Exact and Efficient Bayesian Inference for Privacy Risk Quantification*

Rasmus C. Rønneberg[1], Raúl Pardo[2], and Andrzej Wąsowski[2]

[1] Karlsruhe Institute of Technology, Germany
rasmus.ronneberg@kit.edu
[2] IT University of Copenhagen, Denmark
{raup,wasowski}@itu.dk

Abstract. Data analysis has high value both for commercial and research purposes. However, disclosing analysis results may pose severe privacy risk to individuals. Privug is a method to quantify privacy risks of data analytics programs by analyzing their source code. The method uses probability distributions to model attacker knowledge and Bayesian inference to update said knowledge based on observable outputs. Currently, Privug uses Markov Chain Monte Carlo (MCMC) to perform inference, which is a flexible but approximate solution. This paper presents an exact Bayesian inference engine based on multivariate Gaussian distributions to accurately and efficiently quantify privacy risks. The inference engine is implemented for a subset of Python programs that can be modeled as multivariate Gaussian models. We evaluate the method by analyzing privacy risks in programs to release public statistics. The evaluation shows that our method accurately and efficiently analyzes privacy risks, and outperforms existing methods. Furthermore, we demonstrate the use of our engine to analyze the effect of differential privacy in public statistics.

Keywords: Privacy risk analysis · Bayesian inference · Probabilistic Programming

1 Introduction

Data anonymization methods gain legal importance [5] as data collection and analysis are expanding dramatically in data management and statistical research. Yet applying anonymization, or understanding how well a given analytics program hides sensitive information, is non-trivial [20]. Contemporary anonymization algorithms, such as differential privacy [18], require calibration to balance between reducing risks and preserving the utility of data. To assess the risks, data scientists need to assess the flow (leakage) of information from sensitive data fields to the output of analytics.

* Work partially supported by funding from the topic Engineering Secure Systems of the Helmholtz Association (HGF), the KASTEL Security Research Labs and the Danish Villum Foundation through Villum Experiment project No. 0002302.

Measuring the information leakage is a useful technique to quantify how much an attacker may learn about the sensitive information a program processes. Many methods have been proposed in this domain [4,14,15,9,8,29,12,34,32]. Privug is a recent one [32]. It relies on Bayesian inference to quantify privacy risks in data analytics programs. The attacker's knowledge is modeled as a probability distribution over program inputs, and it is then conditioned on the disclosed program outputs. Then, Bayesian probabilistic programming is used to compute the posterior attacker knowledge, i.e., the updated attacker knowledge after observing the outputs in the program. One of the advantages of Privug is that it works on the program source code, and can be extended to compute most information leakage metrics [4]. However, Privug currently relies on approximate Bayesian inference such as Markov Chain Monte Carlo [33]. These methods can be used to analyze arbitrary programs, but may be computationally expensive and may produce imprecise results. In quantifying privacy risks, precision is critical, as underestimation of risks may result in an illegal disclosure of personal information.

We present a new *exact* and *efficient* Bayesian inference engine for Privug targeting attackers modeled by multivariate Gaussian distributions. Even though not all standard program statements can be mapped to operations on Gaussian distributions (and thus not all programs can be analyzed using our new engine), multivariate Gaussian distributions are a good candidate for a semantic domain of attacker's knowledge for several reasons: i) Multivariate Gaussians are closed under many common operations and they can be computed efficiently; ii) The Gaussian distribution is a *maximum entropy* distribution under common conditions [30,27] that allows modeling prior attacker knowledge with minimum assumptions; iii) Gaussians are commonly used in probabilistic modeling of engineering systems to represent uncertainty of measurement.

This work constitutes a new point in the study of expressiveness vs performance in quantification of privacy risks by means of Bayesian inference. Specifically, our contributions are:

1. A probabilistic programming language for exact Bayesian inference using multivariate Gaussians. The language is a subset of Python (Sect. 3).
2. A definition of a sound (Sect. 3.2) Bayesian inference engine (Sect. 3.1) using multivariate Gaussian distributions.
3. A proof-of-concept implementation of the inference engine as a library that can be applied to analyze our subset of Python.
4. An application of the inference engine to a case study of privacy risk quantification in public statistics (Sect. 4), with and without differential privacy [18].
5. An evaluation of the scalability of the inference engine, and a comparison with existing inference methods for privacy risk quantification (Sect. 5). The evaluation shows that our engine can analyze large systems involving thousands of individuals, and also that we greatly outperform existing tools for the programs supported in our language.

The code to reproduce the evaluation and case study in this paper is available at [35]. The proof-of-concept implementation of the exact inference engine is an open source project available at https://github.com/itu-square/gauss-privug.

2 Background

2.1 Privug: A Data Privacy Debugging Method

Let $\mathbb{I}, \mathbb{O}$ denote sets of *inputs* and *outputs*, respectively. We use $\mathcal{D}(\mathbb{I})$ to denote a space of distributions; in this case over inputs. Let $d \in \mathcal{D}(\mathbb{I})$ denote a distribution over inputs, $I \sim \mathcal{D}(\mathbb{I})$ denotes a random variable distributed according to d.

Privug [32] is a method to explore information leakage on data analytics programs. The method combines a probabilistic model of attacker knowledge with the program under analysis to quantify privacy risks. This process is summarized in the following steps:

(1) Prior. We first model the *prior* knowledge of an attacker as a distribution over program inputs. This distribution represents the input values of a program that the attacker finds plausible. For example, consider a program that takes as input a real number A representing the age of an individual ($\mathbb{I} \triangleq \mathbb{R}$). A possible prior knowledge of the attacker could be: $A \sim \mathcal{U}(0, 120)$ (all ages between 0 and 120 are equally likely for the attacker) or $A \sim \mathcal{N}(\mu = 42, \sigma = 2)$ (the attacker believes that the age of 42 and values nearby are most likely ages). We write $P(I)$ for the distribution of prior attacker knowledge.

(2) Probabilistic program interpretation. The second step is to interpret a target program $\pi : \mathbb{I} \to \mathbb{O}$ using the attacker prior knowledge. To this end, we lift the program to run on distributions $\mathcal{D}(\mathbb{I})$ instead of concrete inputs $\mathbb{I}$. This corresponds to the standard lifting to the probability monad [24]; *lift* : $(\mathbb{I} \to \mathbb{O}) \to (\mathcal{D}(\mathbb{I}) \to \mathcal{D}(\mathbb{O}))$. For example, consider the following program that computes the average age of a list of ages (in Python):

```
def average_age(ages: List[float]): return sum(ages)/len(ages)
```

The lifted version of the program is (Python allows retaining the same body):

```
def average_age(ages: Dist[List[float]] ): return sum(ages)/len(ages)
```

where `Dist[List[float]]` denotes a distribution over lists of floats, $\mathcal{D}(\mathbb{R}^n)$. The lifted program yields the distribution $P(O|A)$. In general, the combination of prior attacker knowledge with the lifted program yields a joint distribution on inputs and outputs, $P(O|I)P(I) = P(O, I)$.

(3) Observations. It is possible to analyze privacy risks for concrete outputs of the program. To this end, one may add observations to the probabilistic model. In the average example above, we could check how the knowledge of the attacker changes when the attacker observes that the average is 44. This step yields the posterior distribution $P(I|O = 44)$. In general, the probabilistic lifting of the program π defines a likelihood function on the joint distribution of input and output, $P(E|I, O)$ with some predicate E on the joint distribution.

(4) Posterior Inference. The next step is to apply Bayesian inference to obtain a posterior distribution on the input variables.

$$P(I, O|E) = P(E|I, O)P(O|I)P(I)/P(E) \tag{1}$$

It is usually intractable to compute a symbolic representation of $P(E)$. Therefore, it is not possible to get analytical solutions for the posterior distributions. The current implementation of Privug uses Markov Chain Monte Carlo (MCMC) [33] to tackle this issue. But MCMC methods are approximate and do not always converge. As mentioned above, the subject of this paper is to provide an exact inference method to efficiently and precisely compute the posterior distribution.

(5) Posterior analysis We query the posterior and prior distributions (attacker knowledge) to measure how much the attacker has learned. This can be done by applying different techniques such as computing probability queries, plotting visualizations of probability distributions, computing information theoretic metrics (e.g., entropy or mutual information), and metrics from *quantitative information flow* [4] such as Bayes vulnerability.

For example, Fig. 1 compares the prior and posterior distributions of the average age program above. We analyze the case where the output of the program is 44. The green line shows the prior attacker knowledge on the victim's age $P(A)$ and the blue line the posterior knowledge $P(A|O = 44)$ when observing that the program output is 44. The prior attacker knowledge is $A \sim \mathcal{N}(35, 2)$ for the victim's age and also for the rest. The figure clearly shows that the attacker now believes that higher ages are more plausible. In other words, the attacker prior knowledge has been corrected towards more accurate knowledge on the victim's age.

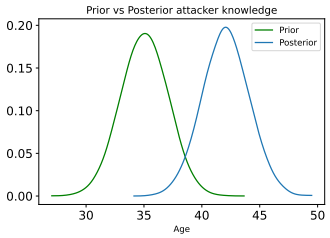

Fig. 1: Prior vs Posterior ages

2.2 Multivariate Gaussian Distributions

In this paper, we use capital Greek letters for matrices, and bold font for column vectors. Small letters a and b are reserved for selecting subvectors (as in $\boldsymbol{\mu}_a$) and pairs of them for selecting submatrices (as in Σ_{ba}). Matrix and vector literals are written in brackets. We write $supp(X)$ for the support of the random variable X.

A *multivariate Gaussian distribution*, denoted $\boldsymbol{X} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$, defines a probabilistic model composed of n normally distributed random variables, $\boldsymbol{X} = [X_1, X_2, \ldots, X_n]^\intercal$. The distribution is parameterized by a vector $\boldsymbol{\mu}$ of n means, and a symmetric $n \times n$ *covariance* matrix Σ, so $\Sigma_{ij} = \text{cov}[X_i, X_j]$ gives the covariance between X_i and X_j, Σ_{kk} gives the variance of X_k. We assume that the covariance matrix is positive definite. The probability density function is:

$$P(\boldsymbol{x}) = ((2\pi)^n|\Sigma|)^{-1/2} \exp\left(-2^{-1}(\boldsymbol{x} - \boldsymbol{\mu})^\intercal \Sigma^{-1}(\boldsymbol{x} - \boldsymbol{\mu})\right) \tag{2}$$

where $|\Sigma|$ denotes the determinant of the matrix Σ. We recall standard properties of multivariate Gaussian distributions [19,28,10].

Theorem 1. *Let* $\begin{bmatrix}\boldsymbol{X}_a\\ \boldsymbol{X}_b\end{bmatrix} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ *with* $\boldsymbol{\mu} = \begin{bmatrix}\boldsymbol{\mu}_a\\ \boldsymbol{\mu}_b\end{bmatrix}$ *and* $\Sigma = \begin{bmatrix}\Sigma_{aa} & \Sigma_{ab}\\ \Sigma_{ba} & \Sigma_{bb}\end{bmatrix}$. *The marginal distributions are* $\boldsymbol{X}_a \sim \mathcal{N}(\boldsymbol{\mu}_a, \Sigma_{aa})$, $\boldsymbol{X}_b \sim \mathcal{N}(\boldsymbol{\mu}_b, \Sigma_{bb})$, *and* $\boldsymbol{X}_i \sim \mathcal{N}(\boldsymbol{\mu}_i, \Sigma_{ii})$ *for* $i = 1 \dots a+b$.

The covariance matrix identifies independent random variables:

Theorem 2. *Let* $[X_1, \dots, X_n]^\intercal \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$, *two marginals* X_i, X_j *with* $i \neq j$ *are* independent *iff* $\Sigma_{ij} = \mathrm{cov}[X_i, X_j] = 0$.

The space of Gaussian distributions is closed under affine transformations:

Theorem 3. *Let* $\boldsymbol{X} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ *and* $\boldsymbol{Y} = \mathrm{A}\boldsymbol{X} + \boldsymbol{b}$ *be an affine transformation with* $\mathrm{A} \in \mathbb{R}^{m \times n}$ *a projection matrix and* $\boldsymbol{b} \in \mathbb{R}^{n \times 1}$ *a column vector. Then,* $\boldsymbol{Y} \sim \mathcal{N}(\mathrm{A}\boldsymbol{\mu} + \boldsymbol{b}, \mathrm{A}\Sigma\mathrm{A}^\intercal)$ *holds.*

We use $Y|X_1, X_2, \dots, X_n$ to denote a random variable Y that is distributed conditionally with respect to $X_1, X_2, \dots, X_n$. Linear combinations of random variables can be used to define hierarchical probabilistic models consisting of dependent random variables, such as Gaussian Bayesian networks [28].

Theorem 4. *Let* $\boldsymbol{X} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ *and* $Y|\boldsymbol{X} \sim \mathcal{N}(\boldsymbol{a}^\intercal \boldsymbol{X} + b, \sigma^2)$, *where* $\boldsymbol{a} \in \mathbb{R}^{n \times 1}$ *is a vector,* $b \in \mathbb{R}$ *and* $\sigma^2 > 0$. *Then* $[\boldsymbol{X}^\intercal, Y]^\intercal \sim \mathcal{N}([\boldsymbol{\mu}^\intercal, \boldsymbol{a}^\intercal\boldsymbol{\mu} + b]^\intercal, \Sigma')$ *with*

$$\Sigma'_{1..n,1..n} = \Sigma,\ \Sigma'_{(n+1)(n+1)} = \sigma^2 + \boldsymbol{a}^\intercal\Sigma\boldsymbol{a},\ \Sigma'_{i(n+1)} = \mathrm{cov}[X_i, Y] = \textstyle\sum_{j=1}^{n} a_j \Sigma_{ij}\ .$$

Example 1. We present an example of a Gaussian Bayesian network [28]. Let $X_1 \sim \mathcal{N}(50, 2)$, $X_2|X_1 \sim \mathcal{N}(2X_1 - 5, 1)$, and $X_3|X_2 \sim \mathcal{N}(X_2 - 10, 4)$. Here the distribution of X_2 is conditioned on X_1, and of X_3 on X_2. The model defines a joint multivariate Gaussian probability distribution $[X_1, X_2, X_3]^\intercal \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$. Theorem 4 allows to compute the mean, variance, and covariance of this joint distribution:

$$\boldsymbol{\mu} = \begin{bmatrix}50\\ 2\cdot\boldsymbol{\mu}_1 - 5\\ 1\cdot\boldsymbol{\mu}_2 - 10\end{bmatrix} = \begin{bmatrix}50\\ 95\\ 85\end{bmatrix},\quad \Sigma = \begin{bmatrix}2 & 2\cdot\Sigma_{11} & 0 + 1\cdot\Sigma_{12}\\ & 1 + 2\cdot\Sigma_{11}\cdot 2 & 0 + 1\cdot\Sigma_{22}\\ & & 4 + 1\cdot\Sigma_{22}\cdot 1\end{bmatrix} = \begin{bmatrix}2 & 4 & 4\\ & 9 & 9\\ & & 13\end{bmatrix}$$

As the matrices are symmetric, we only show the upper-right part. Note that, even tough X_3 does not directly depend on X_1, it still has a non-zero covariance. The reason for this is the indirect dependence through X_2. □

We use *conditioning* to model observations on values of random variables.

Theorem 5. *Let* $\boldsymbol{X} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ *be split into two sub-vectors so that*

$$\boldsymbol{X} = \begin{bmatrix}\boldsymbol{X}_a\\ \boldsymbol{X}_b\end{bmatrix},\ \boldsymbol{\mu} = \begin{bmatrix}\boldsymbol{\mu}_a\\ \boldsymbol{\mu}_b\end{bmatrix},\ \Sigma = \begin{bmatrix}\Sigma_{aa} & \Sigma_{ab}\\ \Sigma_{ba} & \Sigma_{bb}\end{bmatrix}\ \textit{and}\ \boldsymbol{x}_b \in \mathit{supp}(\boldsymbol{X}_b).$$

The conditioned distribution is $\boldsymbol{X}_a|(\boldsymbol{X}_b = \boldsymbol{x}_b) \sim \mathcal{N}(\boldsymbol{\mu}', \Sigma')$ *with* $\boldsymbol{\mu}' = \boldsymbol{\mu}_a + \Sigma_{ab}\Sigma_{bb}^{-}(\boldsymbol{x}_b - \boldsymbol{\mu}_b)$ *and* $\Sigma' = \Sigma_{aa} - \Sigma_{ab}\Sigma_{bb}^{-}\Sigma_{ba}$, *where* Σ^{-} *is the generalized inverse.*

Example 2. Consider the multivariate distribution of Example 1. We condition X_3 to be 85. By Thm. 5 the posterior of $X_1, X_2 | X_3 = 85$ is $\mathcal{N}(\boldsymbol{\mu}', \Sigma')$ with

$$\boldsymbol{\mu}' = \begin{bmatrix}50\\95\end{bmatrix} + \begin{bmatrix}4\\9\end{bmatrix} [13]^{-}(85-85) = \begin{bmatrix}50\\95\end{bmatrix} \text{ and } \Sigma' = \begin{bmatrix}2 & 4\\4 & 9\end{bmatrix} - \begin{bmatrix}4\\9\end{bmatrix} [13]^{-} [4\ 9] = \begin{bmatrix}10/13 & 16/13\\16/13 & 36/13\end{bmatrix} \quad \square$$

3 Exact Inference Engine for Privug

Our inference engine is an interpreter of a probabilistic programming language that corresponds to a subset of Python. We include variable assignments, bounded for-loops, binary operators, sequencing, *probabilistic assignments*, and *observations* (conditioning). Let $v_r \in \mathbb{R}$ be real values, $x, y, z, \ldots$ denote (deterministic) variables, $X, Y, Z, \ldots$ be (Gaussian) random variables, and $\boldsymbol{X}$ a vector of random variables. Let $\oplus \in \{+, -, *, /\}$. The syntax of well-formed programs is generated by the rule p below.

$$\begin{array}{rrcl}
\text{(Expressions)} & e & ::= & v_r \mid x \mid e \oplus e \\
\text{(Distributions)} & d & ::= & \texttt{Normal}(e, e) \mid \texttt{Normal}(e * X + e, e) \\
\text{(Statements)} & s & ::= & X = d \mid X = Y \oplus e \mid X = Y + Z \mid \texttt{condition}(X, e) \mid \\
 & & & x = e \mid s\texttt{;}\ s \mid \texttt{for}\ x\ \texttt{in range}\ v_r\ s \\
\text{(Programs)} & p & ::= & s\texttt{; return}\ \boldsymbol{X}
\end{array}$$

We admit (e) constants expressions, references to deterministic program variables, and binary operations. Two ways of defining normal distributions (d) are supported: an independent Gaussian distribution, or a linear transformation of random variables. Statements (s) are: probabilistic assignments (a normal, a transformed distribution, a sum of two random variables), an observation (conditioning), deterministic assignment, sequencing, and a limited for-loop. We define no expressions over random variables, only statements, to simplify introduction of changes to the state (the probabilistic model) in the semantics for each subexpression (Sect. 3.1). The for-loops are only a convenience construct for repetitive statements. A program (p) terminates returning a random variable (`return`). The distribution of the returned variable is the marginal of the posterior joint probability distribution that we want to reason about.

Although the language appears restrictive, we show in Sect. 4 that it can be used in realistic case studies; e.g., for the study of privacy risks in database reconstruction attacks using public statistics. Furthermore, this syntax ensures soundness and termination (Sect. 3.2) of a highly scalable (Sect. 5) inference engine.

3.1 Semantics

The formal semantics is defined in the small-step style, over terms of multivariate Gaussian distributions (Sect. 2.2). It provides a sound and efficient inference engine to track attacker knowledge in Privug (cf. Sect. 2.1).

A *state* $\mathcal{S}$ is a tuple $\langle \boldsymbol{\mu}, \Sigma, \sigma \rangle$. The first two elements define a multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ over n random variables. Let $\mathcal{V}$ denote the set

of deterministic variables, $\sigma : \mathcal{V} \to \mathbb{R}$ maps variables to values. We use $\Sigma_{[X,X]}$ to denote the variance of marginal variable X, and $\Sigma_{[X,.]}$, $\Sigma_{[.,X]}$ to denote the covariance vectors of X with other variables in the state's multivariate Gaussian distribution. Similarly, $\Sigma_{[\boldsymbol{X},\boldsymbol{X}]}$, and $\Sigma_{[\boldsymbol{X},\boldsymbol{Y}]}$ denote the covariance matrix of the sub-vector $\boldsymbol{X}$ of a multivariate Gaussian, and the covariance matrix between sub-vectors $\boldsymbol{X}, \boldsymbol{Y}$ of a multivariate Gaussian, respectively. We use μ_X to denote the mean of X, and $\boldsymbol{\mu_X}$ for the mean vector of $\boldsymbol{X}$.

Definition 1 (Semantics). *The semantics is given by the relations* $\to_e$: $e \times \mathcal{S} \to \mathbb{R}$, $\to_s$: $s \times \mathcal{S} \to \mathcal{S}$ *and* $\to_p$: $p \times \mathcal{S} \to \mathbb{R}^{n\times 1} \times \mathbb{R}^{n\times n}$ *for expressions* e, *statements* s, *and programs* p, *respectively, as defined in Fig. 2.*

The rules for expressions, (deterministic) assignments, sequence of statements, and for-loops are standard, and they do not change the state's multivariate Gaussian distribution. We omit their details. Programs finish with a `return` instruction. It returns the mean-vector $\boldsymbol{\mu}_a$ and covariance matrix Σ_{aa} of the specified sub-vector of the state's multivariate Gaussian. In what follows, we focus on the rules manipulating the state's multivariate Gaussian distribution.

There are two types of probabilistic assignments: independent and linearly dependent. In both cases the multivariate Gaussian distribution in the state is extended with a new variable, and consequently the mean vector ($\boldsymbol{\mu}$) and covariance matrix (Σ) increase their dimension. Independent assignments (P-Asg-Ind) add to the mean vector the mean of the distribution. The covariance matrix is also updated with two $\mathbf{0}$ vectors indicating the new variable is not correlated with existing variables, and the variable's variance is added to the diagonal of the matrix. Dependent assignments (P-Asg-Dep) add to the mean vector a mean computed as a linear combination with the mean of the dependent random variable Y. The covariance matrix is extended with two vectors computed from the covariance of the dependent variable with other variables, $\Sigma_{[Y,.]}$, $\Sigma_{[.,Y]}$. This is because the new variable depends on Y and consequently on all the variables that Y depends on. The variance of the new variable is added to the diagonal of the matrix as a linear combination with the variance of Y.

Example 3. Consider the program $X = \texttt{Normal}(15, 2)$; $Y = \texttt{Normal}(20, 1)$; $Z = \texttt{Normal}(2X, 1)$. The first assignment results in state $\boldsymbol{\mu} = [15]$ and $\Sigma = [2]$. The second assignment updates the state into,

$$\boldsymbol{\mu}' = \begin{bmatrix} 15 \\ 20 \end{bmatrix} \quad \Sigma' = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$$

Note the zeros in the covariance coefficients as these variables are independent. Finally, the last probabilistic statement results in

$$\boldsymbol{\mu}'' = \begin{bmatrix} 15 \\ 20 \\ 2 \cdot 15 \end{bmatrix} = \begin{bmatrix} 15 \\ 20 \\ 30 \end{bmatrix} \quad \Sigma'' = \begin{bmatrix} 2 & 0 & 2 \cdot 2 \\ 0 & 1 & 2 \cdot 0 \\ 2 \cdot 2 & 2 \cdot 0 & 2^2 \cdot 2 + 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 4 \\ 0 & 1 & 0 \\ 4 & 0 & 9 \end{bmatrix}$$

Here we observe that the covariance between Z and X is updated with a non-zero value due to the dependency between variables, but the coefficients of Y, Z are 0 as these variable remain independent. □

$$\text{(V-Exp)}\ \frac{\sigma(x) = c}{\langle x, \mathcal{S}\rangle \to_e c} \qquad \text{(O-Exp)}\ \frac{\langle e_0, \mathcal{S}\rangle \to_e c_0 \qquad \langle e_1, \mathcal{S}\rangle \to_e c_1}{\langle e_0 \oplus e_1, \mathcal{S}\rangle \to_e c_0 \oplus c_1}$$

$$\text{(C-Exp)}\ \frac{}{\langle c, \mathcal{S}\rangle \to_e c} \qquad \text{(D-Asg)}\ \frac{e \to_e c}{\langle x = e, \mathcal{S}\rangle \to_s \langle \boldsymbol{\mu}, \Sigma, \sigma[x \mapsto c]\rangle}$$

$$\text{(P-Asg-Ind)}\ \frac{\langle e_i, \mathcal{S}\rangle \to_e c_i \text{ for } i = 1, 2 \;\; \boldsymbol{\mu}' = \begin{bmatrix}\boldsymbol{\mu} \\ c_1\end{bmatrix} \;\; \Sigma' = \begin{bmatrix}\Sigma' & \mathbf{0} \\ \mathbf{0} & c_2\end{bmatrix}}{\langle X = \texttt{Normal}(e_1, e_2), \mathcal{S}\rangle \to_s \langle \boldsymbol{\mu}', \Sigma', \sigma\rangle}\ c_2 > 0$$

$$\text{(P-Asg-Dep)}\ \frac{\begin{array}{c}\langle e_i, \mathcal{S}\rangle \to_e c_i \text{ for } i = 1..3 \\ \boldsymbol{\mu}' = \begin{bmatrix}\boldsymbol{\mu} \\ c_1\mu_Y + c_2\end{bmatrix} \quad \Sigma' = \begin{bmatrix}\Sigma & c_1\Sigma_{[.,Y]} \\ c_1\Sigma_{[Y,.]} & c_1^2\Sigma_{[Y,Y]} + c_3\end{bmatrix}\end{array}}{\langle X = \texttt{Normal}(e_1 * Y + e_2, e_3), \mathcal{S}\rangle \to_s \langle \boldsymbol{\mu}', \Sigma', \sigma\rangle}\ c_3 > 0$$

$$\text{(P-Cond)}\ \frac{\begin{array}{c}\langle e, \mathcal{S}\rangle \to_e c \quad \boldsymbol{\mu} = \begin{bmatrix}\boldsymbol{\mu}_a \\ \mu_X\end{bmatrix} \;\; \Sigma = \begin{bmatrix}\Sigma_a & \Sigma_{[.,X]} \\ \Sigma_{[X,.]} & \Sigma_{[X,X]}\end{bmatrix} \\ \boldsymbol{\mu}' = \boldsymbol{\mu}_a + (c - \mu_X)/\Sigma_{[X,X]}\Sigma_{[.,X]} \quad \Sigma' = \Sigma_a - 1/\Sigma_{[X,X]}\Sigma_{[.,X]}\Sigma_{[X,.]}\end{array}}{\langle \texttt{condition}(X, e), \langle \boldsymbol{\mu}, \Sigma, \sigma\rangle\rangle \to_s \langle \boldsymbol{\mu}', \Sigma', \sigma\rangle}$$

$$\text{(Seq)}\ \frac{\langle s_0, \mathcal{S}\rangle \to_s \mathcal{S}'' \;\; \langle s_1, \mathcal{S}''\rangle \to_s \mathcal{S}'}{\langle s_0;\ s_1, \mathcal{S}\rangle \to_s \mathcal{S}'} \qquad \text{(Ret)}\ \frac{\langle s, \mathcal{S}\rangle \to_s \langle \begin{bmatrix}\boldsymbol{\mu_X} \\ \boldsymbol{\mu_Y}\end{bmatrix}, \begin{bmatrix}\Sigma_{[\boldsymbol{X},\boldsymbol{X}]} & \Sigma_{[\boldsymbol{X},\boldsymbol{Y}]} \\ \Sigma_{[\boldsymbol{Y},\boldsymbol{X}]} & \Sigma_{[\boldsymbol{Y},\boldsymbol{Y}]}\end{bmatrix}, \sigma\rangle}{\langle s; \texttt{return}\, \boldsymbol{X}, \mathcal{S}\rangle \to_p (\boldsymbol{\mu_X}, \Sigma_{[\boldsymbol{X},\boldsymbol{X}]})}$$

$$\text{(For-B)}\ \frac{v_r \le 0}{\langle \texttt{for}\, x\, \texttt{in}\, \texttt{range}\, v_r\ \ s, \mathcal{S}\rangle \to_s \mathcal{S}} \qquad \text{(For-I)}\ \frac{\begin{array}{c}v_r > 0 \quad \langle s', \mathcal{S}\rangle \to_s \mathcal{S}' \\ s' = s; x = x{+}1; \texttt{for}\, x\, \texttt{in}\, \texttt{range}\, v_r{-}1\ s\end{array}}{\langle \texttt{for}\, x\, \texttt{in}\, \texttt{range}\, v_r\ \ s, \mathcal{S}\rangle \to_s \mathcal{S}'}$$

$$\text{(P-Sum)}\ \frac{\boldsymbol{\mu}' = \begin{bmatrix}\boldsymbol{\mu} \\ \mu_Y + \mu_Z\end{bmatrix} \quad \Sigma' = \begin{bmatrix}\Sigma & \Sigma_{[.,Y]} + \Sigma_{[.,Z]} \\ \Sigma_{[Y,.]} + \Sigma_{[Z,.]} & \Sigma_{[Y,Y]} + \Sigma_{[Z,Z]} + \Sigma_{[Y,Z]} + \Sigma_{[Z,Y]}\end{bmatrix}}{\langle X = Y + Z, \mathcal{S}\rangle \to_s \langle \boldsymbol{\mu}', \Sigma', \sigma\rangle}$$

$$\frac{\begin{array}{c}\text{(P-Op-PM)} \\ \oplus \in \{+, -\} \quad \langle e, \mathcal{S}\rangle \to_e c \\ \boldsymbol{\mu}' = \begin{bmatrix}\boldsymbol{\mu} \\ \mu_Y \oplus c\end{bmatrix} \quad \Sigma' = \begin{bmatrix}\Sigma & \Sigma_{[.,Y]} \\ \Sigma_{[Y,.]} & \Sigma_{[Y,Y]}\end{bmatrix}\end{array}}{\langle X = Y \oplus e, \mathcal{S}\rangle \to_s \langle \boldsymbol{\mu}', \Sigma', \sigma\rangle} \qquad \frac{\begin{array}{c}\text{(P-Op-MD)} \\ \oplus \in \{*, /\} \quad \langle e, \mathcal{S}\rangle \to_e c \\ \boldsymbol{\mu}' = \begin{bmatrix}\boldsymbol{\mu} \\ \mu_Y \oplus c\end{bmatrix} \quad \Sigma' = \begin{bmatrix}\Sigma & c \oplus \Sigma_{[.,Y]} \\ c \oplus \Sigma_{[Y,.]} & c^2 \oplus \Sigma_{[Y,Y]}\end{bmatrix}\end{array}}{\langle X = Y \oplus e, \mathcal{S}\rangle \to_s \langle \boldsymbol{\mu}', \Sigma', \sigma\rangle}$$

Fig. 2: Operational Semantics rules; $\mathcal{S}$ stands for a tuple $\langle \boldsymbol{\mu}, \Sigma, \sigma\rangle$.

Two rules (P-Op-PM) and (P-Op-MD) define binary operations between random variables and values. These rules always produce a random variable that is added to the state's multivariate Gaussian. This is why the statement is combined with an assignment.

When a value is added/subtracted to a random variable (P-Op-PM), a new random variable is added with its mean updated accordingly. The new variable inherits the variance an covariances of Y. For multiplication and division (P-Op-MD), the mean is updated as before, but also the variance of the random variable, and the covariances with the dependent random variables.

Example 4. Consider the program $X = \texttt{Normal}(1,1);\ Y = X + 2;\ Z = Y * 2$. After the first statement we have the state $\boldsymbol{\mu} = [1]$ and $\Sigma = [1]$. The second statement updates the state such that,

$$\boldsymbol{\mu}' = \begin{bmatrix} 1 \\ 3 \end{bmatrix} \quad \Sigma' = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

The last statements updates the state into

$$\boldsymbol{\mu}'' = \begin{bmatrix} 1 \\ 3 \\ 6 \end{bmatrix} \quad \Sigma'' = \begin{bmatrix} 1 & 1 & 2 \cdot 1 \\ 1 & 1 & 2 \cdot 1 \\ 2 \cdot 1 & 2 \cdot 1 & 2^2 \cdot 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 2 \\ 1 & 1 & 2 \\ 2 & 2 & 4 \end{bmatrix}$$

Sum of random variables. The sum of two Gaussian random variables (P-Sum) adds a new random variable to the multivariate Gaussian. The mean of the new random variable is the sum of the means of the operands. The covariance of the resulting random variable is the sum of the covariances of the operands with other variables, i.e., the new variable depends on all the variables that the operands depend on. The variance is the sum of the variances of the operands, and the covariances of the operands.

Example 5. Consider the program $X = \texttt{Normal}(15,2);\ Y = \texttt{Normal}(2,1);\ Z = X + Y$. After the second assignment we have

$$\boldsymbol{\mu}' = \begin{bmatrix} 15 \\ 2 \end{bmatrix} \quad \Sigma' = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$$

Thus, the third assignment updates the state's multivariate Gaussian into

$$\boldsymbol{\mu}'' = \begin{bmatrix} 15 \\ 2 \\ 15+2 \end{bmatrix} = \begin{bmatrix} 15 \\ 2 \\ 17 \end{bmatrix} \quad \Sigma'' = \begin{bmatrix} 2 & 0 & 2+0 \\ 0 & 1 & 0+1 \\ 2+0 & 0+1 & 2+1+0+0 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 2 \\ 0 & 1 & 1 \\ 2 & 1 & 3 \end{bmatrix}$$

Conditions. For conditioning (P-Cond), we use Thm. 5 introduced in Sect. 2.2. As a result of conditioning, the observed variable is removed from the mean vector and covariance matrix. Note that, despite P-Cond applying to the newest random variable, we may perform an affine transformation using a permutation matrix that swaps the order of random variables. Thus, conditions may refer to any variable in the multivariate Gaussian.

Example 6. Let $\boldsymbol{\mu}''$, and Σ'' be those in the final state of the program in Example 5. Suppose that we extend the program with the statement $\texttt{condition}(Z, 1)$. The resulting multivariate Gaussian is updated as

$$\boldsymbol{\mu}''' = \begin{bmatrix}15\\2\end{bmatrix} + \frac{1-17}{3} \cdot \begin{bmatrix}2\\1\end{bmatrix} = \begin{bmatrix}15\\2\end{bmatrix} + \begin{bmatrix}-32/3\\-16/3\end{bmatrix} = \begin{bmatrix}13/3\\-10/3\end{bmatrix}$$

$$\Sigma''' = \begin{bmatrix}2 & 0\\0 & 1\end{bmatrix} - \frac{1}{3} \cdot \begin{bmatrix}2\\1\end{bmatrix} \cdot \begin{bmatrix}2 & 1\end{bmatrix} = \begin{bmatrix}2 & 0\\0 & 1\end{bmatrix} - \frac{1}{3} \cdot \begin{bmatrix}4 & 2\\2 & 1\end{bmatrix} = \begin{bmatrix}8/3 & -2/3\\-2/3 & 2/3\end{bmatrix}$$

Recall that covariances may be negative as the covariance matrix is positive definite (cf. Sect. 2.2).

3.2 Soundness and Termination

In what follows, we show that the semantics rules in Fig. 2 are *sound*, and that the inference engine always *terminates* for well-formed programs.

We establish *soundness* of our engine by ensuring that all program statements perform a closed-form transformation on the state's multivariate Gaussian distribution. Lemmas (1-5) assert the soundness of each of the rules in $\rightarrow_s$ (cf. Fig. 2). For example, below we show the proof of the lemma for sum of random variables (i.e., P-Sum) whose soundness is based on the affine transformation property of multivariate Gaussian distributions (cf. Thm. 3). The lemma asserts that the distribution resulting from executing the program statement is a well-formed multivariate Gaussian and also that the newly introduced variable is distributed as the sum of the operands. We refer interested readers to the extended version of the paper [36] for the proofs of the remaining lemmas. Again, we omit the soundness details of deterministic statements and expressions as they are standard. The soundness of the Ret rule follows from lemmas (1-5) and Thm. 1.

Lemma 1 (Sum random vars.). *Let* $[X_1, X_2, \ldots]^\intercal \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$. *For all states* $\mathcal{S} = \langle \boldsymbol{\mu}, \Sigma, \sigma \rangle$, *if* $\langle Y = X_i + X_j, \mathcal{S} \rangle \rightarrow_s \langle \boldsymbol{\mu}', \Sigma', \sigma \rangle$ *and* $[X_1, X_2, \ldots, X_i + X_j]^\intercal \sim \mathcal{N}(\boldsymbol{\mu}'', \Sigma'')$, *then* $[X_1, X_2, \ldots, Y]^\intercal \sim \mathcal{N}(\boldsymbol{\mu}', \Sigma')$ *and* $\mathcal{N}(\boldsymbol{\mu}', \Sigma') = \mathcal{N}(\boldsymbol{\mu}'', \Sigma'')$.

Proof. Let A be a $m \times n$ projection matrix where $\mathrm{A}_n = I_n$ where I_n is a $n \times n$ identity matrix, $\mathrm{A}_{n+1}[i] = \mathrm{A}_{n+1}[j] = 1$ and $\mathrm{A}_{n+1}[k] = 0$ for $i, j \neq k$. Let $\boldsymbol{b} = \mathbf{0}$. Then, by matrix multiplication $[X_1, X_2, \ldots, X_i + X_j]^\intercal = \mathrm{A}\boldsymbol{X} + \boldsymbol{b}$. By Thm. 3, $[X_1, X_2, \ldots, X_i + X_j]^\intercal \sim \mathcal{N}(\mathrm{A}\boldsymbol{\mu} + \boldsymbol{b}, \mathrm{A}\Sigma\mathrm{A}^\intercal)$. By matrix multiplication

$$\begin{aligned}\mathrm{A}\boldsymbol{\mu} + \boldsymbol{b} &= \begin{bmatrix}\boldsymbol{\mu}\\ \mu_{X_i} + \mu_{X_j}\end{bmatrix}\\ \mathrm{A}\Sigma\mathrm{A}^\intercal &= \begin{bmatrix}\Sigma & \Sigma_{[.,X_i]} + \Sigma_{[.,X_j]}\\ \Sigma_{[X_i,.]} + \Sigma_{[X_j,.]} & \Sigma_{[X_i,X_i]} + \Sigma_{[X_j,X_j]} + \Sigma_{[X_i,X_j]} + \Sigma_{[X_j,X_i]}\end{bmatrix}\end{aligned} \tag{3}$$

By definition 1 and eq. (3) we have that $\boldsymbol{\mu}' = \mathrm{A}\boldsymbol{\mu} + \boldsymbol{b}$ and $\Sigma' = \mathrm{A}\Sigma\mathrm{A}^\intercal$. Thus, $[X_1, X_2, \ldots, Y]^\intercal \sim \mathcal{N}(\boldsymbol{\mu}', \Sigma')$ and $\mathcal{N}(\boldsymbol{\mu}', \Sigma') = \mathcal{N}(\mathrm{A}\boldsymbol{\mu} + \boldsymbol{b}, \mathrm{A}\Sigma\mathrm{A}^\intercal)$. □

Lemma 2 (Independent assignment). *Let* $[X_1, X_2, \ldots]^\intercal \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$*. For all states* $\mathcal{S} = \langle \boldsymbol{\mu}, \Sigma, \sigma \rangle$*, if* $\langle Y = \texttt{Normal}(e_1, e_2), \mathcal{S} \rangle \rightarrow_s \langle \boldsymbol{\mu}', \Sigma', \sigma \rangle$ *and* $\langle \mathcal{S}, e_i \rangle \rightarrow_e c_i$*, then* $[X_1, X_2, \ldots, Y]^\intercal \sim \mathcal{N}(\boldsymbol{\mu}', \Sigma')$ *and* $Y \sim \mathcal{N}(c_1, c_2)$ *and* Y, X_i *are independent.*

Lemma 3 (Dependent assignments). *Let* $[X_1, X_2, \ldots]^\intercal \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$*. For all states* $\mathcal{S} = \langle \boldsymbol{\mu}, \Sigma, \sigma \rangle$*, if* $\langle Y = \texttt{Normal}(e_1 * X_i + e_2, e_3), \mathcal{S} \rangle \rightarrow_s \langle \boldsymbol{\mu}', \Sigma', \sigma \rangle$ *and* $\langle \mathcal{S}, e_i \rangle \rightarrow_e c_i$*, then* $[X_1, X_2, \ldots, Y]^\intercal \sim \mathcal{N}(\boldsymbol{\mu}', \Sigma')$ *and* $Y \mid X_i \sim \mathcal{N}(c_1 X_i + c_2, c_3)$*.*

Lemma 4 (Binary operations with values). *Let* $[X_1, X_2, \ldots]^\intercal \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ *and* $\oplus \in \{+, -, *, /\}$*. For all states* $\mathcal{S} = \langle \boldsymbol{\mu}, \Sigma, \sigma \rangle$*, if* $\langle Y = X_i \oplus e, \mathcal{S} \rangle \rangle \rightarrow_s \langle \boldsymbol{\mu}', \Sigma', \sigma \rangle$ *and* $\langle \mathcal{S}, e \rangle \rightarrow_e c$ *and* $[X_1, X_2, \ldots, X_i \oplus c]^\intercal \sim \mathcal{N}(\boldsymbol{\mu}'', \Sigma'')$*, then* $[X_1, X_2, \ldots, Y]^\intercal \sim \mathcal{N}(\boldsymbol{\mu}', \Sigma')$ *and* $\mathcal{N}(\boldsymbol{\mu}', \Sigma') = \mathcal{N}(\boldsymbol{\mu}'', \Sigma'')$*.*

Lemma 5 (Conditioning). *Let* $[\boldsymbol{X}_a^\intercal, Y]^\intercal \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$*. For all states* $\mathcal{S} = \langle \boldsymbol{\mu}, \Sigma, \sigma \rangle$*, if* $\langle \mathcal{S}, e \rangle \rightarrow_e c$ *and* $\langle \texttt{condition}(Y, c), \mathcal{S} \rangle \rightarrow_s \langle \boldsymbol{\mu}', \Sigma', \sigma \rangle$*, then* $\boldsymbol{X}' \sim \mathcal{N}(\boldsymbol{\mu}', \Sigma')$ *and* $\boldsymbol{X}' = \boldsymbol{X}_a \mid Y = c$*.*

Readers familiar with semantics of probabilistic programs will note that these lemmas define the cases of a pushfoward measure semantics *à la* Kozen [7] on the program statements in our language. A standard induction on the structure of well-formed programs establishes soundness of our inference engine.

We also establish termination of our inference engine.

Lemma 6 (Termination). *Given a well-formed program, the process of computing the resulting multivariate Gaussian always terminates.*

Proof. Well-formed programs are unbounded but finite sequences of program statements. Thus, to prove termination, it suffices to prove that each program statement is evaluated in finite time. Expressions (V-Exp, C-Exp, O-Exp) and deterministic statements (D-Asg, Seq) can be resolved in constant time. For-loops (For-B,For-I) are bounded and can be unfolded in linear time in the number of iterations. Probabilistic assignments (P-Asg-Ind, P-Asg-Dep) extend the mean vector with one element and the covariance matrix with a row and column vectors. Both operations can be performed in linear time in the number of variables of the program. Binary operations between random variables and values (P-Op-PM, P-Op-MD), summation of random variables (P-Sum) and conditioning (P-Cond) are computed as a sequence of matrix multiplication operations. All these operations can be computed in polynomial time in the size of the program state, which is never larger than quadratic in the number of variables in the program. Return (Ret) performs a lookup in the mean vector and covariance matrix, which can be computed in constant time. □

Our inference engine not only terminates for all well-formed programs, but, most importantly, it is *very* efficient. In Sect. 5, we study the scalability of the inference engine, and we show that it can efficiently analyze systems with thousands of random variables. Moreover, we show that our method scales much better than existing tools for the family of programs captured by our language.

Age group	Males		Females	
21-30	500	480	470	410
	460	430	420	450
	490	510	460	410
	440	480	510	310
	520	410	370	440
31-40	550	410	450	500
	490	580	520	530
	530	420	510	600
	590	400	620	390
	680	510	550	390
41-50	600	540	590	640
	640	590	540	580
	580	620	740	540
	340	510	140	830
	620	660	540	740
51-60	700	680	690	680
	740	640	720	780
	590	650	680	580
	770	630	590	730
	540	840	640	980

Age group	Males		Females	
21-30	$\mathcal{N}(480, 100)$	$\mathcal{N}(490, 100)$	$\mathcal{N}(450, 100)$	$\mathcal{N}(430, 100)$
	$\mathcal{N}(440, 100)$	$\mathcal{N}(420, 100)$	$\mathcal{N}(410, 100)$	$\mathcal{N}(430, 100)$
	$\mathcal{N}(490, 100)$	$\mathcal{N}(490, 100)$	$\mathcal{N}(470, 100)$	$\mathcal{N}(400, 100)$
	$\mathcal{N}(520, 100)$	$\mathcal{N}(490, 100)$	$\mathcal{N}(490, 100)$	$\mathcal{N}(330, 100)$
	$\mathcal{N}(470, 100)$	$\mathcal{N}(400, 100)$	$\mathcal{N}(350, 100)$	$\mathcal{N}(400, 100)$
31-40	$\mathcal{N}(500, 100)$	$\mathcal{N}(410, 100)$	$\mathcal{N}(400, 100)$	$\mathcal{N}(450, 100)$
	$\mathcal{N}(470, 100)$	$\mathcal{N}(490, 100)$	$\mathcal{N}(490, 100)$	$\mathcal{N}(480, 100)$
	$\mathcal{N}(500, 100)$	$\mathcal{N}(410, 100)$	$\mathcal{N}(500, 100)$	$\mathcal{N}(550, 100)$
	$\mathcal{N}(540, 100)$	$\mathcal{N}(410, 100)$	$\mathcal{N}(590, 100)$	$\mathcal{N}(350, 100)$
	$\mathcal{N}(500, 100)$	$\mathcal{N}(400, 100)$	$\mathcal{N}(510, 100)$	$\mathcal{N}(360, 100)$
41-50	$\mathcal{N}(580, 100)$	$\mathcal{N}(530, 100)$	$\mathcal{N}(550, 100)$	$\mathcal{N}(490, 100)$
	$\mathcal{N}(590, 100)$	$\mathcal{N}(510, 100)$	$\mathcal{N}(520, 100)$	$\mathcal{N}(500, 100)$
	$\mathcal{N}(560, 100)$	$\mathcal{N}(590, 100)$	$\mathcal{N}(650, 100)$	$\mathcal{N}(480, 100)$
	$\mathcal{N}(280, 100)$	$\mathcal{N}(500, 100)$	$\mathcal{N}(150, 100)$	$\mathcal{N}(790, 100)$
	$\mathcal{N}(580, 100)$	$\mathcal{N}(600, 100)$	$\mathcal{N}(510, 100)$	$\mathcal{N}(700, 100)$
51-60	$\mathcal{N}(680, 100)$	$\mathcal{N}(570, 100)$	$\mathcal{N}(680, 100)$	$\mathcal{N}(670, 100)$
	$\mathcal{N}(620, 100)$	$\mathcal{N}(610, 100)$	$\mathcal{N}(690, 100)$	$\mathcal{N}(700, 100)$
	$\mathcal{N}(600, 100)$	$\mathcal{N}(570, 100)$	$\mathcal{N}(630, 100)$	$\mathcal{N}(570, 100)$
	$\mathcal{N}(700, 100)$	$\mathcal{N}(600, 100)$	$\mathcal{N}(500, 100)$	$\mathcal{N}(670, 100)$
	$\mathcal{N}(520, 100)$	$\mathcal{N}(770, 100)$	$\mathcal{N}(600, 100)$	$\mathcal{N}(770, 100)$

Table 1: *Left*: Incomes pr. year in DKK for different age groups and genders. The numbers have been scaled down by a factor of 1000. *Right*: Priors used in the experiments. The means are scaled down by a factor of 1000.

4 Case Study: Privacy Risks in Public Statistics

We analyze a program computing statistics on a database containing incomes for different genders and age groups. The purpose of this case study is to demonstrate the applicability of our approach in a real-life example. Average incomes are available through public national statistics banks [1,2,3], which makes information available to attackers. Leakage of private information and database reconstruction attacks are known issues (e.g., in US census data [21]). We use our inference engine to quantify the increase of attacker knowledge, as she gradually obtains statistics from a database. We also analyze a differentially private [18] mechanism in this setting. The case study uses a small database, but in Sect. 5 we show that our inference engine scales to databases with thousands of individuals.

Releasing public statistics. Consider a data analyst that releases average statistics on population income for different age groups and genders. An attacker with access to the statistics attempts to learn the income of an individual in the database. We consider the synthetic data shown in Tbl. 1 (left). The table shows the income for 40 individuals in different age groups and genders. We consider 3 different cases. *Case* 1) the attacker obtains the average income for males in the age group 21-30. *Case* 2) the attacker also obtains the average income for all people in the age group 21-30. *Case* 3) the attacker also obtains the average income for all males. In all cases the attacker attempts to learn the income of the first male in database in the age group 21-30. We note that the observations made by the attacker are independent, so it does not matter in which order the attacker obtains the observations.

To understand the privacy risks for these 3 cases we use the Privug method described in Section 2. In the following code snippet we show how to model the steps for case 1 using our inference engine. We model the program using our syntax (step 2).

```
1 def agg():
2     male_21_30 = [Normal(480_000, 100), ....]
3     male_21_30_total = male_21_30[0] + male_21_30[1] ...
4     male_21_30_average = male_21_total/10
5     condition("male_21_30_average", 472_000)
6     return male_21_30[0]
```

The array in line 2 contains the priors of the income for the individuals in the database—Tbl. 1 (right) shows the complete list—denoted as $P(I_i)$. They represent the possible incomes that the attacker considers possible before making any observations (step 1). The victim is the first male in the 21-30 age group, $P(I_1)$. In lines 2-3, we compute the average income of this each group, which defines $P(O|I_i)$. In line 5, we add the attacker observation in the condition statement (step 3). Finally, in line 6 we return the posterior distribution of the victim $P(I_1|O)$, which represents the updated attacker knowledge (step 4). Note that, for brevity, we omit the repetitive parts of the code. Also, arrays are syntactic sugar. We refer interested readers to [35] for the complete source code.

Figure 3 shows how attacker knowledge is updated in the 3 cases above, and how close it is to real victim data (vertical line). We plot the prior attacker knowledge $P(I_1)$, and for each case we plot the posterior distribution after conditioning on the output $P(I_1|O)$. The left plot shows the updated attacker knowledge without differential privacy. As the plot shows, the attacker knowledge gets closer to the actual income when obtaining more information. The most accurate attacker knowledge is case 3 where the attacker obtains several average statistics.

Releasing public statistics with differential privacy. Given the above results, the data analyst decides to use a differentially private mechanism [18] to protect the individuals' privacy. Differential Privacy (DP) is used in realistic settings for the release of public statistics. Notably, it was used in the 2020 US Census as a result of privacy issues in previous US Census editions [21]. Intuitively, if the privacy protection mechanism satisfies differential privacy, then the impact of an individual on the output of the program is negligible. More precisely, differential privacy states that: a randomized mechanism $\mathcal{M} : \mathbb{I} \to \mathbb{O}$ is (ϵ, δ)-differentially private if for all $\mathcal{O} \subseteq \mathbb{O}$, and neighboring inputs $i_1, i_2 \in \mathbb{I}$, the following holds

$$P(\mathcal{M}(i_1) \in \mathcal{O}) \leq \exp(\epsilon) P(\mathcal{M}(i_2) \in \mathcal{O}) + \delta.$$

The neighboring relation between inputs depends on the input domain ($\mathbb{I}$). For instance, when it applies to datasets of n natural numbers, $\mathbb{N}^n$, it is usually defined as the first norm $||i_x - i_y||_1$. The parameter ϵ is often referred to as the *privacy parameter*, and it is used to specify the required level of privacy. The parameter δ is the probability of failure. This parameter relaxes the definition of differential privacy. It is used to specify the probability that pure differential

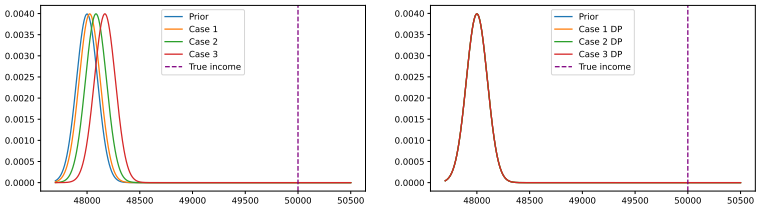

Fig. 3: Updated attacker knowledge after adding observations. Incomes are scaled down by a factor of 10. *Left*: Public stats. *Right*: Public stats with DP.

privacy (i.e., with $\delta = 0$) does not hold. This parameter may be used to, e.g., enable high utility gains while keeping a good level of privacy. Both ϵ and δ are often determined empirically [17].

We analyze a differentially private mechanism for the 3 cases presented above. To this end, we apply the Gaussian mechanism [18], which adds Gaussian noise to the observable output (o) as $o + \mathcal{N}(0, \sigma^2)$. The parameter σ^2 is calculated as follows: $\sigma^2 = 2\Delta^2 \log(1.25/\delta)/\epsilon^2$. The sensitivity $\Delta \in \mathbb{R}$ denotes how much o changes if computed in two datasets differing in at most 1 entry. In our setting, it is $\Delta = (max_{income} - min_{income})/size_{DB}$. We set $\delta = 1/size_{DB}^2$—as usual for this query [18]. We set ϵ to 0.9—this is an arbitrary value, but it is common to use values < 1 in practice [17]. Adding Gaussian noise is proven to satisfy (ϵ, δ)-differential privacy [18]. We remark that our method can be used to determine the values of ϵ and δ that satisfy high level privacy requirements. For instance, privacy requirements specified as probability queries for a given individual or using quantitative information flow metrics [32]. The program implementing the Gaussian mechanism is shown in the following listing

```
1 def agg_dp():
2     ...
3     noise = Normal(0, 1442533240)
4     male_21_30_average_dp = male_21_30_average + noise
5     condition("male_21_30_average_dp", 472_000)
6     return male_21_30[0]
```

We only show lines that change namely: line 3 where the noise distribution is defined, and line 6 where we add the noise to the output. The variance σ^2 of the noise distribution is calculated using the equation above.

The right plot in Fig. 3 shows the updated attacker knowledge in the 3 cases. We observe a decrease in privacy risks when using differential privacy; as the change in attacker knowledge is insignificant for all cases. The plot shows that the impact of the victim's data on the released statistics is minuscule compared to the non-differentially private version of the output.

Information leakage metrics. In addition to inspecting the distributions of attacker knowledge in Fig. 3, we show for demonstration how to compute two

	KL divergence			Mutual information		
	Case 1	Case 2	Case 3	Case 1	Case 2	Case 3
Public stats	312.762	3621.29	14374.13	4.17e-04	6.25e-05	7.81e-06
Diff. Priv.	1.55e-12	1.72e-12	1.81e-12	5.00e-12	3.14e-13	2.16e-14

Table 2: *Left*: KL-divergence between prior and posterior attacker knowledge on secret. *Right*: Mutual information between secret and output random variables.

metrics for information leakage: KL-divergence and mutual information [16]. Let $[P, Q]^\intercal \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$. KL-divergence is $KL(P, Q) = \log_2(\Sigma_{[Q,Q]}^{1/2}/\Sigma_{[P,P]}^{1/2}) + (\Sigma_{[P,P]} + (\mu_P - \mu_Q)^2)/2\Sigma_{[Q,Q]} - 1/2$, and mutual information is $I(P, Q) = 1/2 \log_2(\Sigma_{[P,P]}\Sigma_{[Q,Q]}/|\Sigma|)$.

Table 2 shows the results. The left shows the KL-divergence between prior and posterior attacker knowledge on the secret. Intuitively, this is commonly understood as *information gain* [11]. We observe an increase in information gain from case 1 to 3 (both with and without differential privacy). However, with differential privacy, information gain is virtually 0 for all cases. Tbl. 2 (right) shows mutual information between attacker knowledge on the secret and the output. When mutual information between two random variables is 0, it means that the variables are independent. Thus, a value of mutual information close to 0 indicates that the amount of information shared between secret and output is low. We observe that mutual information decreases from case 1 to 3 (both with and without differential privacy). This is due to the output containing information for a larger set of individuals (which minimizes the effect of the secret on the output). As expected, mutual information is lower with differential privacy. Admittedly, these metrics are hard to interpret in practice, but we remark that more important than the concrete values is their relative distance—it provides a quantitative mean to compare information leakage in different settings.

5 Scalability Evaluation

We evaluate the scalability of our exact inference engine proof-of-concept implementation. The scalability of Bayesian inference engines mainly depends on the number of random variables. Thus, we consider two synthetic benchmark programs with increasing number of variables. The first computes the sum over an increasing number of variables $O = \sum_{i=1}^{n} X_i$. We choose this benchmark as it was originally used to measure the scalability of Privug [32]. We compare the scalability of our engine to Privug MCMC using the NUTS [26] sampler and the exact inference engine PSI [23]—PSI is the leading inference engine supporting the features of our language (cf. Sect. 6). We instruct NUTS to draw 10000 samples in 2 chains—this number of samples produces an accurate posterior in this benchmark, see [32]. The second program performs the same computation

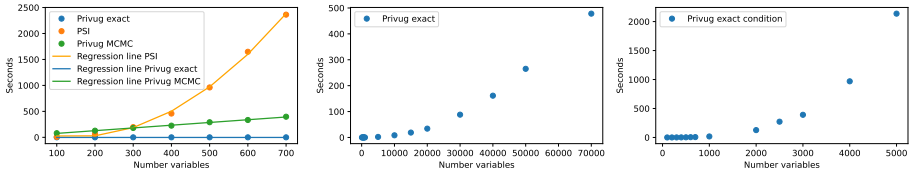

Fig. 4: Execution time for our engine (privug-exact), Privug NUTS (privug-mcmc) and PSI. *Left*: privug-mcmc, privug-exact, and PSI on $O = \sum_i^n X_i$. *Middle*: privug-exact on $O = \sum_i^n X_i$. *Right*: privug-exact on $\texttt{condition}(O, c)$.

but adds a condition statement $\texttt{condition}(O, c)$. The purpose is to evaluate the scalability of our engine in more realistic settings for the case study in Sect. 4. The evaluation run on a 4x2.80GHz cores machine with 16 GB RAM.

Figure 4 (left) shows the measured times for the first program. Execution time does not increase significantly when going from 100 to 700 variables using our engine. On the other hand, PSI takes approximately 40 minutes when summing 700 variables. Most notably, our engine greatly outperforms PSI—in this experiment, it was more than 40.000 times faster than PSI for 700 variables. Privug MCMC exhibits better results, but our engine scales better. As the number of variables increases, we observe a bigger gap between our engine and Privug MCMC—with our engine 6 times faster for 700. It is noteworthy that our exact engine outperforms an approximate inference method.

Figure 4 (middle,right) focus on the scalability for larger systems. The middle plot, shows that our engine can handle the first program with 70000 variables more efficiently that PSI for 700. Figure 4 (right) mimics the case study (Sect. 4). We observe that the condition statement notably degrades the performance of our engine. However, the running time for 5000 individuals is less than 40min. We omitted PSI in this benchmark as conditioning would only decrease its performance, and the previous experiment showed its lower scalability w.r.t. our engine. Privug MCMC is also omitted as a fair comparison requires determining the number of samples to draw to obtain an accurate posterior.

6 Related Work

The majority of existing methods to estimate privacy risks use sampling based techniques [32,34,12,15,14,13]. In [32], Privug made use of MCMC algorithms to perform Bayesian inference, e.g., *Metropolis-Hastings* or *Hamiltonian Monte Carlo* [6,25,26]. Other sampling based methods target specific quantitative information flow metrics [4]—these metrics are supported by Privug [32], and hence by our engine. LeakWatch/Leakiest [15,14] use program samples to estimate mutual information between secret inputs and public outputs. Cherubin et al. and Romanelli et al. [12,34], use machine learning to compute metrics from the g-leakage family [4]. These methods treat programs as black-boxes, so they can analyze any program, as opposed to our method that targets a subset of Python

programs. However, their accuracy guarantees are proven in the limit, i.e., assuming an infinite size sample. In practice, samples are finite and it is often difficult to ensure that results are accurate; specially for programs with large number of variables (such as the ones in Sect. 5). On the contrary, our inference engine produces exact results. This is crucial as a under-approximations could miss important privacy breaches. Furthermore, the scalability evaluation shows that the inference engine scales better than MCMC-based Privug, which is one of the most scalable methods for this type of systems [32].

There exist several works that use exact inference in the context of privacy risk analysis. SPIRE [29] uses the exact inference engine PSI [22,23] to model attacker knowledge and synthesize privacy enforcers. PSI computes a symbolic representation of the joint probability distribution of a given program. It can handle continuous and discrete random variables. It targets a more expressive programming language than the subset of Python that our engine supports. However, PSI scales poorly compared to our engine for programs that our engine supports (cf. Sect. 5). Hakaru [31] and SPPL [37] are exact inference engines—not used for privacy risk analysis. We did not consider them in our evaluation because they do not handle some features of our language. Hakaru cannot handle conditioning probability-zero events (as in lemma 5). SPPL does not support linear combination and sum of Gaussians (as in lemmas 3, 1). QUAIL [9] computes mutual information between input and output variables. It performs forward state exploration of a program to construct a Markov chain, which is then used to compute mutual information. QUAIL works on discrete random variables. Instead, our inference engine works on Gaussian (continuous) random variables and computes the posterior distribution that can be used to compute mutual information (cf. Sect. 4) and other quantitative information flow metrics [32,4].

Stein and Staton proposed a Gaussian-based semantics to study exact conditioning through the lens of category theory [38]. They do not study the use of the semantics for privacy risks quantification on a subset of Python programs, or evaluate the efficiency of the semantics.

7 Conclusion

We have presented an exact Bayesian inference engine for quantifying privacy risks in a subset of Python. We have proven that our inference engine is sound. We have presented an application of our engine to analyze privacy risks on public statistics; a realistic case study for national statistics agencies where privacy risks analysis is crucial. We have also analyzed the impact of differential privacy on data release. In the scalability evaluation, we have shown that our engine can analyze systems with thousands of random variables, and that it greatly outperforms existing tools. All in all, this work provides a new point in the study of expressiveness vs performance. Future work includes adapting our engine with underlying probabilistic models that capture more Python program statements, for instance Gaussian mixtures or the exponential family of probability distributions.

References

1. Statistics Denmark. https://www.dst.dk/en, accessed: 2023-06-23
2. Statistics New Zealand. https://www.stats.govt.nz/, accessed: 2023-06-23
3. US Census Bureau. https://www.census.gov/, accessed: 2023-06-23
4. Alvim, M., Chatzikokolakis, K., McIver, A., Morgan, C., Palamidessi, C., Smith, G.: The Science of Quantitative Information Flow. Springer (2020)
5. Article 29 Data Protection Working Party: Opinion 05/2014 on Anonymisation Techniques (2014), http://www.pdpjournals.com/docs/88197.pdf
6. Avi Pfeffer: Practical probabilistic programming. Manning Publications Co. (2016)
7. Barthe, G., Katoen, J.P., Silva, A. (eds.): Foundations of Probabilistic Programming. Cambridge University Press (2020)
8. Biondi, F., Kawamoto, Y., Legay, A., Traonouez, L.: Hybrid statistical estimation of mutual information and its application to information flow. Formal Aspects Comput. **31**(2), 165–206 (2019)
9. Biondi, F., Legay, A., Traonouez, L., Wasowski, A.: QUAIL: A quantitative security analyzer for imperative code. In: CAV'13. pp. 702–707. Springer Berlin Heidelberg (2013)
10. Bishop, C.M.: Pattern recognition and machine learning. Information science and statistics, Springer, New York (2006)
11. Burnham, K.P., Anderson, D.R.: Model Selection and Multimodel Inference: A Practical Information-Theoretic Approach. Springer-Verlag New York (2002)
12. Cherubin, G., Chatzikokolakis, K., Palamidessi, C.: F-BLEAU: fast black-box leakage estimation. In: SP'19. pp. 835–852. IEEE (2019)
13. Chothia, T., Guha, A.: A statistical test for information leaks using continuous mutual information. In: CSF'11. pp. 177–190. IEEE (2011)
14. Chothia, T., Kawamoto, Y., Novakovic, C.: A tool for estimating information leakage. In: CAV'13. LNCS, vol. 8044, pp. 690–695. Springer (2013)
15. Chothia, T., Kawamoto, Y., Novakovic, C.: Leakwatch: Estimating information leakage from Java programs. In: ESORICS'14. LNCS, vol. 8713. Springer (2014)
16. Cover, T.M., Thomas, J.A.: Elements of information theory (2. ed.). Wiley (2006)
17. Dwork, C., Kohli, N., Mulligan, D.: Differential privacy in practice: Expose your epsilons! Journal of Privacy and Confidentiality **9**(2) (2019)
18. Dwork, C., Roth, A.: The algorithmic foundations of differential privacy. Found. Trends Theor. Comput. Sci. **9**(3-4), 211–407 (2014)
19. Eaton, M.: Multivariate Statistics: A Vector Space Approach. Lecture notes-monograph series, Institute of Mathematical Statistics (2007)
20. Elliot, M., Mackey, E., O'Hara, K., Tudor, C.: The Anonymisation Decision - Making Framework. UKAN, University of Manchester (2016)
21. Garfinkel, S.L., Abowd, J.M., Martindale, C.: Understanding database reconstruction attacks on public data. Commun. ACM **62**(3), 46–53 (2019)
22. Gehr, T., Misailovic, S., Vechev, M.T.: PSI: exact symbolic inference for probabilistic programs. In: CAV'16. LNCS, vol. 9779, pp. 62–83 (2016)
23. Gehr, T., Steffen, S., Vechev, M.: λPSI: exact inference for higher-order probabilistic programs. In: PLDI'20. pp. 883–897. ACM (2020)
24. Gordon, A.D., Henzinger, T.A., Nori, A.V., Rajamani, S.K.: Probabilistic programming. In: FOSE'14. pp. 167–181. ACM (2014)
25. Greenberg, S.C.E.: Understanding the Metropolis-Hastings Algorithm p. 10
26. Homan, M.D., Gelman, A.: The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo. J. Mach. Learn. Res. **15**(1), 1593–1623 (jan 2014)

27. Jaynes, E.T.: Probability theory: The logic of science. Cambridge University Press, Cambridge (2003)
28. Koller, D., Friedman, N.: Probabilistic Graphical Models - Principles and Techniques. MIT Press (2009)
29. Kucera, M., Tsankov, P., Gehr, T., Guarnieri, M., Vechev, M.T.: Synthesis of probabilistic privacy enforcement. In: CCS'17. pp. 391–408. ACM (2017)
30. McElreath, R.: Statistical rethinking: A Bayesian course with examples in R and Stan. CRC press (2020)
31. Narayanan, P., Carette, J., Romano, W., Shan, C., Zinkov, R.: Probabilistic inference by program transformation in hakaru (system description). In: FLOPS'16. LNCS, vol. 9613, pp. 62–79. Springer (2016)
32. Pardo, R., Rafnsson, W., Probst, C.W., Wasowski, A.: Privug: Using probabilistic programming for quantifying leakage in privacy risk analysis. In: ESORICS'21. LNCS, vol. 12973. Springer (2021)
33. Robert, C.P., George Casella: Monte Carlo Statistical Methods. Springer (2004)
34. Romanelli, M., Chatzikokolakis, K., Palamidessi, C., Piantanida, P.: Estimating g-leakage via machine learning. In: CCS'20. ACM (2020)
35. Rønneberg, R.C., Pardo, R., Wąsowski, A.: Exact and Efficient Bayesian Inference for Privacy Risk Quantification (Accompanying Artifact). https://www.doi.org/10.5281/zenodo.8173905
36. Rønneberg, R.C., Pardo, R., Wąsowski, A.: Exact and efficient bayesian inference for privacy risk quantification (extended version). https://doi.org/10.48550/arXiv.2308.16700 (2023)
37. Saad, F.A., Rinard, M.C., Mansinghka, V.K.: SPPL: Probabilistic programming with fast exact symbolic inference. In: PLDI'21. p. 804–819. ACM (2021)
38. Stein, D., Staton, S.: Compositional semantics for probabilistic programs with exact conditioning. In: LICS'21. pp. 1–13. IEEE (2021)