[\![\boldsymbol{d}]\!]$. We expect that the latter is the standard function that returns -1 when the element

Algorithm 7: Privacy-preserving algorithm for compiling the contingency table for two classes with k options for the χ^2 test

Data: Value vector $[\![\boldsymbol{a}]\!]$, corresponding mask vectors $[\![\boldsymbol{ca}]\!]$ and $[\![\boldsymbol{co}]\!]$ for cases and controls respectively and a public code book **CB** defining k options
Result: Contingency table $[\![\mathbf{CT}]\!]$

1 $[\![x_{ca}]\!] \leftarrow [\![\boldsymbol{a}]\!] \cdot [\![\boldsymbol{ca}]\!]$ and $[\![x_{co}]\!] \leftarrow [\![\boldsymbol{a}]\!] \cdot [\![\boldsymbol{co}]\!]$
2 Let $[\![\mathbf{CT}]\!]$ be a $2 \times k$ matrix
3 **for** $i \in \{1, \ldots, k\}$ **do**
4 $\quad [\![b_{ca}]\!] \leftarrow [\![x_{ca}]\!] = \mathbf{CB}_{i,2}$ and $[\![b_{co}]\!] \leftarrow [\![x_{co}]\!] = \mathbf{CB}_{i,2}$
5 $\quad [\![\mathbf{CT}_{1,i}]\!] \leftarrow \mathbf{sum}([\![b_{ca}]\!])$ and $[\![\mathbf{CT}_{2,i}]\!] \leftarrow \mathbf{sum}([\![b_{co}]\!])$
6 **end**
7 **return** $[\![\mathbf{CT}]\!]$

is negative, 1 if it is positive and 0 otherwise. The signs are then sorted based on the absolute values $[\![\boldsymbol{d'}]\!]$ (line 4) and the ranking function $\mathbf{rank_0}$ is called. This ranking function is otherwise similar to the function **rank**, but differs in the fact that we also need to exclude the differences that have the value 0. Let the number of 0 values in vector $[\![\boldsymbol{d}]\!]$ be $[\![k]\!]$. As $[\![\boldsymbol{d}]\!]$ has been sorted based on absolute values, the 0 values are at the beginning of the vector so it is possible to use $[\![k]\!]$ as the offset for our ranks. Function $\mathbf{rank_0}$ assigns $[\![r_i]\!] \leftarrow 0$ while $[\![s_i]\!] = 0$, and works similarly to **rank** on the rest of the vector $[\![\boldsymbol{s}]\!]$, with the difference that $i \in \{1, \ldots, [\![n-k]\!]\}$.

Both algorithms only publish the statistic value and the population sizes.

5.4 The χ^2-tests for consistency.

If the attribute values are discrete such as income categories then it is impossible to apply t-tests or their non-parametric counterparts and we have to analyse frequencies of certain values in the dataset. The corresponding statistical test is known as the χ^2-test.

	Option 1	**Option 2**	...	**Total**
Cases	c_1	c_2	...	r_1
Controls	d_1	d_2	...	r_2
Total	p_1	p_2	...	n

Table 1. Contingency table for the standard χ^2 test

The standard χ^2-test statistic is computed as

$$\chi^2 \leftarrow \sum_{i=1}^{k} \sum_{j=1}^{2} \frac{(f_{ji} - e_{ji})^2}{e_{ji}} \ ,$$

Algorithm 8: Privacy-preserving χ^2 test

Data: Contingency table $[\![\mathbf{C}]\!]$ of size $2 \times k$
Result: The test statistic χ^2

1 Let $[\![n]\!]$ be the total count of elements
2 Let $[\![r_1]\!]$ and $[\![r_2]\!]$ be the row subtotals and $[\![p_1]\!], \ldots, [\![p_k]\!]$ be the column subtotals
3 Let $[\![\mathbf{E}]\!]$ be a table of expected frequencies such that $[\![\mathbf{E}_{i,j}]\!] \leftarrow \frac{[\![r_i]\!] \cdot [\![p_j]\!]}{n}$, $i \in \{1, 2\}, j \in \{1, \ldots, k\}$
4 $[\![\chi^2]\!] \leftarrow \sum_{j=1}^{k} \frac{([\![\mathbf{C}_{1,j}]\!] - [\![\mathbf{E}_{1,j}]\!])^2}{[\![\mathbf{E}_{1,j}]\!]} + \frac{([\![\mathbf{C}_{2,j}]\!] - [\![\mathbf{E}_{2,j}]\!])^2}{[\![\mathbf{E_{2,j}}]\!]}$
5 **return** $[\![\chi^2]\!]$

where f_{ji} is the observed frequency and e_{ji} is the expected frequency of the i-th option and j-th group. For simplification, we denote $c_i = f_{1i}$ and $d_i = f_{2i}$, then the frequencies can be presented as the contingency Table 1. Algorithm 7 compiles a contingency table from a data vector, mask vector and a code book **CB** that gives the algorithm information about which elements will be converted into which option.

Algorithm 8 shows how to compute the χ^2 test statistic based on a contingency table. The algorithm can be optimised if the number of classes is small, e.g. two. The algorithm publishes only the statistic value and the population sizes.

5.5 Multiple testing

When we have a dataset with several distinct variables ready for analysis, we can test multiple hypotheses on the gathered data. However, this can lead to false positive results as the chances of accidental correlation rise with each variable tested. When working with multiple testing, different precautions can be taken. In this section, we discuss how to apply privacy-preserving versions of Bonferroni correction and Benjamini-Hochberg procedur (false discovery rate control).

As the correction for multiple-hypothesis testing is trivial when p-values are public, we consider the case where privacy-preserving statistical testing reveals only whether the corrected significance threshold is reached or not.

Bonferroni correction Let α be the significance threshold and let k be the number of tests applied on the same data. Then the Bonferroni correction simply reassigns the same significance threshold $\hat{\alpha} = \alpha/k$ to all tests. As a result, the implement Bonferroni correction is trivial to implement, we just have to use the corrected significance threshold $\hat{\alpha}$ of all privacy-preserving statistical tests.

Benjamini-Hochberg procedure Bonferroni correction is often too harsh compared to false discovery rate (FDR) correction. Unfortunately, FDR is not as

Algorithm 9: Privacy-preserving Benjamini-Hochberg procedure

Data: Vector of k test statistics $[\![\boldsymbol{t}]\!]$, significance threshold α
Result: List of significant hypotheses
1 Publicly compute $q_{(i)} \approx Q\left(\frac{i\alpha}{k}\right)$ for $i \in \{1, \ldots, k\}$.
2 Obliviously sort pairs $([\![t_i]\!], [\![i]\!])$ in descending order wrt. $[\![t_i]\!]$.
3 Let $([\![t_{(i)}]\!], [\![c_{(i)}]\!])$ be the result.
4 Compute $[\![s_i]\!] \leftarrow ([\![t_{(i)}]\!] \geq q_{(i)})$ for $i \in \{1, \ldots, k\}$.
5 Declassify values $[\![s_k]\!], [\![s_{k-1}]\!], \ldots [\![s_1]\!]$ until the first 1 ($[\![s_{i_*}]\!] = 1$) is revealed.
6 Declassify locations $[\![c_{(1)}]\!], \ldots, [\![c_{(i_*)}]\!]$ and declare the corresponding hypotheses significant.

straightforward to apply in the privacy-preserving setting as for multiple testing p-values must remain private. Therefore, we look at the Benjamini-Hochberg (BH) procedure [2]. The BH procedure first orders p-values in ascending order and then finds the largest i such that

$$p_{(i)} \leq \frac{i}{k}\alpha$$

where $p_{(i)}$ is the i-th p-value and k is the number of hypotheses.

Recall that for any statistical test the correspondence between between p-values and test statistics is anti-monotone. Namely, for any significance threshold α we can find a value $Q(\alpha)$ such that for any value of the test statistic $t \geq Q(\alpha)$ the corresponding p-value is less than α. As a result, the Benjamini-Hochberg criterion can be expressed in terms of decreasingly ordered test statistics:

$$p_{(i)} \leq \frac{i}{k}\alpha \qquad \Longleftrightarrow \qquad t_{(i)} \geq Q\left(\frac{i\alpha}{k}\right) \ .$$

For most tests, the function Q is computed as the upper quantile of a distribution that depends only on the size of the size of case and control group. Hence, we can publicly compute fractional approximations of significance thresholds

$$q_{(i)} \approx Q\left(\frac{i\alpha}{k}\right)$$

and use secure computing to evaluate comparisons $t_{(i)} \geq q_{(i)}$.

Algorithm 9 depicts the corresponding privacy-preserving significance testing procedure, which reveals only the locations c_i and ordering of significant test statistics. It is even possible to hide the ordering if the shares $[\![c_{(1)}]\!], \ldots, [\![c_{(i_*)}]\!]$ are obliviously shuffled before opening such that $c_{(1)}, \ldots, c_{(i_*)}$ are opened in random order.

6 Predictive modelling

6.1 Matrix operations

We have implemented privacy-preserving versions of the following vector and matrix operations: computing the dot product, computing vector length, com-

puting the unit vector, and matrix multiplication. It is also possible to transpose a matrix and compute its determinant. In addition, we have implemented cross product for vectors of length 3, and eigenvector and eigenvalue computation for 2×2 symmetric matrices. The privacy-preserving versions of these algorithms are straightforward.

6.2 Linear regression

Linear regression can be used to predict values of variables based on other existing variables. It can also be used to find out if and how strongly certain variables influence other variables in a dataset.

Let us first look at simple linear regression. Using covariance, we can perform simple linear regression with a single explanatory variable. The aim of this analysis is to find $\hat{\alpha}$ and $\hat{\beta}$ that fit the approximation $y_i \approx \alpha + \beta x_i$ best for all data points $x_i, y_i,\ i \in \{1, \ldots, n\}$. The estimations of α and β can be computed in the following way

$$\hat{\beta} = \frac{\mathbf{cov}(\boldsymbol{x}, \boldsymbol{y})}{\mathbf{var}(\boldsymbol{x})} \text{ and } \hat{\alpha} = \mathbf{mean}(\boldsymbol{y}) - \hat{\beta} \cdot \mathbf{mean}(\boldsymbol{x}) \ .$$

As we have the capability to compute covariance and variance in the privacy-preserving setting, we can also estimate the values of $\hat{\alpha}$ and $\hat{\beta}$ in this setting.

Next, let us assume that we have k independent variable vectors of n elements, i.e $\mathbf{X} = (\mathbf{X}_{j,i})$, where $i \in \{0, \ldots, k\}$ and $j \in \{1, \ldots, n\}$. The vector $\mathbf{X}_{j,0} = (1)$ is an added variable for the constant term. Let $\boldsymbol{y} = (y_j)$ be the vector of dependent variables. We want to estimate the unknown coefficients $\boldsymbol{\beta} = (\beta_i)$ such that

$$y_j \approx \beta_k \mathbf{X}_{j,k} + \ldots + \beta_1 \mathbf{X}_{j,1} + \beta_0 \mathbf{X}_{j,0} \ .$$

This equation can be written as $\mathbf{X}\boldsymbol{\beta} \approx \boldsymbol{y}$. As this system of linear equations is overdetermined, we use the linear least squares method to solve the system. Now, we have to find β in the following system of linear equations:

$$\mathbf{X}^T \mathbf{X} \boldsymbol{\beta} = \mathbf{X}^T \boldsymbol{y} \ .$$

We will look at four different methods for solving such systems: for $k < 4$, we simply invert the matrix by computing determinants. For the general case, we give algorithms for the Gaussian elimination method [46], LU decomposition [46] and the conjugate gradient method [30]. Note, that when using these algorithms, we assume that the data matrix has already been multiplied with its transpose $[\![\mathbf{A}]\!] = [\![\mathbf{X}]\!]^T [\![\mathbf{X}]\!]$ and the dependent variable has been multiplied with the transpose of the data matrix as well ($[\![\boldsymbol{b}]\!] = [\![\mathbf{X}]\!]^T [\![\boldsymbol{y}]\!]$).

In the privacy-preserving setting, matrix inversion using determinants is straightforward, and using this method to solve a system of linear equations requires only the use of multiplication, addition and division, and, therefore, we will not discuss it in length. For the more general methods for solving systems of linear equations, we first give an algorithm that finds the first maximum element in a vector and also returns its location in the vector, used for finding

Algorithm 10: maxLoc: Finding the first maximum element and its location in a vector in a privacy-preserving setting

Data: A vector $[\![\boldsymbol{a}]\!]$ of length n
Result: The maximum element $[\![b]\!]$ and its location l in the vector

```
1 Let π(j) be a permutation of indices j ∈ {1, ..., n}
2 [[b]] ← [[a_π(1)]] and [[l]] ← π(1)
3 for i ∈ {π(2), ..., π(n)} do
4 |   [[c]] ← ([[a_π(i)]] > [[b]])
5 |   [[b]] ← [[b]] − [[c]] · [[b]] + [[c]] · [[a_π(i)]]
6 |   [[l]] ← [[l]] − [[c]] · [[l]] + [[c]] · π(i)
7 end
8 return ([[b]], declassify([[j]]))
```

the pivot element. While the Gaussian and LU decomposition algorithms can be used without pivoting, it is not advisable as the algorithm is numerically unstable in the presence of any roundoff errors [46]. Algorithm 10 describes the function **maxLoc** that finds the pivot element and its location from a given vector. To avoid leaking information about equal values in a vector, the indices are first permuted to ensure cryptographic privacy. Hence, the algorithm leaks the location of a maximum element, but in the case of several equal maximum elements, it returns the location of one of these elements. In addition, the rows of both algorithms that call this function have been shuffled, hence, it leaks nothing about the original matrix.

Of the three algorithms for solving a system of linear equations, let us first look more closely at Gaussian elimination with backsubstitution. Algorithm 11 gives the privacy-preserving version of the Gaussian elimination algorithm. The computation of coefficients is done in place for both the Gaussian and the LU decomposition algorithms.

At the start of the algorithm (line 1), the rows of the input matrix $[\![\mathbf{A}]\!]$ are shuffled along with the elements of the dependent variable vector $[\![\boldsymbol{b}]\!]$ retaining the relations. On lines 4-12, the pivot element is located from the remaining matrix rows and then the rows are interchanged so that the one with the pivot element becomes the current row. Note that all the matrix indices are public and, hence, all of the conditionals work in the public setting. As we need to use the pivot element as the divisor, we need to check whether it is 0. However, we do not want to give out information about the location of this value so, on line 13, we privately make a note whether any of the pivot elements is 0, and on line 33, we finish the algorithm early if we are dealing with a 0. Note that in the platform we are using, division by 0 will not be reported during privacy-preserving computations as this would reveal the divisor immediately.

On lines 16 - 21, elements on the pivot line are reduced. Similarly, on lines 22 - 30, elements below the pivot line are reduced. Finally, on lines 35 - 38, backsubstitution is performed to get the values of the coefficients.

Algorithm 11: Privacy-preserving Gaussian elimination with backsubstitution

Data: a $k \times k$ matrix $[\![\mathbf{A}]\!] = [\![\mathbf{X}]\!]^T [\![\mathbf{X}]\!]$, a vector $[\![\boldsymbol{b}]\!] = [\![\mathbf{X}]\!]^T [\![\boldsymbol{y}]\!]$ of k values for the dependent variable

Result: Vector $[\![\boldsymbol{b}]\!]$ of coefficients

Shuffle $[\![\mathbf{A}]\!], [\![\boldsymbol{b}]\!]$ retaining the dependencies
Let $[\![c]\!] \leftarrow$ **false** be a boolean value
for $i \in \{1, \ldots, k-1\}$ **do**
  $[\![\boldsymbol{m}]\!]$ be a subvector of $[\![\mathbf{A}_{u,v}]\!]$ such that $u \in \{i+2, \ldots, k\}, v = i$
  $([\![t]\!], irow) \leftarrow \mathbf{maxLoc}([\![\boldsymbol{m}]\!])$
  $irow \leftarrow irow + i + 1$
  if $irow \neq i$ **then**
    for $j \in \{1, \ldots, k\}$ **do**
      Exchange elements $[\![\mathbf{A}_{irow,j}]\!]$ and $[\![\mathbf{A}_{i,j}]\!]$
    end
    Exchange element $[\![b_{irow}]\!]$ and $[\![b_i]\!]$
  end
  $[\![c]\!] \leftarrow [\![c]\!] \vee ([\![\mathbf{A}_{i,i}]\!] = 0)$
  $[\![pivinv]\!] \leftarrow [\![\mathbf{A}_{i,i}]\!]^{-1}$
  $[\![\mathbf{A}_{i,i}]\!] \leftarrow 1$
  for $j \in \{1, \ldots, k\}$ **do**
    if $j \neq i$ **then**
      $[\![\mathbf{A}_{i,j}]\!] \leftarrow [\![\mathbf{A}_{i,j}]\!] \cdot [\![pivinv]\!]$
    end
    $[\![b_i]\!] \leftarrow [\![b_i]\!] \cdot [\![pivinv]\!]$
  end
  for $m \in \{i+1, \ldots, k\}$ **do**
    for $j \in \{1, \ldots, k\}$ **do**
      if $j \neq i$ **then**
        $[\![\mathbf{A}_{m,j}]\!] \leftarrow [\![\mathbf{A}_{m,j}]\!] - [\![\mathbf{A}_{i,j}]\!] \cdot [\![\mathbf{A}_{m,i}]\!]$
      end
    end
    $[\![b_m]\!] \leftarrow [\![b_m]\!] - [\![b_i]\!] \cdot [\![\mathbf{A}_{m,i}]\!]$
    $[\![\mathbf{A}_{m,i}]\!] \leftarrow 0$
  end
end
if declassify$([\![c]\!])$ **then**
  return "Singular matrix"
end
$[\![b_k]\!] \leftarrow \dfrac{[\![b_k]\!]}{[\![\mathbf{A}_{k,k}]\!]}$
for $i \in \{k-1, \ldots, 1\}$ **do**
  $[\![b_i]\!] \leftarrow [\![b_i]\!] - \sum_{j=i+2}^{k} [\![\mathbf{A}_{i,j}]\!] \cdot [\![b_j]\!]$
end
return $[\![\boldsymbol{b}]\!]$

Let us look at the differences between the original and the privacy-preserving version of Gaussian elimination with backsubstitution. As the platform provides us with functions for addition, multiplication and inversion, we do not need to use any workarounds in the main computations. The first additional method we have to take into account, however, is that we start with shuffling the rows of the input matrix and the vector containing the dependent variable keeping the relationship intact. We do this, because we later need to know the location of the largest element, and we do not want to reveal this information on the input matrix. We can shuffle the rows without breaking the algorithm because the input matrix and vector represent a set of linear equations and the order of these equations does not influence the outcome. Furthermore, during execution, the Gaussian algorithm itself rearranges the rows of the matrix based on the location of the pivot element.

The main difference between the original and the privacy-preserving Gaussian elimination algorithm is actually in the **maxLoc** function. In the original version, elements are compared one-by-one to the largest element so far and at the end of the subroutine, the greatest element and its location have been found. As for our system, we basically do the same thing only we use oblivious choice instead of the straightforward if-clauses. This way, we are able to keep the largest element secret and we only reveal its location at the end without finding out other relationships between elements in the vector during the execution of this algorithm.

Let us now look at LU decomposition. In the ordinary setting, this method is faster than the Gaussian elimination method. LU decomposition uses matrix decomposition to achieve this speed-up. If we can decompose the input matrix into a lower and upper triangular matrix $\mathbf{L}$ and $\mathbf{U}$, respectively, so that $\mathbf{L} \cdot \mathbf{U} = \mathbf{A}$, we can use forward substitution and backsubstitution on these matrices, similarly to the process we used in the Gaussian elimination method. Algorithm 12 gives the privacy-preserving version of LU decomposition. Note, that the elements on the diagonal of the lower triangular matrix $\mathbf{L}$ are equal to 1. Knowing this, $\mathbf{L}$ and $\mathbf{U}$ can be returned as one matrix such that the diagonal and elements above it belong to the upper triangular matrix $\mathbf{U}$ and the elements below the diagonal belong to the lower triangular matrix $\mathbf{L}$ without losing any information.

Similarly to Algorithm 11, first the pivot element is found using the **maxLoc** function. After the elements are exchanged, the row permutations are saved for use in the algorithm for solving the set of linear equations. As a result, the decomposition matrix and the permutations are returned. The permutations are public information but they reveal nothing about the original dataset because the rows have been shuffled before inputting them to the decomposition algorithm similarly to what was done in Algorithm 11.

The difference between the original and the privacy-preserving versions of the matrix decomposition algorithm is in using the **maxLoc** function for the same reasons as for Algorithm 11.

Algorithm 13 shows how to solve a set of linear equations using LU decomposition. The matrix rows are shuffled as in Algorithm 11 and the LU decomposition matrix is composed using the **LUDecomp** function. As an additional result we

Algorithm 12: LUDecomp: Privacy-preserving LU decomposition of a matrix

Data: a $k \times k$ matrix $[\![\mathbf{B}]\!]$
Result: The LU decomposition matrix $[\![\mathbf{B}]\!]$ and $\boldsymbol{q}$ containing the row permutations

1 Let $[\![c]\!] \leftarrow 0$ be a boolean value
2 **for** $i \in \{1, \ldots, k\}$ **do**
3 $\quad [\![\boldsymbol{m}]\!]$ be a subvector of $[\![\mathbf{B}_{u,v}]\!]$ such that $u \in \{i, \ldots, k\}, v = i$
4 $\quad ([\![t]\!], irow) \leftarrow \mathbf{maxLoc}([\![\boldsymbol{m}]\!])$
5 $\quad$ **if** $irow \neq i$ **then**
6 $\quad\quad$ **for** $j \in \{1, \ldots, k\}$ **do**
7 $\quad\quad\quad$ Exchange elements $[\![\mathbf{B}_{irow,j}]\!]$ and $[\![\mathbf{B}_{i,j}]\!]$
8 $\quad\quad$ **end**
9 $\quad$ **end**
10 $\quad [\![c]\!] \leftarrow [\![c]\!] \vee ([\![\mathbf{B}_{i,i}]\!] = 0)$
11 $\quad q_i \leftarrow irow$
12 $\quad [\![ipiv]\!] \leftarrow [\![\mathbf{B}_{i,i}]\!]^{-1}$
13 $\quad$ **for** $m \in \{i+1, \ldots, k\}$ **do**
14 $\quad\quad [\![\mathbf{B}_{m,i}]\!] \leftarrow [\![\mathbf{B}_{m,i}]\!] \cdot [\![ipiv]\!]$
15 $\quad\quad$ **for** $j \in \{i+1, \ldots, k\}$ **do**
16 $\quad\quad\quad [\![\mathbf{B}_{m,j}]\!] \leftarrow [\![\mathbf{B}_{m,j}]\!] - [\![\mathbf{B}_{m,i}]\!] \cdot [\![\mathbf{B}_{i,j}]\!]$
17 $\quad\quad$ **end**
18 $\quad$ **end**
19 **end**
20 **if declassify**($[\![c]\!]$) **then**
21 $\quad$ **return** "Singular matrix"
22 **end**
23 **return** $([\![\mathbf{B}]\!], \boldsymbol{q})$

receive the permutation that was done for pivoting purposes during the decomposition phase. Next, on rows 3 - 6, elements of the vector $[\![\boldsymbol{b}]\!]$ containing the dependent variable are permuted to be in concurrence with the permutations that were performed during the decomposition phase. Normally, this step does not need to be done, as the elements can be accessed on the fly using the permutation vector $\boldsymbol{q}$, but in the privacy-preserving setting, it is more feasible to first rearrange the vector and then access the elements in order.

On rows 7 - 9, forward substitution is performed using the values from the lower triangular matrix. Finally, on rows 10 - 12, backsubstitution is performed using the values from the upper triangular matrix.

In addition to the two elimination methods, we decided to look at an iterative algorithm for solving sets of linear equations to test the difference in performance and accuracy. We chose the conjugate gradient method which is a quadratic programming task. If the computations are done without errors, it is guaranteed to converge in k steps, where k is the number of columns in the matrix [1].

As our matrix is symmetric and positive semi-definite, we can use the simplest version of this method with the exception that we forego the comparison

Algorithm 13: Solving linear regression using the LU decomposition matrix in a privacy-preserving setting

Data: a $k \times k$ matrix $[\![\mathbf{A}]\!] = [\![\mathbf{X}]\!]^T [\![\mathbf{X}]\!]$, a vector $[\![\boldsymbol{b}]\!] = [\![\mathbf{X}]\!]^T [\![\boldsymbol{y}]\!]$ of k values for the dependent variable
Result: A vector $[\![\boldsymbol{b}]\!]$ of coefficients

Shuffle $[\![\mathbf{A}]\!], [\![\boldsymbol{b}]\!]$ retaining the dependencies
$([\![\mathbf{B}]\!], \boldsymbol{q}) \leftarrow \mathbf{LUDecomp}([\![\mathbf{A}]\!])$
for $i \in \{1, \ldots, k\}$ **do**
  Let $j \leftarrow \boldsymbol{q}i$
  Exchange elements $[\![b_i]\!]$ and $[\![b_j]\!]$
end
for $i \in \{2, \ldots, k\}$ **do**
  $[\![b_i]\!] \leftarrow [\![b_i]\!] - \sum_{j=1}^{i} [\![\mathbf{B}_{i,j}]\!] \cdot [\![b_j]\!]$
end
for $i \in \{k, \ldots, 1\}$ **do**
  $[\![b_i]\!] \leftarrow \left([\![b_i]\!] - \sum_{j=i+1}^{k} [\![\mathbf{B}_{i,j}]\!] \cdot [\![b_j]\!] \right) \cdot [\![\mathbf{B}_{i,i}]\!]^{-1}$
end
return $[\![\boldsymbol{b}]\!]$

for finding when the method is converging, and simply do a fixed number of iterations that exceeds the conversion point.

Algorithm 14 shows how to solve a set of linear equations using the conjugate gradient method. As discussed, the only real difference from the original algorithm is in the detail that we do not measure the convergence as this can give too much information about the original matrix. Instead, we simply fix a number of iterations z. We also considered computing the optimal number of iterations as given in [1], but decided against it, as this computation will be almost as expensive as performing a hundred iterations and, in addition, the computed iteration count reveals information about the data itself. Considering that the initial convergence of the conjugate gradient method is rapid during a small number of iterations [1] and that the algorithm should converge in k steps, 10 iterations are enough to get an estimate of the solution. If more iterations are needed, it is sensible to use the elimination methods instead.

7 The implementation of Rmind

7.1 Implementation architecture

We have built an implementation of the privacy-preserving statistical analysis tool on the Sharemind secure computation framework [3]. We used the `additive3pp` PDK originally introduced in [8] as the computation backend. This PDK uses secret sharing among three servers to protect the confidentiality of the

Algorithm 14: Privacy-preserving conjugate gradient method

Data: a $k \times k$ matrix $[\![\mathbf{A}]\!] = [\![\mathbf{X}]\!]^T[\![\mathbf{X}]\!]$, a vector $[\![\boldsymbol{b}]\!] = [\![\mathbf{X}]\!]^T[\![\boldsymbol{y}]\!]$ of k values for the dependent variable, public number of iterations z

Result: A vector $[\![\boldsymbol{x}]\!]$ of coefficients

Let $[\![\boldsymbol{x}]\!]$ be a vector of k values 0
$[\![\boldsymbol{x}]\!] \leftarrow [\![\boldsymbol{x}]\!]^T$
$[\![\boldsymbol{r}]\!], [\![\boldsymbol{p}]\!] \leftarrow [\![\boldsymbol{b}]\!]$
repeat
  $[\![\alpha]\!] \leftarrow \frac{[\![\boldsymbol{r}]\!]^T[\![\boldsymbol{r}]\!]}{[\![\boldsymbol{p}]\!]^T[\![\mathbf{A}]\!][\![\boldsymbol{p}]\!]}$
  $[\![\boldsymbol{x}]\!] \leftarrow [\![\boldsymbol{x}]\!] + \alpha[\![\boldsymbol{p}]\!]$
  $[\![\boldsymbol{s}]\!] \leftarrow [\![\boldsymbol{r}]\!] - \alpha[\![\mathbf{A}]\!][\![\boldsymbol{p}]\!]$
  $[\![\beta]\!] \leftarrow \frac{[\![\boldsymbol{s}]\!]^T[\![\boldsymbol{s}]\!]}{[\![\boldsymbol{r}]\!]^T[\![\boldsymbol{r}]\!]}$
  $[\![\boldsymbol{p}]\!] \leftarrow [\![\boldsymbol{s}]\!] + [\![\beta]\!][\![\boldsymbol{p}]\!]$
  $[\![\boldsymbol{r}]\!] \leftarrow [\![\boldsymbol{s}]\!]$
  $z \leftarrow z - 1$
until $z = 0$;
return $[\![\boldsymbol{x}]\!]^T$

data and has a wide range of implemented protocols (see Section 2.1). Figure 4 shows the architecture of the tool and how different Sharemind components were used in its implementation.

We implemented a command line utility for uploading data that can secret-share CSV-formatted files so that each server gets one share of each value in the input file. These tables can then later be used by the Rmind tool in the analysis. Rmind is an interactive tool with a command line interface that allows the analyst to manipulate data tables and run statistical analyses. Rmind is implemented in the Haskell programming language because of the ease of implementing interpreters and compilers in Haskell.

We consciously made the choice not to build on top of an existing system (e.g. R, SPSS etc) for two reasons. First, we cannot re-use the statistical functions implemented by existing tools, because they are implemented on standard processors and cannot be easily retargeted to secure computation. Second, existing tools have no elegant support for separating public and private data. Solutions have been proposed, e.g., in [17] that have a tuned system that performs just the minimal amount of secure computation, interleaving public and private operations. However, their paper does not describe a way for saving and reusing the results of secure computation on the server side. In our tool, all data, including intermediate tables, are stored remotely, and only statistical procedures can make their results public.

Commands of Rmind are preprocessed at the client and sent to all Sharemind servers, where the necessary secure computation procedures are executed. These procedures are implemented in the SecreC 2 programming language [5]

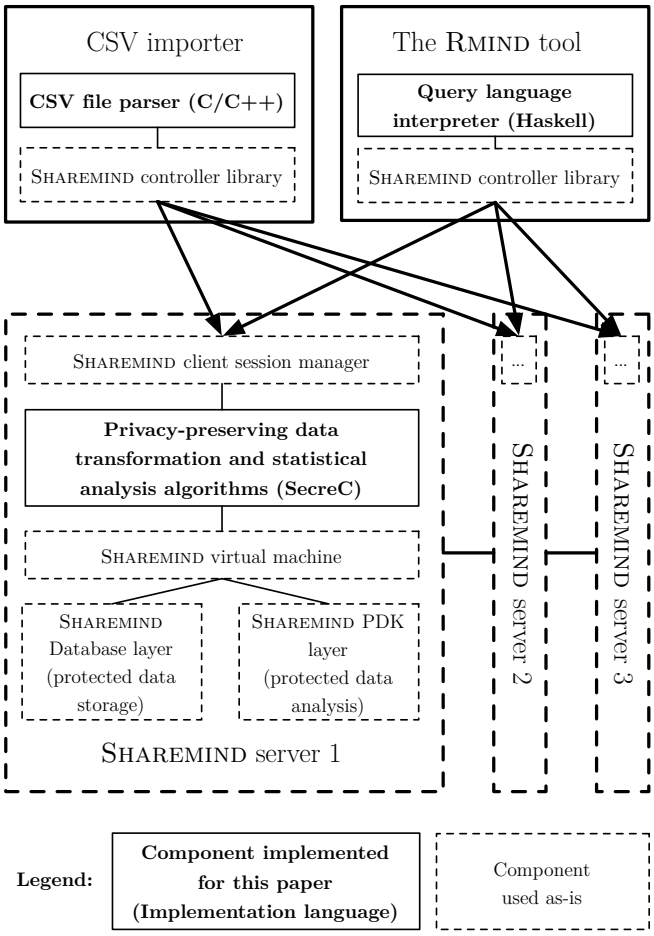

Fig. 4. The architecture of the RMIND tool (servers 2 and 3 are identical to server 1)

that separates public and private data on a type system level, thus also supporting the data tagging design goal of our tool. If the procedure needs to use data uploaded by a user or the result of a previous intermediate result, it can access it from the database system built into SHAREMIND that also separates data based on which protection domain it belongs to.

RMIND can only perform operations for which the respective procedure has been deployed on all SHAREMIND servers. This is an additional control mechanism to ensure that no unauthorised operations can be performed. The three servers shown in our architecture must, therefore, be deployed by independent organisations interested in preserving the privacy of the data. If that assumption holds, the whole RMIND systems provides much improved privacy when compared to traditional statistical tools.

7.2 Privacy-preserving statistical analysis language

Managing public and private variables We have adapted a subset of the language used by the statistical analysis tool R into our privacy-preserving setting. In RMIND, data is stored in public and private arrays of signed integers, floating point numbers or boolean values. The language also supports public strings for names. We are not giving the full language description here and focus only on the parts that are important from a privacy perspective.

Functions can return either public or private data. For example, the `load` function that loads a private table from the database, returns a value representing a database with private values. However, functions that describe the sizes of tables, such as `nrow` and `ncol` functions, return public values. The `typeof` function returns a string containing the data type of the expression, including its security type. The values of public expressions can be printed using `print`. Private variables can be used in statistical analysis that may print out their result. Intermediate private results can also be stored in the database with the `store.table` function.

```
tbl <- load("db", "table")
rowcount <- nrow(tbl)
print (typeof (tbl$col))
store.table("db", "table2", list("x", "y"),
 list(tbl$x * 10, tbl$y + 100))
```

RMIND has several control structures like `for`, `if`, `repeat` and `while`. Arrays are indexed with rectangular brackets (`a[i]`). Currently, conditional expressions in control structures and indices can only be public expressions. RMIND lets the analyst define procedures similarly to R and supports features like keyword arguments and argument lists.

```
for (i in 1:10) print (a[i])
```

Preparing private data for analysis RMIND can prepare private data for analysis using a range of transformations that result in new private data. For example, it can perform arithmetic, comparisons and logic on private data to compute new attributes. Private data can be combined with public data, but the public data will be converted to private in the process.

```
products <- tbl$column1 *
tbl$column2
mask <- tbl$column < 10
```

There are two syntactic ways for filtering private data. There is a simpler inline version and a more flexible procedural version. The second can easily process tables. Filtering is implemented using techniques described in Section 3.

```
c <- tbl$col1[tbl$col2 < 10]
t <- subset(tbl, col1 < 10 & col2 != 1)
```

Tables and vectors can be sorted using the `sort` procedure. Tables can be linked using `merge`. The underlying SHAREMIND system uses protocols described in [7] and [41], respectively.

```
sortedcol <- sort(tbl$col)
tbl3 <- merge(tbl1, "key1", tbl2, "key2")
```

Analyzing private data Table 2 shows the statistical analysis features implemented in RMIND using the respective algorithms in this paper. For some operations, such as `lm`, several algorithms are available. The implementation has a default one, but the user can select the preferred one using the corresponding parameter. All operations run on private data and return a public result.

Rmind operation	**Statistical value computed or plot drawn**
`sum, mean, min/max`	sum and mean of values, smallest/largest value
`median, sd, var, cov`	median, standard deviation, variance and covariance
`fivenum, boxplot`	five number summary and/or box plot
`hist, freqplot, heatmap`	histogram, frequency plot or heatmap
`mad`	MAD (median absolute deviation)
`rm.outliers`	remove outliers with quantiles or MAD
`t.test`	paired and standard t-test with (non-)equal variances
`wilcoxon.test`	Wilcoxon rank sum test and signed rank test
`mann.whitney.test`	Mann-Whitney test (extended Wilcoxon rank sum test)
`chisq.test`	χ^2 tests with two or more categories
`multiple.t.test`	multiple t-tests with Benjamini-Hochberg correction
`multiple.chisq.test`	multiple χ^2 tests with Benjamini-Hochberg correction
`lm`	linear regression (several choices for private algorithm)

Table 2. Statistical operations in RMIND

7.3 Performance analysis

We tested the performance of RMIND on a SHAREMIND installation running on three computers with 3 GHz 6-core Intel CPUs with 8 GB RAM per core (a total of 48 GB RAM). The computers were connected using gigabit ethernet network interfaces. While a subset of these algorithms have been benchmarked in [4], we have optimized the implementation and redone all benchmarks for this paper using the new RMIND tool. The benchmarks in [4] were performed on an identical hardware setting, but they are less optimized and use an older version of SHAREMIND.

Tables 3 and 4 show the performance of statistical operations in comparison with earlier work in [4]. We see, on average, an order of magnitude improvement in performance. The majority of operations were not implemented in previous work so no comparison could be made. We note that the performance measures

should not be considered linear in the size of the input, due to the effects described in Section 2.4.

Operation	***Inputs***	***Time*** (RMIND)	***Time*** ([4])
`mean`	2 000	0.05 s	—
`min/max`	2 000	0.2 s	3 s
`median`	2 000	4.7 s	—
`sd`	2 000	4.4 s	—
`var`	2 000	4.4 s	—
`cov`	2×1 000	3.5 s	—
`fivenum`	2 000	2.4 s	21 s
`hist`	2 000	3.4 s	16 s
`freqplot` *(10 classes)*	2 000	0.5 s	—
`heatmap`	2×1 000	5.8 s	—
`rm.outliers` *(quantiles)*	2 000	3.3 s	—
`rm.outliers` *(MAD)*	2 000	18.3 s	—
`merge`	5×2 000 3×2 000	23.7 s	28 s
`sort`	10×1 000	13.1 s	—
`t.test`	2×1 000	4.2 s	167 s
`t.test` *(paired)*	2×1 000	2.6 s	98 s
`chisq.test` *(2 classes)*	2 000	0.1 s	9 s
`chisq.test` *(5 classes)*	2 000	0.4 s	23 s
`wilcoxon.test`			
(signed rank)	2×1 000	1.5 s	38 s
(rank sum)	2×1 000	2.7 s	34 s
`mann.whitney.test`	2×1 000	2.7 s	—
Benjamini-Hochberg[1]	1 000	52.3 s	—

Table 3. Performance of RMIND operations (in seconds)

[1] We measured the Benjamini-Hochberg procedure standalone on 1000 test results, without the multiple tests that lead to it.

Acknowledgments

This work was supported by the European Regional Development Fund through the Estonian Center of Excellence in Computer Science, EXCS and by the Estonian Research Council under Institutional Research Grants IUT2-1 and IUT27-1. It has also received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 284731.

Operation	***Time*** (RMIND)
`lm` *(simple)*	33.9 s
`lm` *(2 variables, inverse)*	0.6 s
`lm` *(3 variables, inverse)*	1.1 s
`lm` *(4 variables, Gaussian)*	2.9 s
`lm` *(4 variables, LU decomp.)*	3.1 s
`lm` *(4 variables, conj. grad.)*	5.9 s
`lm` *(7 variables, Gaussian)*	9.4 s
`lm` *(7 variables, LU decomp.)*	8.7 s
`lm` *(7 variables, conj. grad.)*	7.8 s
`lm` *(10 variables, Gaussian)*	21.5 s
`lm` *(10 variables, LU decomp.)*	19.1 s
`lm` *(10 variables, conj. grad.)*	11 s

Table 4. Performance of RMIND linear regression on 10000-element arrays (in seconds)

References

1. Owe Axelsson. Iteration number for the conjugate gradient method. *Mathematics and Computers in Simulation*, 61(36):421 – 435, 2003. MODELLING 2001 - Second IMACS Conference on Mathematical Modelling and Computational Methods in Mechanics, Physics, Biomechanics and Geodynamics.
2. Y. Benjamini and Y. Hochberg. Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing. *Journal of the Royal Statistical Society. Series B (Methodological)*, 57(1):289–300, 1995.
3. Dan Bogdanov. *Sharemind: programmable secure computations with practical applications*. PhD thesis, University of Tartu, 2013.
4. Dan Bogdanov, Liina Kamm, Sven Laur, Pille Pruulmann-Vengerfeldt, Riivo Talviste, and Jan Willemson. Privacy-preserving statistical data analysis on federated databases. In *Proceedings of the Annual Privacy Forum. APF'14*, volume 8450 of *LNCS*, pages 30–55. Springer, 2014.
5. Dan Bogdanov, Peeter Laud, and Jaak Randmets. Domain-Polymorphic Programming of Privacy-Preserving Applications. In *Proceedings of the Ninth ACM SIGPLAN Workshop on Programming Languages and Analysis for Security*, 2014. To appear.
6. Dan Bogdanov, Sven Laur, and Riivo Talviste. Oblivious Sorting of Secret-Shared Data. Technical Report T-4-19, Cybernetica, `http://research.cyber.ee/`., 2013.
7. Dan Bogdanov, Sven Laur, and Riivo Talviste. A Practical Analysis of Oblivious Sorting Algorithms for Secure Multi-party Computation. In *Secure IT Systems - 19th Nordic Conference, NordSec 2014*. Springer, 2014. To appear.
8. Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In Sushil Jajodia and Javier Lopez, editors, *Proceedings of the 13th European Symposium on Research in Computer Security, ESORICS '08*, volume 5283 of *Lecture Notes in Computer Science*, pages 192–206. Springer, 2008.
9. Dan Bogdanov, Margus Niitsoo, Tomas Toft, and Jan Willemson. High-performance secure multi-party computation for data mining applications. *International Journal of Information Security*, 11(6):403–418, 2012.

10. Dan Bogdanov, Riivo Talviste, and Jan Willemson. Deploying secure multi-party computation for financial data analysis (short paper). In *Proceedings of FC 2012*, pages 57–64, 2012.
11. Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas P. Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael I. Schwartzbach, and Tomas Toft. Secure Multiparty Computation Goes Live. In *Proceedings of FC 2009*, pages 325–343, 2009.
12. Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. Lof: Identifying density-based local outliers. In *Proceedings of CM SIGMOD 2000*, pages 93–104, 2000.
13. Martin Burkhart, Mario Strasser, Dilip Many, and Xenofontas A. Dimitropoulos. SEPIA: Privacy-Preserving Aggregation of Multi-Domain Network Events and Statistics. In *Proceedings of USENIX 2010*, pages 223–240, 2010.
14. Ran Canetti, Yuval Ishai, Ravi Kumar, Michael K. Reiter, Ronitt Rubinfeld, and Rebecca N. Wright. Selective private function evaluation with applications to private statistics. In *Proceedings of PODC 2001*, pages 293–304. ACM, 2001.
15. Octavian Catrina and Sebastiaan De Hoogh. Improved primitives for secure multiparty integer computation. In *Proceedings of the 7th international conference on Security and cryptography for networks*, SCN'10, pages 182–199, Berlin, Heidelberg, 2010. Springer-Verlag.
16. David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing. STOC'88*, pages 11–19, 1988.
17. Koji Chida, Gembu Morohashi, Hitoshi Fuji, Fumihiko Magata, Akiko Fujimura, Koki Hamada, Dai Ikarashi, and Ryuichi Yamamoto. Implementation and evaluation of an efficient secure computation system using R for healthcare statistics. *Journal of the American Medical Informatics Association*, 04, 2014.
18. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer, 2012.
19. Wenliang Du and Mikhail J. Atallah. Privacy-preserving cooperative statistical analysis. In *Proceedings of ACSAC 2001*, pages 102–110, 2001.
20. Wenliang Du, Shigang Chen, and Yunghsiang S. Han. Privacy-preserving multivariate statistical analysis: Linear regression and classification. In *Proceedings of SDM 2004*, pages 222–233, 2004.
21. Cynthia Dwork. Differential privacy. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Proceedings of the 33rd International Colloquium on Automata, Languages and Programming. ICALP'06*, volume 4052 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2006.
22. Khaled El Emam, Saeed Samet, Jun Hu, Liam Peyton, Craig Earle, Gayatri C. Jayaraman, Tom Wong, Murat Kantarcioglu, Fida Dankar, and Aleksander Essex. A Protocol for the Secure Linking of Registries for HPV Surveillance. *PLoS ONE*, 7(7):e39915, 07 2012.
23. Martin Franz and Stefan Katzenbeisser. Processing encrypted floating point signals. In *Proceedings of the thirteenth ACM multimedia workshop on Multimedia and security*, MM&Sec '11, pages 103–108, New York, NY, USA, 2011. ACM.
24. Martin Geisler. *Cryptographic Protocols: Theory and Implementation.* PhD thesis, Aarhus University, February 2010.

25. Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing. STOC'09*, pages 169–178. ACM, 2009.
26. Koki Hamada, Ryo Kikuchi, Dai Ikarashi, Koji Chida, and Katsumi Takahashi. Practically Efficient Multi-party Sorting Protocols from Comparison Sort Algorithms. In *Proc. of ICISC'12*, volume 7839 of *LNCS*, pages 202–216. Springer, 2013.
27. Frank R. Hampel. A general qualitative definition of robustness. *The Annals of Mathematical Statistics*, 42(6):1887–1896, 1971.
28. Frank R. Hampel. The influence curve and its role in robust estimation. *Journal of the American Statistical Association*, 69(346):383–393, June 1974.
29. Wilko Henecka, Stefan Kögl, Ahmad-Reza Sadeghi, Thomas Schneider, and Immo Wehrenberg. TASTY: tool for automating secure two-party computations. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *Proceedings of the 17th ACM Conference on Computer and Communications Security. CCS'10*, pages 451–462. ACM, 2010.
30. Magnus R. Hestenes and Eduard Stiefel. Methods of Conjugate Gradients for Solving Linear Systems. *Journal of Research of the National Bureau of Standards*, 49(6):409–436, December 1952.
31. Myles Hollander and Douglas A Wolfe. *Nonparametric statistical methods*. John Wiley New York, 2nd ed. edition, 1999.
32. Rob J Hyndman and Yanan Fan. Sample quantiles in statistical packages. *The American Statistician*, 50(4):361–365, 1996.
33. Marek Jawurek and Florian Kerschbaum. Fault-tolerant privacy-preserving statistics. In *Privacy Enhancing Technologies*, volume 7384 of *LNCS*, pages 221–238. Springer, 2012.
34. Liina Kamm, Dan Bogdanov, Sven Laur, and Jaak Vilo. A new way to protect privacy in large-scale genome-wide association studies. *Bioinformatics*, 29(7):886–893, 2013.
35. Liina Kamm and Jan Willemson. Secure Floating-Point Arithmetic and Private Satellite Collision Analysis. Cryptology ePrint Archive, Report 2013/850, 2013. `http://eprint.iacr.org/`.
36. Gopal K Kanji. *100 statistical tests*. Sage, 2006.
37. Florian Kerschbaum. Practical privacy-preserving benchmarking. In *Proceedings of IFIP TC-11 SEC 2008*, volume 278, pages 17–31. Springer US, 2008.
38. Eike Kiltz, Gregor Leander, and John Malone-Lee. Secure computation of the mean and related statistics. In *Procedings of TCC 2005*, volume 3378 of *LNCS*, pages 283–302. Springer, 2005.
39. Benjamin Kreuter, Abhi Shelat, Benjamin Mood, and Kevin R. B. Butler. PCF: A Portable Circuit Format for Scalable Two-Party Secure Computation. In Samuel T. King, editor, *USENIX Security*, pages 321–336. USENIX Association, 2013.
40. Sven Laur, Riivo Talviste, and Jan Willemson. From Oblivious AES to Efficient and Secure Database Join in the Multiparty Setting. In *Proceedings of ACNS'13*, volume 7954 of *LNCS*, pages 84–101. Springer, 2013.
41. Sven Laur, Riivo Talviste, and Jan Willemson. From Oblivious AES to Efficient and Secure Database Join in the Multiparty Setting. In *Applied Cryptography and Network Security*, volume 7954 of *LNCS*, pages 84–101. Springer, 2013.
42. Sven Laur, Jan Willemson, and Bingsheng Zhang. Round-Efficient Oblivious Database Manipulation. In *Proceedings of ISC 2011*, pages 262–277, 2011.

43. Qinghua Li and Guohong Cao. Efficient privacy-preserving stream aggregation in mobile sensing with low aggregation error. In Emiliano Cristofaro and Matthew Wright, editors, *Privacy Enhancing Technologies*, volume 7981 of *Lecture Notes in Computer Science*, pages 60–81. Springer Berlin Heidelberg, 2013.
44. Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay — a secure two-party computation system. In Proceedings of the 13th USENIX Security Symposium (2004), pp. 287-302., 2004.
45. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Proceedings of the 17th International Conference on the Theory and Application of Cryptographic Techniques, EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 1999.
46. William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 3 edition, 2007.
47. Reimo Rebane. A Feasibility Analysis of Secure Multiparty Computation Deployments. Master's thesis, Institute of Computer Science, University of Tartu, 2012.
48. Elaine Shi, T.-H. Hubert Chan, Eleanor G. Rieffel, Richard Chow, and Dawn Song. Privacy-preserving aggregation of time-series data. In *NDSS*. The Internet Society, 2011.
49. Nigel P. Smart and Frederik Vercauteren. Fully homomorphic simd operations. *Des. Codes Cryptography*, 71(1):57–81, 2014.
50. Herbert A Sturges. The choice of a class interval. *Journal of the American Statistical Association*, 21(153):65–66, 1926.
51. Hiranmayee Subramaniam, Rebecca N. Wright, and Zhiqiang Yang. Experimental analysis of privacy-preserving statistics computation. In *Proceedings of SDM 2004*, volume 3178 of *LNCS*, pages 55–66. Springer, 2004.
52. Latanya Sweeney. k-anonymity: a model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):557–570, 2002.
53. Frank Wilcoxon. Individual Comparisons by Ranking Methods. *Biometrics Bulletin*, 1(6):80–83, 1945.
54. Zhiqiang Yang, Rebecca N. Wright, and Hiranmayee Subramaniam. Experimental analysis of a privacy-preserving scalar product protocol. *Computer Systems Science & Engineering*, 21(1), 2006.
55. Andrew Chi-Chih Yao. Protocols for Secure Computations (Extended Abstract). In *Proceedings of FOCS'82*, pages 160–164. IEEE, 1982.