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 analysts have direct access to the data, thus creating a risk to privacy. Even if data analysts are diligent and keep secrets, the data in their possession can leak through an external attack.

Cryptographic research has given us encryption schemes that protect databases until the data are processed. However, existing analysis tools require us to decrypt data before processing, bringing us back to the privacy problem. Emerging cryptographic technologies like secure multi-party computation (SMC) solve the problem by allowing data to be processed in encrypted form. This enables a new kind of computer that can manipulate individual data records without seeing them—much like a blind craftsman sculpting a work of art.

Secure multi-party computation was considered impractical for years, but clever protocol design and determined engineering have led to first real-world applications [12, 11]. Therefore, we decided to investigate the feasibility of using SMC to perform statistical analyses with better privacy.

We interviewed statisticians to explain the privacy potential of SMC technology and find their expectations [5]. They saw potential, but also had concerns.

First, statisticians were used to seeing individual values and were unsure if they can find inconsistencies and ensure analytical quality without such access. SMC takes away control from the analyst by only disclosing the query results.

The second concern was the lack of user-friendly tools for cryptographically secure data analysis. Statisticians are used to the workflows of interactive environments like SAS, SPSS and R. The interviewees expected to see a drop-in replacement of their environment with the new privacy guarantees.

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

In 2014, we presented the Estonian Data Protection Agency with a proposal for linking and analyzing two government databases using the tools and methods presented in this paper. Their conclusion was that, according to the legislation, our method does not process personally identifiable information and thus no permit is needed as long as our technical solution is used and the study plan is followed [5]. While this is not yet the prevailing legal position, it marks a paradigm shift in data protection.

Our contribution. This paper builds on our earlier work in [5, 37] where we presented our first attempts on privacy-preserving statistics. In this paper, we have made major improvements. First, we designed and implemented new privacy-preserving algorithms for statistical testing including popular value correction methods for multiple testing. Second, we designed and implemented privacy-preserving methods for multivariate linear regression, which can be easily extended to polynomial regression or linear regression with regard to other basic functions. For this, we developed privacy-preserving methods for solving a set of linear equations based on Gaussian elimination with backsubstitution and LU decomposition. Both are, of course, applicable in other contexts beside statistics. The same applies to our novel privacy-preserving version of the conjugate gradient method for minimizing quadratic forms.

Third, we present Rmind, a privacy-preserving statistical analysis environment designed to resolve user acceptance issues. Rmind supports a complete data analysis process where data are collected from various sources, linked and statistically analysed. The user interface is identical to that of R. We also took great care to fix all details of our algorithms so that they provide the same results as the R tool, to the precision of a floating point comparison.

Fourth, we have implemented the new algorithms, optimised the previous implementations, and provide new performance results that prove the feasibility of the system. Comparable performance results for the entire suite of statistical operations have not been published and thus it has been difficult to estimate whether cryptographically secure statistical analysis is practically feasible or not.

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

Related work. The closest system to what we propose has been introduced by Chida et al. in [19]. They are using the statistics environment R to 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 different database tables.

Another recent implementation with similar goals is by El Emam et al [25]. 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 [16, 21, 22, 41, 40], filtered sums, weighted sums and scalar products [53, 56, 36]. Related papers on private data aggregation have also targeted streaming data [50, 45]. However, all of these components have been implemented as separate special purpose programs using various programming languages and different underlying secure multi-party computation protocol sets. It would be very difficult, if not impossible, to integrate these programs into one comprehensive statistics suite.

2 Designing a tool for secure statistical analysis

2.1 Requirements for a secure statistical analysis tool

There is a perceived need of cryptographic security capability in the following privacy-preserving data integration scenario [5].

1. Data owners and analysts agree on the data format and the study plan.
2. Data owners upload their inputs to a secured database.
3. Data analysts send queries that are in the study plan.
4. Analysts receive analysis results and interpret them in a resulting report.

We propose a tool for performing privacy-preserving statistical studies in the given setting using secure multi-party computation. The difference between Rmind and standard statistical tools is that Rmind collects and analyses data in encrypted form without decrypting them and gives provable privacy guarantees to the analysis process. 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. Rmind is designed with the following goals:

1. **Similarity to existing tools.** Based on the interviews, the end users prefer tools with familiar user interfaces.
2. **Separation of public and private data.** The tool should separate public and private values in data.
3. **Seamless use of cryptographic technology.** Cryptographic processing must be applied automatically when private data are processed.

4. **Features chosen according to real-world needs.** Data transformation and analysis features are chosen after interviews with statisticians to determine the tools they most commonly use.

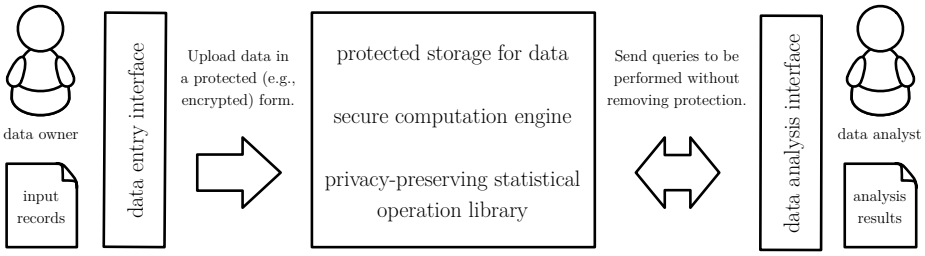

Fig. 1. Design of a privacy-preserving statistical analysis tool and interfaces based on secure computation

The main components of Rmind are the data entry interface, the data analysis interface and the storage and computation backend (see Figure 1). Standalone data entry or analysis interfaces can be implemented on any programming platform supported by the secure multi-party computation framework. Data are tagged public or private during the upload to allow the statistical tool to apply secure multi-party computation on private data. As the study plan is public, we do not need to hide data formats or the algorithms being executed. However, private parameters to the queries can be hidden from the computing parties as any other private inputs, e.g., the criteria for forming subgroups can be kept secret.

2.2 Secure multi-party computation with external parties

Secure multi-party computation (SMC) is a cryptographic technology for computing a function with multiple parties where only the party who provided a particular input can see that input and every party only gets the output specifically intended for them. In the standard model, parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ jointly evaluate a function $f(x_1, \ldots, x_n)$ with outputs $(y_1, \ldots, y_n)$ so that $\mathcal{P}_i$ submits its input x_i and learns the output y_i and nothing about the other inputs and outputs.

Not all parties in the data integration scenario have the resources and motivation to participate equally in the secure analysis. Therefore, we consider three different party types. *Input parties* (data owners) provide the inputs to the analysis and expect that nobody else learns them. *Computing parties* store inputs, participate in secure multi-party computation protocols and give outputs to the *result parties* (the analysts).

SMC protocols can be built using homomorphic encryption [47, 28], garbled circuits [57] and secret sharing [18, 18], among others. All of these can provide a

set of composable cryptographic primitives that implement the secure arithmetic we need for the statistical operations.

There are a number of SMC implementations available. Some provide secure integer arithmetic [46, 14, 32, 27, 20, 3, 42], floating point arithmetic [17, 26], shuffling and sorting [29] and linking [25], but, to our knowledge, only the SHAREMIND framework provides all the operations integrated into a single implementation [10, 44, 43, 38, 8].

2.3 Cryptographically secure operations for statistical analysis

The privacy-preserving statistical analysis algorithms presented in in this paper are independent of a particular SMC approach. Instead, we assume the existence of an abstract set of SMC protocols, or a *protection domain kind* as defined in [7].

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.*

Functionality	**Notation**
Protected storage of a private value x (signed integer, floating point, boolean)	$[\![\mathrm{x}]\!]$
Conversion to the private form	$[\![x]\!] \leftarrow x$
Support for value vectors and matrices	$[\![\boldsymbol{x}]\!]$ and $[\![\mathbf{M}]\!]$
Privacy-preserving binary operations (signed integer, floating point, boolean)	$[\![z]\!] \leftarrow [\![x]\!] \circledast [\![y]\!]$
Privacy-preserving functions	$[\![y]\!] \leftarrow f([\![x]\!])$
Declassify value to computing parties	**declassify**$([\![x]\!])$
Publish value to result parties	**publish**$([\![x]\!])$
Private shuffling, linking and sorting	—

Table 1. Secure computation capabilities and notation for statistical analysis algorithms.

Specifically, we require the existence of the privacy-preserving operations listed in Table 1. The arithmetic operations are used for implementing statistical functions. Secure linking and sorting are required for preparing data tables for analysis. We also require a cryptographically private shuffling protocol that can randomly rearrange the values in a vector or rows in a matrix without leaking the elements being rearranged.

In the simplest model, the input party encrypts its input x to get $[\![x]\!]$. Then, it sends $[\![x]\!]$ to the computing party $\mathcal{CP}$ who computes $[\![y]\!] \leftarrow f([\![x]\!])$ using PDK operations and sends $[\![y]\!]$ to the result party who will decrypt it to learn y. With a secure PDK, $\mathcal{CP}$ learns nothing about x or y in the process.

The threat models for different PDKs vary. For example, threats for a hardware-assisted PDK include backdoors and design flaws. For secure multi-party computation, the main threat is that computing parties collude to reveal private inputs. PDKs also differ in their security assumptions and guarantees, e.g., they may remain private only when there is an honest majority, or withstand malicious tampering. Stronger security guarantees often come at a cost of computational power. Security against passive adversaries (who do not modify software but try to learn private values from its state) is sufficient for processing personal data when the computing parties are organizations with a legal responsibility to protect privacy.

One example of a practically feasible PDK is secure multi-party computation on *additively secret shared* data [9]. If an input party wants to provide a secret value $x \in Z$ (where Z is a quotient ring) as a private input to n computing parties, it uniformly generates *shares* $x_1, \ldots, x_{n-1} \leftarrow Z$ and calculates the final share $x_n \leftarrow x - x_1 - \cdots - x_{n-1}$. Each computing party receives one share x_i that it stores as $[\![x]\!]$. As an individual share x_i is just a uniformly distributed value, no computing party can learn anything about x without colluding with others.

Computing parties can process the shares without recovering the secret. For example, if each computing party has shares x_i and y_i of secrets x and y, they can calculate $z_i \leftarrow x_i + y_i$ to get the shares of $z = x + y$. Further operations in this protection domain kind require more complex protocols, as described in [9, 10]. The privacy and composability proofs for the cited protocols can be found in [6].

2.4 Limiting the leakage of private inputs

In our setting, data owners and analysts agree on a study plan, including what results can be published to the analysts. An ideal privacy goal would be to require that no information about the private inputs is revealed during the computations or in the outputs. However, this is impossible to achieve, as all practically useful outputs contain information about the private inputs.

Instead of forbidding leakage, we will use automatically enforced mechanisms to minimize it. We implement *controlled statistics*—our statistical tool will only publish results that the data owners and computing parties have cleared for publishing. This approach has been shown to be a sufficient and legally acceptable way of protecting personal information and tax secrets in social studies [5].

The privacy-preserving algorithms in this paper are designed to minimize leakage using a combination of techniques that follow the following privacy goals.

Goal 1: Cryptographic security. During the evaluation of a function

$$([\![y_1]\!], \ldots, [\![y_k]\!]) \leftarrow f([\![x_1]\!], \ldots, [\![x_m]\!]),$$

a computing party $\mathcal{CP}$ cannot learn any private input x_j where $j \in \{1, \ldots, m\}$, output y_ℓ where $\ell \in \{1, \ldots, k\}$ or intermediate value computed by f unless the value is published to the result parties using the **publish** function. We also

want to prevent leaking private values through changes in the running time of the algorithm.

We achieve this goal as follows. First, our algorithms process all private values using composable secure operations in the PDK. This prevents leakage through any storage used by the computing party. Second, whenever possible, we design the algorithm as a straight line program. A straight line program consisting of universally composable secure operations is cryptographically secure and universally composable itself [15]. Hence, we can omit the security proofs of such algorithms in this paper (Algorithms 3, 7, 8, 10, 14).

The remaining algorithms are not straight line programs and require separate security proofs. Most of these algorithms (Algorithms 1, 2, 4, 5, 6, 9) declassify the size of the selected subgroup, which is assumed to be public or is a desired output according to the study plan. We just optimize the running time by declassifying certain results slightly earlier so that computing parties can reduce the amount of PDK operations they have to perform. For example, the **cut** function reduces the amount of data to be processed by declassifying the size of the dataset that matches a search criterion, which is usually an allowable output in the study. Three remaining algorithms (Algorithms 11, 12, 13) declassify values that are independent of input data and thus privacy-preserving.

Fortunately, it is sufficient to analyse security in a hybrid model where abstract operations with private values reveal no information and an attacking computing party can make decisions only based on values that are declassified during the computations. Consequently, we must show that the transcript of values declassified during runtime can be simulated knowing only the final published results of the algorithm. If this assumption is satisfied, then the implementation in the hybrid model can be shown to be cryptographically secure. As the composition theorem from [15] assures that the security of the hybrid implementation does not degrade when we replace all abstract operations with the actual PDK operations. As long as all of them are universally composable, we have formally shown that our implementation is cryptographically secure.

Goal 2: Source privacy. An algorithm for computing $f(x_1, \ldots, x_m)$ is source-private if all outputs and all intermediate values do not depend on the order of inputs. If an algorithm for computing $f(x_1, \ldots, x_m)$ is cryptographically secure, it is sufficient to prove that the output distributions of $f(x_1, \ldots, x_m)$ and $f(x_{\pi(1)}, \ldots, x_{\pi(m)})$ coincide for all permutations of inputs π.

Source privacy ensures that the computing parties are not capable of linking computation results with the private inputs of a specific input party. For instance, the simple sum $x_1 + \cdots + x_n$ is source private and, thus, it does not reveal the relation between input parties and their inputs even if we somehow know the values of $x_1, \ldots, x_n$. The latter does not mean that we cannot infer something about x_i. For instance, if all inputs are in a fixed range, the sum can limit the set of potential values of x_i. Similarly, we might get extra information about x_i if we know some other values x_j.

All algorithms in this paper are source-private. Straight line programs with no declassified values are trivially source-private, because their data access patterns

are independent of the inputs. All other statistical functions are independent on the order of inputs. Hence, it is straightforward to achieve source privacy by obliviously shuffling the inputs before the actual computations.

Goal 3: Query restrictions. A privacy-preserving statistical analysis system enforces query restrictions if it provides a mechanism for the input parties to control which computations can be performed by the computing parties on the private inputs and which results can be published to the result parties.

We achieve this goal by combining organizational controls with technical ones. First, the input parties negotiate a study plan that describes the computations to be performed and their publishable results. Second, the study plan is implemented on the Rmind tool using a PDK and the algorithms in this paper. Every result in the study plan is published using the **publish** operation.

Rmind enforces these restrictions by requiring that computing parties have exactly the same version of the study plan deployed at the time of execution. The result party can only run queries for which the necessary algorithms are deployed. Parties cannot upload new code to perform any query. If at least one computing party refuses to run a query, it cannot be run. Input parties can have a greater control over the process, if they or their representatives also host a computing party.

Goal 4: Output privacy. A statistical study is output-private, if its published results do not leak the private inputs of the input parties.

Output privacy cannot be absolute, as this way we learn nothing from the private inputs. In practice, the amount of allowable leakage is strongly application-dependent. In the data integration scenario, data owners help compile the study plan and perform a privacy impact assessment of the publishable outputs. They can then decide to augment the algorithms with limitations. For example, they can require that a result cannot be published if it is a direct aggregation of, e.g., less than five, inputs. Also, they can limit the number of queries that can be sent to prevent the siphoning of private data.

Formal quantification of output privacy should be defined on well-founded mathematical formalisations like k-anonymity [54] and differential privacy [23]. Achieving these goals comes with the cost of reduced precision. The result parties must accept either noisy or coarse-grained results.

Legislations often specify requirements in terms equivalent or similar to k-anonymity, as it is easy to understand and verify such requirements. However, these privacy requirements do not protect against attackers with background information. Differential privacy guarantees security against attackers with unbounded background knowledge, provided that the result parties are willing to accept a potentially unbounded amount of noise added to results [24].

In principle, there is no technical difficulties for implementing cryptographically secure output randomisation that provides differential guarantees similar to the seminal work [23]. However, it must be a separate layer, as in statistical studies, the customers often expect precise results regardless of privacy implications. Also, the amount of accessible background information is limited in a

typical statistical study, or otherwise the study would be redundant. Hence, differential privacy may provide overly conservative results by adding too much noise to outputs.

The authors conclude that the RMIND tool should have optional support for output randomisation techniques that one can use to achieve differential privacy. Moreover, defining a flavour of differential privacy that faithfully models the bounded nature of background information in statistical studies without inconsistencies is an important research goal.

2.5 Achieving efficiency

Protection domain kinds may have specific performance profiles. For example, PDKs based on secret sharing, are significantly more efficient when the algorithms perform many parallel operations together. Figure 2 shows a generic running time profile for a secure multi-party computation operation [10].

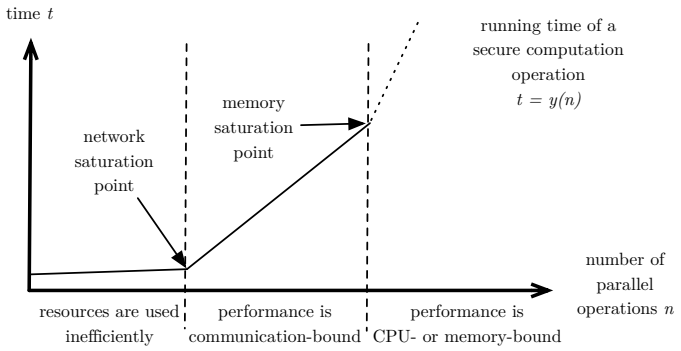

Fig. 2. Performance model of the secure computation protocols based on secret sharing.

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 [49].

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. [51]. This gives us reason to use parallel operations as

much as possible in our algorithm design, utilizing batch processing to prevent exhausting computational resources.

We will allow the algorithm to optimize its running time, if this requires the declassification of a private value that is among the published outputs of the statistical function. For example, we will use aggregates like counts and sums to reduce the amount of data we have to process after certain filtering tasks.

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. Alternatively, existing datasets can be used in analyses, 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 such studies, 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) and the joint database is considered to be *horizontally 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 into a new *vertically 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 privacy-preserving 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. 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 $[\![\boldsymbol{c}]\!]$ 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 $[\![m_i]\!] \leftarrow [\![a_i]\!] \circledast [\![c_i]\!]$, where $\circledast$ is a comparison operation, $i \in \{1, \ldots, n\}$, and n is the size of vectors $[\![\boldsymbol{a}]\!], [\![\boldsymbol{c}]\!]$ 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 $[\![\boldsymbol{c}]\!]$ 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 $[\![\boldsymbol{p}]\!] \leftarrow ([\![\boldsymbol{a}]\!] =$ "F''), $[\![\boldsymbol{q}]\!] \leftarrow ([\![\boldsymbol{b}]\!] =$ "$statistics''$), 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

Algorithm 1: Privacy-preserving function **cut** for cutting the dataset according to a given filter that leaks the size of the selected subset n.

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 Obliviously shuffle the value pairs in vectors $([\![\boldsymbol{a}]\!], [\![\boldsymbol{m}]\!])$ into $([\![\boldsymbol{a'}]\!], [\![\boldsymbol{m'}]\!])$
2 $\boldsymbol{s} \leftarrow$ **declassify**$([\![\boldsymbol{m'}]\!])$
3 Form an output vector $[\![\boldsymbol{x}]\!]$ by collecting all $[\![{a'}_i]\!]$ for which $s_i = 1$
4 **return** $[\![\boldsymbol{x}]\!]$

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. On the other hand, some algorithms require that a subset vector containing only the filtered data be built. We use Algorithm 1 for selecting a subset 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 [44].

Lemma 1. *If operations in the PDK are universally composable, then Algorithm 1 leaks only the number of non-zero elements in the input mask filter.*

Proof. In short, due to oblivious shuffle the elements of $\boldsymbol{s}$ are in unknown random order. Hence, it is trivial to simulate the results of the declassify protocol—it suffices to fix n ones in random locations.

More formally, we first define the trusted third party $\mathcal{T}$ in the ideal implementation as follows: $\mathcal{T}$ takes in vectors $[\![\boldsymbol{a}]\!]$ and $[\![\boldsymbol{m}]\!]$ and submits $[\![\boldsymbol{x}]\!]$ to computing parties where $\boldsymbol{x}$ consists of a_i satisfying $m_i = 1$ in random order. In the hybrid world, we must simulate all actions given only the shares of $[\![\boldsymbol{x}]\!]$ corresponding to corrupted computing parties. The number of such shares reveals the number of elements n in the subset. To simulate the outcomes of the oblivious shuffle protocol, we fix n random locations among N cells. To get shares of the elements in the permuted vector $[\![\boldsymbol{a}]\!]$, we insert elements of $[\![\boldsymbol{x}]\!]$ one-by-one into these slots and fill the remaining slots with $[\![0]\!]$. To simulate vector $[\![\boldsymbol{s}]\!]$, we insert $[\![1]\!]$ to the slots corresponding to the locations where elements of $[\![\boldsymbol{x}]\!]$ were inserted and fill the remaining slots with $[\![0]\!]$. After that we open the vector $[\![\boldsymbol{s}]\!]$ to the attacker. As all operations are ideal, computing parties observe no difference.

Also, note that the order of shares $[\![\boldsymbol{x}]\!]$ matches: if corrupted parties behave honestly, their shares of $[\![\boldsymbol{x}]\!]$ are in the same order as specified by $\mathcal{T}$. Consequently, the hybrid protocol is cryptographically secure. The security of the real implementation based on the PDK follows directly form the universal composability of shuffle and declassify operations.

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) [4]. 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 simple tools 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 [35] 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 that leaks the size of the selected subset

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 that hides the size of the selected subset.

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 unwanted or missing values from the data vector. The subset vector is sorted in a privacy-preserving way and 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. Note that Algorithm 2 consists of the **cut** operation followed by a straight line program. Hence, we can use the universal composability property of the PDK together with the **cut** security proof to conclude that Algorithm 2 is cryptographically secure.

Algorithm 3 starts with a call to the function $\mathbf{sort}^*$. This function first sorts the data vector by the data values and then by the mask values (line 1) to ensure that unwanted or 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

Algorithm 4: Privacy-preserving algorithm for finding the frequency table of a data vector that leaks the size of the selected subset.

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 $n \leftarrow \mathbf{size}([\![\boldsymbol{x}]\!])$
3 $k \leftarrow \lceil \log_2(n) + 1 \rceil$
4 $[\![min]\!] \leftarrow \mathbf{min}([\![\boldsymbol{x}]\!])$, $[\![max]\!] \leftarrow \mathbf{max}([\![\boldsymbol{x}]\!])$
5 Securely compute breaks $[\![\boldsymbol{b}]\!]$ according to $[\![min]\!], [\![max]\!]$ and k
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 works as **Q** 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 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 density estimation

The distribution of values of an attribute 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 size n of the returned vector $[\![\boldsymbol{x}]\!]$ is determined by the function **size** on line 2. As we do not hide the sizes of private vectors, this function returns a public value. The number of bins k is publicly computed according to Sturges' formula [52]. Bins are created in a privacy-preserving manner based on the minimum, maximum and k. Finally, on lines 6 - 9, the elements belonging in each bin are counted using secure multi-party 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 $[\![\boldsymbol{a}]\!]$ of size N 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 are 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 Univariate 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 [13], 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 are 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 [30, 31] (MAD). Element $[\![x]\!]$ is considered an outlier in a value vector $[\![\boldsymbol{a}]\!]$ of length n if

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

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.

5 Statistical tests

It is often useful to compare behaviours in two groups, often referred to as 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.

5.1 The principles of statistical testing

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 information we are willing to publish.

Option 1 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

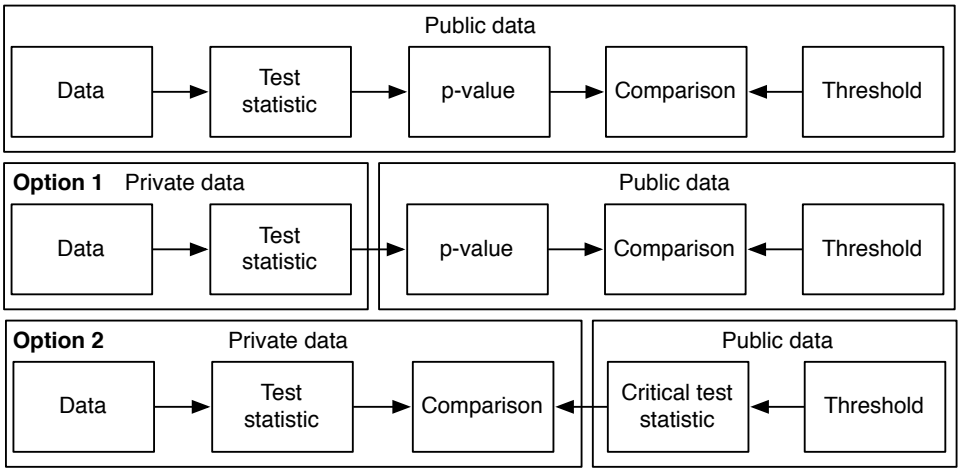

Fig. 3. Statistical testing procedure in the public and private setting

given threshold value. As the function that converts the test statistic to into the p-value is always monotone and depends only on the sizes of case and control groups. Consequently, it can be always inverted if sample sizes are public. Hence, publishing the test statistic is equivalent to revealing the p-value together with the sizes of case and control groups.

However, revealing the sizes of the case and control groups or the raw p-value might sometimes reveal too much information. For this occasion, we propose Option 2, where 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 private test statistic is compared to the critical test statistic in a privacy-preserving manner. The only thing that is published is the decision whether the alternative hypothesis is supported by the data.

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 [39] 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 [39] 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 $[\![\boldsymbol{x}]\!]$ and $[\![\boldsymbol{y}]\!]$ 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

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

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

The t-test provides accurate results only if measurements in the case and control groups follow 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 [34] 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 we need the dataset to retain elements from both groups—cases and controls. On line 1, we combine the two input mask vectors into one. The

Algorithm 5: Privacy-preserving two-sided Mann-Whitney U test that leaks the total size of the case and control group

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 $[\![R_{ca}]\!] \leftarrow \mathbf{sum}([\![\boldsymbol{r}_{ca}]\!])$
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 that leaks the size of the case and control group

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** $[\![w]\!] \leftarrow \mathbf{sum}([\![\boldsymbol{s}]\!] \cdot [\![\boldsymbol{r}]\!])$

function **cut** is the same as before, except 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. It is possible to use this test without the correction which makes it give a stricter bound and might not accept borderline hypotheses, but the algorithm 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 [55] is a paired difference test. Our version, given in Algorithm 6, takes into account Pratt's correction [34] 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 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. As the first operation **cut** is followed by a straight line program in both algorithms, the universal composability property of the PDK together with the security proof of function **cut** is sufficient for concluding cryptographic security of both algorithms.

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. The standard χ^2 test statistic is computed as

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

where f_{ji} is the observed frequency and e_{ji} is the expected frequency of the i-th option and j-th group. As we are working with two populations, we can simplify this formula as

$$\chi^2 = \sum_{i=1}^{k} \frac{(c_i - e_{1i})^2}{e_{1i}} + \frac{(d_i - e_{2i})^2}{e_{2i}} \ ,$$

then the frequencies can be presented as the contingency Table 2.

To give the analyst the possibility to choose, which values will be converted into which option in the contingency table, we introduce the notion of a public codebook **CB**. This matrix essentially holds the possible values of the attribute in the first column, and the option they will belong to, in the second.

	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 2. Contingency table for the standard χ^2 test

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 $\mathbf{CB}$ defining k options from u possible values of the attribute

Result: Contingency table $[\![\mathbf{CT}]\!]$

$[\![x_{ca}]\!] \leftarrow [\![\boldsymbol{a}]\!] \cdot [\![\boldsymbol{ca}]\!]$ and $[\![x_{co}]\!] \leftarrow [\![\boldsymbol{a}]\!] \cdot [\![\boldsymbol{co}]\!]$

Let $[\![\mathbf{CT}]\!]$ be a $2 \times k$ matrix

for $i \in \{1, \ldots, u\}$ **do**

  $[\![b_{ca}]\!] \leftarrow [\![x_{ca}]\!] = \mathbf{CB}_{i,1}$ and $[\![b_{co}]\!] \leftarrow [\![x_{co}]\!] = \mathbf{CB}_{i,1}$

  $j \leftarrow \mathbf{CB}_{i,2}$

  $[\![\mathbf{CT}_{1,j}]\!] \leftarrow [\![\mathbf{CT}_{1,j}]\!] + \mathbf{sum}([\![b_{ca}]\!])$

  $[\![\mathbf{CT}_{2,j}]\!] \leftarrow [\![\mathbf{CT}_{2,j}]\!] + \mathbf{sum}([\![b_{co}]\!])$

end

return $[\![\mathbf{CT}]\!]$

Algorithm 8: Privacy-preserving χ^2 test of independence

Data: Contingency table $[\![\mathbf{C}]\!]$ of size $2 \times k$

Result: The test statistic χ^2

Let $[\![n]\!]$ be the total count of elements

Let $[\![r_1]\!]$ and $[\![r_2]\!]$ be the row subtotals and $[\![p_1]\!], \ldots, [\![p_k]\!]$ be the column subtotals

Let $[\![\mathbf{E}]\!]$ be a table of expected frequencies such that

$[\![\mathbf{E}_{i,j}]\!] \leftarrow \frac{[\![r_i]\!] \cdot [\![p_j]\!]}{n}$, $i \in \{1, 2\}, j \in \{1, \ldots, k\}$

$[\![\chi^2]\!] \leftarrow \sum_{j=1}^{k} \frac{([\![\mathbf{C}_{1,j}]\!] - [\![\mathbf{E}_{1,j}]\!])^2}{[\![\mathbf{E}_{1,j}]\!]} + \frac{([\![\mathbf{C}_{2,j}]\!] - [\![\mathbf{E}_{2,j}]\!])^2}{[\![\mathbf{E_{2,j}}]\!]}$

return $[\![\chi^2]\!]$

Algorithm 7 compiles a contingency table from a data vector, mask vector and the public code book $\mathbf{CB}$. 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 spurious results 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.

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 $[\![c_{(1)}]\!], \ldots, [\![c_{(i_*)}]\!]$. Declare these hypotheses significant.

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, 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 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 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. As the number of revealed zeroes is equal to the number of significant hypotheses, we can directly simulate the published values knowing only the desired outcome. The formal security proof is analogous to the proof of Lemma 1.

6 Predictive modelling

As most prediction models are defined in terms of linear algebra, we have implemented privacy-preserving versions of the following vector and matrix operations: computing the dot product, computing vector length, computing 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 and can be used in linear regression models.

6.1 Linear regression

Linear regression is the most commonly used method for predicting 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 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

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

where the vector of errors $\boldsymbol{\varepsilon} = (\varepsilon_j)$ is assumed to be white Gaussian noise. This list of equations can be compactly written as $\boldsymbol{\varepsilon} = \mathbf{X}\boldsymbol{\beta} - \boldsymbol{y}$. Most methods for linear regression try to minimize the square of residuals

$$\|\boldsymbol{\varepsilon}\|^2 = \|\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}\|^2 \ . \tag{1}$$

This can be done directly or by first converting the minimization task into its equivalent characterization in terms of linear equations:

$$\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} = \mathbf{X}^T\boldsymbol{y} \ . \tag{2}$$

Let us first look at simple linear regression, where the aim of the analysis is to find $\hat{\alpha}$ and $\hat{\beta}$ that fit the approximation $y_i \approx \beta_0 + \beta_1 x_i$ best for all data points. Then the corresponding linear equation (2) can be solved directly:

$$\hat{\beta}_1 = \frac{\mathbf{cov}(\boldsymbol{x}, \boldsymbol{y})}{\mathbf{var}(\boldsymbol{x})}$$
$$\hat{\beta}_0 = \mathbf{mean}(\boldsymbol{y}) - \hat{\beta}_1 \cdot \mathbf{mean}(\boldsymbol{x}) \ .$$

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 $\pi(j)$ be a permutation of indices $j \in \{1, \ldots, n\}$
2 $[\![b]\!] \leftarrow [\![a_{\pi(1)}]\!]$ and $[\![l]\!] \leftarrow \pi(1)$
3 **for** $i \in \{\pi(2), \ldots, \pi(n)\}$ **do**
4 $\quad [\![c]\!] \leftarrow (\left|[\![a_{\pi(i)}]\!]\right| > |[\![b]\!]|)$
5 $\quad [\![b]\!] \leftarrow [\![b]\!] - [\![c]\!] \cdot [\![b]\!] + [\![c]\!] \cdot [\![a_{\pi(i)}]\!]$
6 $\quad [\![l]\!] \leftarrow [\![l]\!] - [\![c]\!] \cdot [\![l]\!] + [\![c]\!] \cdot \pi(i)$
7 **end**
8 **return** $([\![b]\!], [\![l]\!])$

As we have the capability to compute covariance and variance in the privacy-preserving setting, we can also estimate the values of $\hat{\beta}_0$ and $\hat{\beta}_1$ in this setting.

We will look at three different methods for solving the system (2) with more than one explanatory variable: for $k < 4$, we simply invert the matrix by computing determinants. For the general case, we give algorithms for the Gaussian elimination method [48] and LU decomposition [48]. We also describe the conjugate gradient method [33] that directly minimizes the square of residuals (1).

In all 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: $[\![\boldsymbol{b}]\!] = [\![\mathbf{X}]\!]^T [\![\boldsymbol{y}]\!]$.

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 the pivot element. While the Gaussian and LU decomposition algorithms can be used without pivoting, it is not advisable as the algorithms are numerically unstable in the presence of any roundoff errors [48].

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. This means that the indices are traversed in random order during each execution. On line 4, the current maximum element is compared with the element that is being viewed. On lines 5 and 6 the value and its location are determined obliviously. Namely, if the new element was larger, it will be considered as the new maximum element and its location will be recorded based on the same comparison result. For several maximum elements, it returns the location of one of these elements.

Of the two 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.

At the start of the algorithm (line 2), the rows of the input matrix $[\![\mathbf{A}]\!]$ are shuffled along with the elements of the dependent variable vector, that have been copied to $[\![\boldsymbol{x}]\!]$, retaining the relations. On lines 5-13, 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 14, we privately make a note whether any of the pivot elements is 0, and on line 34, we finish the algorithm early if we are dealing with a singular matrix. 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 17 - 22, elements on the pivot line are reduced. Similarly, on lines 23 - 31, elements below the pivot line are reduced. Finally, on lines 36 - 39, backsubstitution is performed to get the values of the coefficients.

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.

Lemma 2. *If operations in the PDK are universally composable, then Algorithm 11 leaks only whether the matrix is singular or not.*

Proof. During the execution of the algorithm, row indices p_i of pivoting elements are declassified to computing parties. For the proof, we must show that pivoting index $\boldsymbol{p} = (p_i)$ leaks no information about the matrix $\mathbf{A}$.

For technical reasons, it is better to represent $\boldsymbol{p}$ in terms of original row labels. For that, we initially label rows of $\mathbf{A}$ after the shuffling with $1, \ldots, n$ and carry these labels through the computations. Hence, p_i is not equal to the declassified output of the **maxLoc** operation. Instead, p_i is the original label of the row after shuffling. For example, let the first maximum element be in row 4. Then $p_i = 4$, the first and fourth rows are interchanged and the matrix is reduced. During the next step, let the maximum element be again in row 4. Now p_i will not be 4 but the original label, i.e. 1. Clearly, both representations are equivalent, as one can compute the outputs of **maxLoc** form $\boldsymbol{p}$ and vice versa.

Let us first compare what happens if we apply Algorithm 11 with and without shuffling the rows (line 1). Let $\mathbf{A}$ and $\mathbf{A}^*$ denote the matrix states throughout the execution of the corresponding algorithms. In the simplest case, all pivoting

Algorithm 11: Privacy-preserving Gaussian elimination with backsubstitution for a matrix equation $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$ that leaks if the matrix is singular

Data: a $k \times k$ matrix $[\![\mathbf{A}]\!]$, a vector $[\![\boldsymbol{b}]\!]$ of k values
Result: Vector $[\![\boldsymbol{x}]\!]$ of coefficients

Let $[\![\boldsymbol{x}]\!]$ be a copy of $[\![\boldsymbol{b}]\!]$
Obliviously shuffle $[\![\mathbf{A}]\!], [\![\boldsymbol{x}]\!]$ retaining the dependencies
Let $[\![c]\!] \leftarrow$ **false** privately store the failure flag during execution
for $i \in \{1, \ldots, k-1\}$ **do**
    $[\![\boldsymbol{m}]\!]$ be a subvector of $[\![\mathbf{A}_{u,v}]\!]$ such that $u \in \{i, \ldots, k\}, v = i$
    $([\![t]\!], [\![irow]\!]) \leftarrow$ **maxLoc**$([\![\boldsymbol{m}]\!])$
    $irow \leftarrow$ **declassify**$([\![irow]\!]) + i$
    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 $[\![x_{irow}]\!]$ and $[\![x_i]\!]$
    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
        $[\![x_i]\!] \leftarrow [\![x_i]\!] \cdot [\![pivinv]\!]$
    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
        $[\![x_m]\!] \leftarrow [\![x_m]\!] - [\![x_i]\!] \cdot [\![\mathbf{A}_{m,i}]\!]$
        $[\![\mathbf{A}_{m,i}]\!] \leftarrow 0$
    end
end
if declassify$([\![c]\!])$ **then**
    return "Singular matrix"
end
$[\![x_k]\!] \leftarrow \dfrac{[\![x_k]\!]}{[\![\mathbf{A}_{k,k}]\!]}$
for $i \in \{k-1, \ldots, 1\}$ **do**
    $[\![x_i]\!] \leftarrow [\![x_i]\!] - \sum_{j=i+2}^{k} [\![\mathbf{A}_{i,j}]\!] \cdot [\![x_j]\!]$
end
return $[\![\boldsymbol{x}]\!]$

steps are deterministic, as there is only one element with maximal absolute value larger than zero for $\boldsymbol{m}$. In this case, we can always obtain $\mathbf{A}^*$ by permuting the rows of $\mathbf{A}$. This claim clearly holds, at the beginning of execution. Now assume that the claim holds at the beginning of the for-cycle (line 4). Then vectors $\boldsymbol{m}$ and $\boldsymbol{m}^*$ will contain the same elements. Consequently, both algorithms must choose the row with the same matrix elements. The following reduction steps of the algorithm use this row to modify remaining rows. As the set of rows to be modified is the same up to the permutation of rows, the reduction step yields the same results up to the row permutation. This completes the induction.

If there are several equal maximal elements then the pivoting index p is not uniquely determined by the matrices $\mathbf{A}$ and $\mathbf{A}^*$. Nevertheless, we can compare the distributions of $\boldsymbol{p}$ and $\boldsymbol{p}^*$. We do this by carefully aligning runs of both algorithms. We can represent such runs by the tree of events where intermediate decision nodes represent choices made by the **maxLoc** algorithm. As **maxLoc** chooses elements with the maximal absolute value with equal probability, each child is chosen uniformly when we reach such a decision node. Note that up to the first equality the claim about matrices $\mathbf{A}$ and $\mathbf{A}^*$ still holds. As the order of children in the decision node does not alter probabilities, we can align event trees so that the children match. As a result, the row with the same element is chosen in matching children and thus matrix $\mathbf{A}^*$ can be still obtained by permuting the rows of $\mathbf{A}^*$. Hence, both runs of the algorithms can be represented with the same event tree such that $\mathbf{A}^*$ in each leaf node can be obtained by permuting the rows of $\mathbf{A}$.

Note that if we assign row labels to $\mathbf{A}$ before shuffling, then the pivoting indices $\boldsymbol{p}$ and $\boldsymbol{p}^*$ will be identical in each leaf node. Consequently, the distributions of pivoting indices will be the same. As the indices are assigned after the shuffling step, we can obtain the distribution of $\boldsymbol{p}$ by taking $\boldsymbol{p}^*$ and applying a random permutation π to all elements, i.e., $p_i = \pi(p_i^*)$. The latter directly implies that $\boldsymbol{p}$ is distributed as a uniform permutation.

The claim holds even if the matrix is singular and thus some $\mathbf{A}_{i,i} = 0$. If the inversion operation returns a fixed value as 0^{-1} all claims presented above still hold. If the inversion operation returns several values with different probabilities, then we must add additional decision nodes to the event tree. The latter, does not change the reasoning, as choices can be matched similarly to before. Hence, we get matching event trees and the claim still holds. For the same reason, the claim holds even if all arithmetical operations are imprecise and probabilistic.

To complete the proof, we must formally define the corresponding simulator construction. The latter is straightforward. We first sample the pivoting index $\boldsymbol{p}$ and then simulate shares to match the execution dictated by $\boldsymbol{p}$. We omit the construction here as it is analogous to the simulator construction in Lemma 1.

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

Algorithm 12: LUDecomp: Privacy-preserving LU decomposition for a symmetric matrix $\mathbf{A}$ that leaks if the matrix is singular

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

Let $[\![c]\!] \leftarrow 0$ be a boolean value
for $i \in \{1, \ldots, k\}$ **do**
  $[\![\boldsymbol{m}]\!]$ be a subvector of $[\![\mathbf{B}_{u,v}]\!]$ such that $u \in \{i, \ldots, k\}, v = i$
  $([\![t]\!], [\![irow]\!]) \leftarrow \mathbf{maxLoc}([\![\boldsymbol{m}]\!])$
  $irow \leftarrow \mathbf{declassify}([\![irow]\!])$
  if $irow \neq i$ **then**
    for $j \in \{1, \ldots, k\}$ **do**
      Exchange elements $[\![\mathbf{B}_{irow,j}]\!]$ and $[\![\mathbf{B}_{i,j}]\!]$
    end
  end
  $[\![c]\!] \leftarrow [\![c]\!] \vee ([\![\mathbf{B}_{i,i}]\!] = 0)$
  $q_i \leftarrow irow$
  $[\![ipiv]\!] \leftarrow [\![\mathbf{B}_{i,i}]\!]^{-1}$
  for $m \in \{i+1, \ldots, k\}$ **do**
    $[\![\mathbf{B}_{m,i}]\!] \leftarrow [\![\mathbf{B}_{m,i}]\!] \cdot [\![ipiv]\!]$
    for $j \in \{i+1, \ldots, k\}$ **do**
      $[\![\mathbf{B}_{m,j}]\!] \leftarrow [\![\mathbf{B}_{m,j}]\!] - [\![\mathbf{B}_{m,i}]\!] \cdot [\![\mathbf{B}_{i,j}]\!]$
    end
  end
end
if $\mathbf{declassify}([\![c]\!])$ **then**
  return "Singular matrix"
end
return $([\![\mathbf{B}]\!], \boldsymbol{q})$

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.

Lemma 3. *If operations in the PDK are universally composable, then Algorithm 12 leaks only whether the matrix is singular or not.*

Algorithm 13: Solving linear regression task $\boldsymbol{y} \approx \mathbf{X}\boldsymbol{\beta}$ using the LU decomposition matrix in a privacy-preserving setting

Data: An $n \times k$ data matrix $[\![\mathbf{X}]\!]$ and an n element response vector $[\![\boldsymbol{y}]\!]$
Result: A vector $[\![\boldsymbol{b}]\!]$ of coefficients

1 Compute correlation matrix $[\![\mathbf{A}]\!] \leftarrow [\![\mathbf{X}]\!]^T [\![\mathbf{X}]\!]$
2 Compute new target vector $[\![\boldsymbol{b}]\!] \leftarrow [\![\mathbf{X}]\!]^T [\![\boldsymbol{y}]\!]$
3 Shuffle $[\![\mathbf{A}]\!], [\![\boldsymbol{b}]\!]$ retaining the dependencies
4 $([\![\mathbf{B}]\!], \boldsymbol{q}) \leftarrow \mathbf{LUDecomp}([\![\mathbf{A}]\!])$
5 Rearrange $[\![\boldsymbol{b}]\!]$ based on permutation $\boldsymbol{q}$
6 **for** $i \in \{2, \ldots, k\}$ **do**
7 $\quad [\![b_i]\!] \leftarrow [\![b_i]\!] - \sum_{j=1}^{i} [\![\mathbf{B}_{i,j}]\!] \cdot [\![b_j]\!]$
8 **end**
9 **for** $i \in \{k, \ldots, 1\}$ **do**
10 $\quad [\![b_i]\!] \leftarrow \left([\![b_i]\!] - \sum_{j=i+1}^{k} [\![\mathbf{B}_{i,j}]\!] \cdot [\![b_j]\!] \right) \cdot [\![\mathbf{B}_{i,i}]\!]^{-1}$
11 **end**
12 **return** $[\![\boldsymbol{b}]\!]$

Proof. This reasoning is the same as for Lemma 2. It is easy to see that both algorithms output the same pivoting vectors, as the sub-matrix used for choosing the pivot row is updated identically in both algorithms.

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 receive the permutation that was done for pivoting purposes during the decomposition phase. Next, on row 5, 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. In the usual setting, 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 6 - 8, forward substitution is performed using the values from the lower triangular matrix. Finally, on rows 9 - 11, backsubstitution is performed using the values from the upper triangular matrix.

For the security consideration, note that Lemma 3 assures that line 4 is universally composable and, thus, the entire algorithm is a straight line program consisting of universally composable operations. As such, it is also secure.

In addition to the methods based on the formulation (2), we decided to look at an iterative algorithm that trie to minimize the quadratic residuals directly to test the difference in performance and accuracy. We chose the conjugate gradient method for this quadratic programming task (1). For a quadratic programming task, the conjugate gradient algorithm is guaranteed to converge in k steps, where

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 $[\![\boldsymbol{x}]\!]^T$

k is the number of columns in the matrix $\mathbf{A}$, provided that all computations are done without errors [1].

As our matrix is symmetric and positive semi-definite, we can use the simplest version of this method with the fixed number of iterations that depends on the number of variables k. Considering that the initial convergence of the conjugate gradient method is rapid during a small number of iterations [1] and that privacy-preserving floating point operations are approximately as imprecise as operations with the float datatype in the normal setting, we decided to use fixed number of iterations in all of our experiments. As the largest number of variables was 10, the number of iterations was fixed to 10. Algorithm 14 shows how to solve our quadratic programming task using the conjugate gradient method. As Algorithm 14 is a straight line program, it is secure by default.

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 [9] as the computation backend. In this setting, no computing party $\mathcal{CP}$ can derive information about intermediate values unless it colludes with another computing party.

This PDK uses secret sharing among three servers to protect the confidentiality of the data and has a wide range of implemented protocols (see Section 2.2).

Figure 4 shows the architecture of the tool and how different SHAREMIND components were used in its implementation.

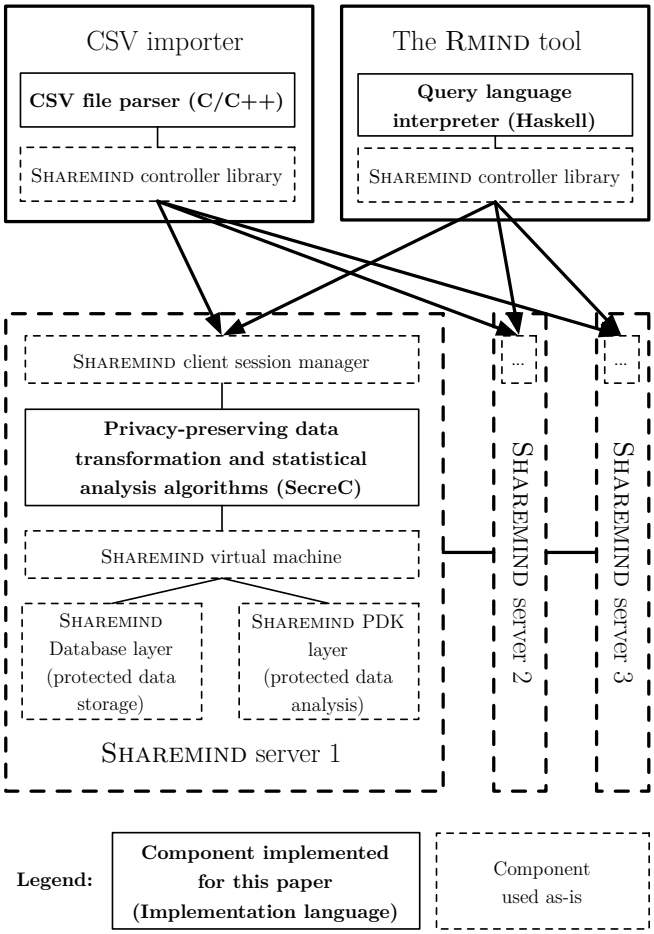

Fig. 4. The architecture of the RMIND tool (servers 2 and 3 are identical to server 1)

We implemented a command line utility for uploading data that can secret-share values from files in CSV-format so that each server gets one share of each value in the input file. These tables can 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 [19] 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 [7] 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 previous intermediate results, it can access them 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 are performed. The three servers shown in our architecture must, therefore, be deployed by independent organizations interested in preserving the privacy of the data. If that assumption holds, the whole RMIND system provides much improved privacy with provable security guarantees 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 are stored in public and private arrays of signed integers, floating point numbers or boolean values. The language also supports public strings for names. We do not give 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 their 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 [8] and [43], respectively.

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

Analyzing private data. Table 3 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.

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

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 3. Statistical operations in RMIND

in [5], we have optimized the implementation and redone all benchmarks for this paper using the new RMIND tool. The benchmarks in [5] were performed on an identical hardware setting, but they are less optimized and use an older version of SHAREMIND.

Tables 4 and 5 show the performance of statistical operations in comparison with earlier work in [5]. 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 2.5.

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.

References

1. Owe Axelsson. Iteration number for the conjugate gradient method. *Mathematics and Computers in Simulation*, 61(3–6):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.

Operation	***Inputs***	***Time*** (RMIND)	***Time*** ([5])
`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 4. 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.

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

3. Dan Bogdanov. *Sharemind: programmable secure computations with practical applications*. PhD thesis, University of Tartu, 2013.
4. Dan Bogdanov, Roman Jagomägis, and Sven Laur. A Universal Toolkit for Cryptographically Secure Privacy-Preserving Data Mining. In Michael Chau, G. Alan Wang, Wei Thoo Yue, and Hsinchun Chen, editors, *Intelligence and Security Informatics - Pacific Asia Workshop, PAISI'12, Kuala Lumpur, Malaysia, May 29, 2012. Proceedings*, volume 7299 of *Lecture Notes in Computer Science*, pages 112–126. Springer, 2012.
5. 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.
6. Dan Bogdanov, Peeter Laud, Sven Laur, and Pille Pullonen. From input private to universally composable secure multi-party computation. In *Proc. of CSF'14*. IEEE Computer Society, 2014.
7. 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.
8. 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*, volume 8788 of *LNCS*, pages 59–74. Springer, 2014.
9. 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.
10. 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.
11. 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.
12. 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.
13. 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.
14. 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.
15. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science. FOCS'01*, pages 136–145. IEEE Computer Society, 2001.
16. 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.

17. 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.
18. 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.
19. 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.
20. 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.
21. Wenliang Du and Mikhail J. Atallah. Privacy-preserving cooperative statistical analysis. In *Proceedings of ACSAC 2001*, pages 102–110, 2001.
22. 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.
23. 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 *LNCS*, pages 1–12. Springer, 2006.
24. Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *Advances in Cryptology (EUROCRYPT 2006)*, volume 4004 of *Lecture Notes in Computer Science*, pages 486–503. Springer Verlag, 2006.
25. 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.
26. 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.
27. Martin Geisler. *Cryptographic Protocols: Theory and Implementation.* PhD thesis, Aarhus University, February 2010.
28. 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.
29. 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.
30. Frank R. Hampel. A general qualitative definition of robustness. *The Annals of Mathematical Statistics*, 42(6):1887–1896, 1971.
31. Frank R. Hampel. The influence curve and its role in robust estimation. *Journal of the American Statistical Association*, 69(346):383–393, June 1974.

32. 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.
33. 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.
34. Myles Hollander and Douglas A Wolfe. *Nonparametric statistical methods*. John Wiley New York, 2nd ed. edition, 1999.
35. Rob J Hyndman and Yanan Fan. Sample quantiles in statistical packages. *The American Statistician*, 50(4):361–365, 1996.
36. Marek Jawurek and Florian Kerschbaum. Fault-tolerant privacy-preserving statistics. In *Privacy Enhancing Technologies*, volume 7384 of *LNCS*, pages 221–238. Springer, 2012.
37. 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.
38. 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/`.
39. Gopal K Kanji. *100 statistical tests*. Sage, 2006.
40. Florian Kerschbaum. Practical privacy-preserving benchmarking. In *Proceedings of IFIP TC-11 SEC 2008*, volume 278, pages 17–31. Springer US, 2008.
41. 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.
42. 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.
43. 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.
44. Sven Laur, Jan Willemson, and Bingsheng Zhang. Round-Efficient Oblivious Database Manipulation. In *Proceedings of ISC 2011*, pages 262–277, 2011.
45. 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.
46. 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.
47. 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.
48. 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.
49. Reimo Rebane. A Feasibility Analysis of Secure Multiparty Computation Deployments. Master's thesis, Institute of Computer Science, University of Tartu, 2012.

50. 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.
51. Nigel P. Smart and Frederik Vercauteren. Fully homomorphic simd operations. *Des. Codes Cryptography*, 71(1):57–81, 2014.
52. Herbert A Sturges. The choice of a class interval. *Journal of the American Statistical Association*, 21(153):65–66, 1926.
53. 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.
54. Latanya Sweeney. k-anonymity: a model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):557–570, 2002.
55. Frank Wilcoxon. Individual Comparisons by Ranking Methods. *Biometrics Bulletin*, 1(6):80–83, 1945.
56. Zhiqiang Yang, Rebecca N. Wright, and Hiranmayee Subramaniam. Experimental analysis of a privacy-preserving scalar product protocol. *Computer Systems Science & Engineering*, 21(1), 2006.
57. Andrew Chi-Chih Yao. Protocols for Secure Computations (Extended Abstract). In *Proceedings of FOCS'82*, pages 160–164. IEEE, 1982.