Roseman Labs

Technical sheet

Version 1.1

2 September 2024

Roseman Labs

Table of contents

Table of contents 3

Introduction 5

Multi-party computation and Secret Sharing 6

- Secret-Shared Data Tables 7
- Security model 8

Software components 9

- Engine 9
- Protocols 10
- Crandas (Python package) 10
- Platform 12
- Account authentication 13
- Key generation 14

Data flow 14

- Input module 14
- Summary 17
- Authorization 18
- Script signing 18
- Disclosure control 19

Architecture & deployment 20

- Components 20
- Routing 20
- Infra-as-code 21

Logging 21
Quality and certification 22
Certifications 22
Penetration tests 23
Risk and compliance 24

Introduction

Roseman Labs enables you to encrypt, link and analyze multiple data sets, while safeguarding the privacy and commercial sensitivity of the underlying data. You can combine information from several organizations, run your analyses on the aggregated records, and generate new insights – all without ever being able to view other participants' input. You get the insights you need, while the data stays protected.

Our software employs a cryptographic technology called Multi Party Computation that encrypts all data from beginning to end. This means data owners always stay in control of how their data is processed, enhancing privacy compliance through data minimization and proportionality.

Through the ease of a familiar Python interface, you can enjoy 50+ ready to use functionalities, ranging from basic operations to machine learning and regular expressions. These features unlock previously inaccessible information without compromising data privacy, offering more detail into statistics including time efficiency, product effectiveness, cost savings, resource allocation, and risk analysis.

Here we illustrate the components of which the software consists, the technology on which it is based and the way in which the data is processed. This document is meant as an elaborate explanation of the Roseman Labs software. It is aimed at technical readers assessing the product with respect to its functionality and security guarantees, and how these are scoped and assessed.

Roseman Labs

Multi-party computation and Secret Sharing

Multi-party computation (MPC) is a privacy-enhancing technology[1] that allows strong privacy guarantees regarding collaborations on sensitive data. MPC has a strong theoretical foundation in academic research. Currently, the MPC implementation of Roseman Labs is based on the BGW[2] (Ben-Or, Goldwasser, Widgerson) protocol with Shamir secret-sharing.

When a computation is performed by means of MPC, the cleartext data is first encrypted and distributed (via secret sharing) over multiple compute servers (later also referred to as *nodes* [of Roseman Labs' *engine*]), in such a way that this cleartext data, even during computation, is never disclosed to individual compute servers (hence is neither present on disk nor present in the memory of individual compute servers).

We apply MPC according to the "outsourcing model", in which the organizations that provide the input data ("input parties") do not run the compute servers themselves, but instead delegate this to a set of "compute parties" (typically three). The input parties need to trust those compute parties not to collude and to run our software as-is.

In Roseman Labs' SaaS offering, Roseman Labs provides the three compute parties, by internally hosting each compute server at a different third-party cloud provider, where each such server is administered by a separate Roseman Labs employee. To safeguard that no employee has access to more than one compute server, we apply segregation of duties (see also "Software components" > "Engine").

Our software is designed such that only the results of pre-approved analyses on the source data are disclosed to the analysts. In this way, our implementation of MPC can

---

[1] For more information about privacy enhancing technologies in general and how MPC relates to these, see also https://support.rosemanlabs.com/data-collaboration-and-pets and https://support.rosemanlabs.com/comparison-of-privacy-enhancing-technologies-and-mpc

[2] https://dl.acm.org/doi/10.1145/62212.62213

help to conform to principles laid out in privacy regulations such as the GDPR, specifically purpose binding and data minimization[3].

Secret-Shared Data Tables

Our software is aimed at processing tabular data, which means that data is organized in tables. Each table consists of a list of columns with varying data types (integer, string, etc). Within a single column, every value has the same datatype.

When a table is uploaded to the Roseman Labs engine, each value v in that dataset is transformed into a list of three randomly chosen Shamir secret shares (v_1,v_2,v_3). These shares v_i have the property that an individual share v_i does not reveal any information about v, and that v can be uniquely reconstructed from either of the pairs (v_1,v_2), (v_1,v_3), (v_2,v_3). These shares are then each transferred to a different compute server[4], such that the structure of the original table is present in each of the compute servers, but where the values from the original table have been replaced by their corresponding Shamir shares. On such secret-shared tables various types of operations and computations can be performed, such as joining, filtering and arithmetic calculations[5]. The result of such an operation is stored in another secret-shared table, which can only be revealed with explicit approval.

[3] For more information on how MPC can help to conform to GDPR principles, see also: https://support.rosemanlabs.com/is-it-legally-allowed-to-use-sensitive-data
[4] For more information on how this is securely done, we refer you to the paragraph "Data flow" > "Input module".
[5] For an overview of all operations currently supported, see https://docs.rosemanlabs.com

Security model

Our software implementation conforms to the “honest majority, passive security” security model. In a three-server setup, this means that the system is secure against a scenario in which an attacker observes the state (i.e., the contents of memory and disk) of not more than a single compute server, under the assumption that the administrators of the compute servers run the software as is (passive security).
When a computation is performed, no single compute party / compute server administrator learns any information about the inputs, outputs and intermediate values that are produced, apart from the metadata that is explicitly deemed public. This metadata includes:

- Operation that is performed
- Table names
- Table shapes (number of rows and columns)
- Column names
- Column data types
- Computation time
- Memory use

We always adhere to this security model when implementing and optimizing our protocols. This implies, for example, that the running time of our multiparty computation protocols will never depend on secret information. Hence, an attacker cannot infer any secret information about the input data from measuring the computation time.

Data integrity is guaranteed as long as the server software (the code running on each of the compute servers) and the cache files that the servers store on disk are not tampered with. This is safeguarded by our server-administration access control policies.

Intermediate results that are produced as part of the execution of a computation are after a while automatically removed from cache[6]. In addition to this, data providers can always remove data that they uploaded earlier via our online platform (see also "Software components" > "Platform"). In this way, we guarantee data minimization.

Software components

Here we outline the components of our software and how these interact with one another. For each project, a separate deployment is made available to which data can be uploaded. This prevents that datasets can be processed in an unrelated project, further supporting purpose limitation. A Roseman Labs deployment involves each of the following:

Engine

The engine constitutes the core of the product. The engine consists of three nodes (the compute servers), labeled *node0*, *node1*, *node2,* which collaboratively store and compute on secret shared datasets that are uploaded to them. An engine can run in authorized mode, which means that it will only perform operations that have been approved by the pre-defined set of approvers for that given environment (for more information, see "Data flow" > "Authorization"). Sensitive data may only be uploaded to an environment running in authorized mode. We call an environment in which the engine does not run in authorized mode a "design environment", as it can be used for designing an analysis on fake dummy data.

To safeguard segregation of duties, each of the MPC nodes is deployed at a different European cloud provider (currently Fuga, OVH, Scaleway) and maintained by a separate admin team. Each admin team is screened and contractually bound to only

[6] See also: https://docs.rosemanlabs.com/latest/guide/99-tips.html#computed-objects-might-be-removed-from-cache

managing their designated node. A Roseman Labs admin team cannot inspect customer data as it remains encrypted during the entire lifecycle. A data transfer agreement is not required as the data centres are located and headquartered within the European Economic Area.

Protocols

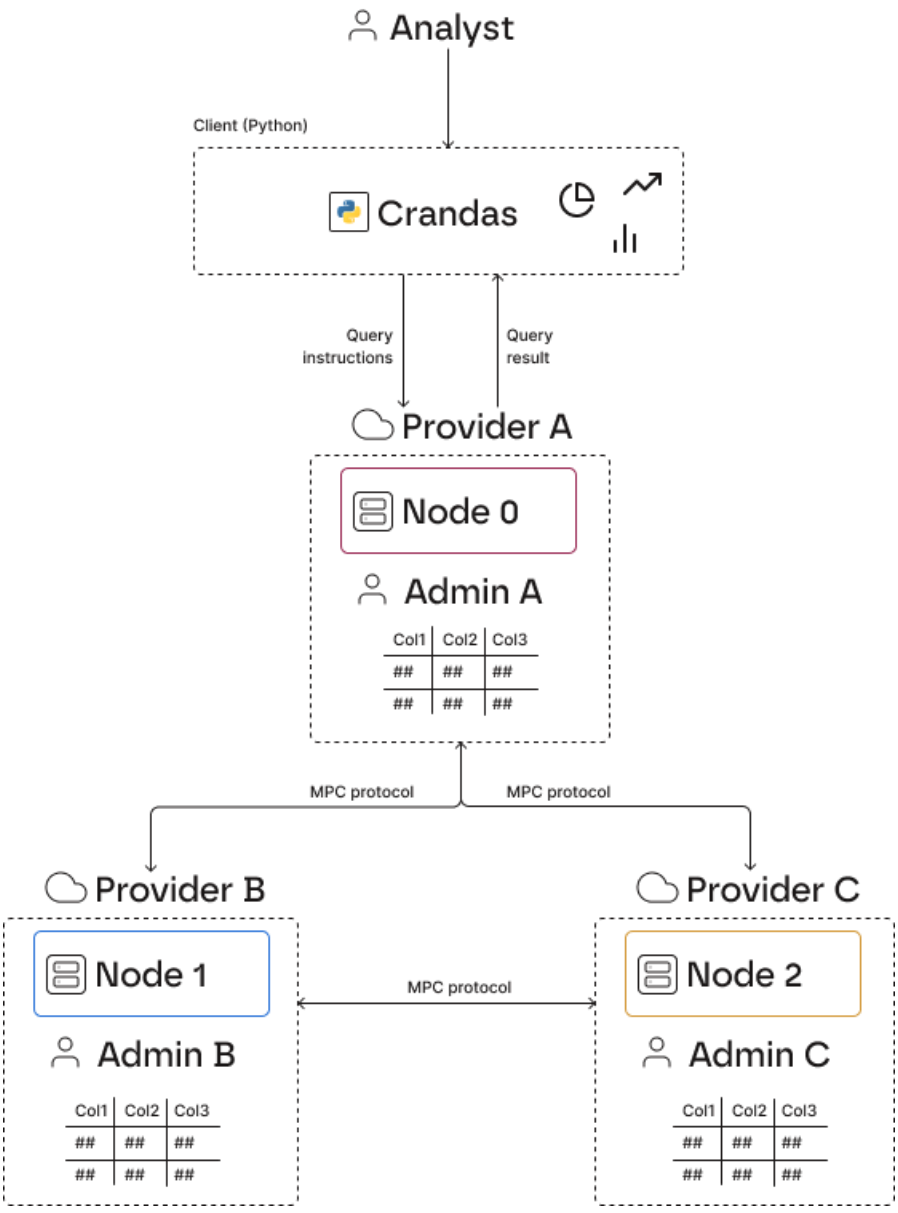
Each of the operations supported by the engine has been implemented in-house. Some of the protocols are based on the academic literature, while other protocols have been designed in-house, and for all used protocols we understand mathematically why the protocol is secure. To execute these protocols, each pair of compute servers communicates via a dedicated TCP/IP connection secured with TLS (v1.3). The protocols have been optimized for performance.

Crandas (Python package)

An analysis on secret-shared datasets, and all the steps of which it consists, is captured in a “crandas” script. Crandas is a Python package we developed ourselves to be able to compute on encrypted (secret shared) datasets in the engine. Its API is inspired by the widely used data science library pandas. In this way, users can write an analysis by means of a user-friendly interface, without having to worry about the underlying cryptographic principles. When a script is running in a Python environment in which crandas is installed (a “client”), and the environment is connected to a Roseman Labs deployment, the user can send queries to the engine to perform operations on datasets that it stores.

When a user runs a crandas script, the package will convert each of the script’s crandas commands into a query description in JSON format. This JSON captures the properties of the query, including the type of operation and the unique table handles of the datasets on which it should be applied. These are then sent to one of the engine nodes

(say node 0) over HTTPS. Node 0 will then orchestrate the execution of the operation with the rest of the nodes and communicate the result, encoded as JSON, back to the client. The crandas library then parses this encoding and transforms the result into Python objects for the user to work with. The result can be either the table metadata or, if opening of the result was authorized, an actual pandas DataFrame.



To connect to the engine, the user needs to refer to the following:

- Endpoint: URL of the DNS-record of engine node 0. This is where queries from crandas should be sent to.
- Port number: Open port of node 0 that the client can connect to.
- API-token: unique string variable that should be referred to from the script. This authorizes the user to send queries to the engine.
- Certificate: Authenticates the user as a valid entity to communicate with the engine.
- Server public keys: public keys of each of the nodes. These are needed to securely communicate with the nodes using public-key cryptography.

These values can be specified directly as environment variables in the script. To make connecting to the engine easier, they are also provided in a single "connection file" that can be downloaded from the platform ("Software components" > "Platform"). To connect to the engine, the client then only needs to refer to this connection file in their script.

Platform

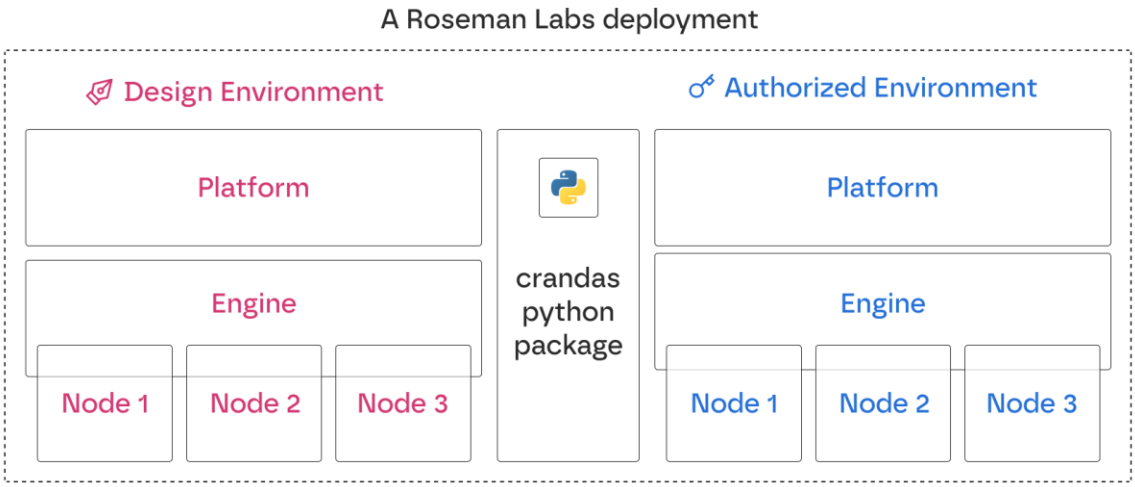
The platform is a web-application that can be used to perform various tasks in the engine in a user-friendly way, without having to run Python/crandas code. The platform can be used to do the following:

- Upload datasets
- Delete datasets
- Send out data requests (with data validation if desired)[7]

[7] For more information on the data request functionality, see: https://support.rosemanlabs.com/data-request-functionality

- Send out surveys[8]
- Request approval for an analysis (see “Data flow” > “Authorization”)
- Approve an analysis
- Access cryptographic material to connect to the engine (e.g. connection file)

The rights that a user in the platform has depends on their role, which can be assigned by the admins[9].



Account authentication

Account authorization and authentication in the platform are managed by Keycloak[10], an identity and access management component. Keycloak manages the roles[11] that each account has and the associated rights in the platform.

[8] For more information on the survey functionality, see https://support.rosemanlabs.com/how-to-create-a-survey
[9] For information about platform roles, see https://support.rosemanlabs.com/roles-in-the-platform
[10] https://www.keycloak.org/
[11] https://support.rosemanlabs.com/roles-in-the-platform

On all accounts, multi-factor authentication is enabled. Logging in can either be done using local authentication (with username/password) or via SAML/OICD using the customer's identity provider.

Key generation

Once a user has an account that has been assigned the role of analyst or approver, they can use the platform to generate (client-side, i.e., in the user's browser) a public/private keypair. The latter is downloaded to the user's system and can be used to authenticate actions within the software (as either an analyst key or as a signing key for approvers (see also "Query handling" > "Authorization")).

Data flow

Here we describe how data flows through the system during data uploading and query execution.

Input module

Data can be uploaded into the engine either via the platform, or directly by means of a crandas query. In either case, the table is encrypted locally client-side, meaning that the platform does not get access to the values in the table. The nodes only get access to the tables in secret-shared format.

Data uploading is taken care of by the input module, which is an engine component that is called by either crandas or the platform. When a dataset is uploaded to the engine, for each value in the dataset a one-time-pad (OTP) is established between the client and each pair of nodes. The client then encrypts the value v with all OTPs and sends it via node 0 to all nodes. Based on their OTPs and the encrypted value, each node can decrypt the value and calculate its own secret share without seeing v.

A detailed explanation of the procedure is given below[12]:

Distributing seed shares

1. Client sets up an HTTPS-connection with node 0
2. Client generates a random seed value for each pair of nodes ($seed_{0\text{-}1}$, $seed_{0\text{-}2}$, $seed_{1\text{-}2}$).
3. Client encrypts each seed share with the public key of one of its nodes (e.g. $seed_{0\text{-}1}$ is encrypted with the public key of node 0, $seed_{1\text{-}2}$ with the public key of node 1 etc.)
4. Send the encrypted seed shares to their nodes (via node 0)
5. Each node that received a seed share sends it to its pair partner (e.g. node 0 sends $seed_{0\text{-}1}$ to node 1 via TLS-encryption).
6. Based on the seed shares, the client generates an OTP for each pair of nodes: $OTP_{0\text{-}1}$, $OTP_{0\text{-}2}$, $OTP_{1\text{-}2}$

Distributing data secret shares

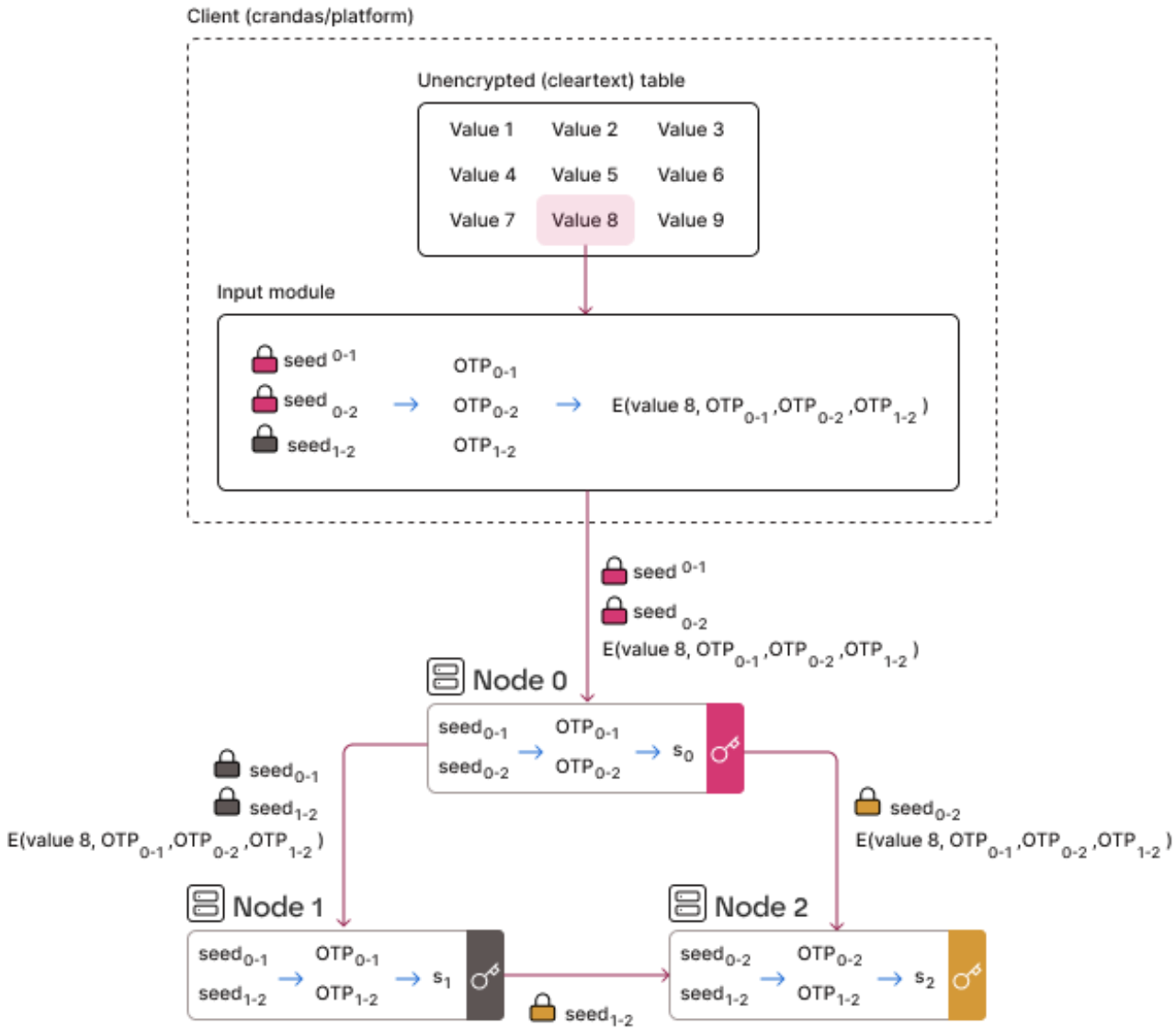
7. For each cell value in the table, the client parses the value into a set of integers such that it can be secret shared (e.g. for a string, every character is transformed into an integer using a standard encoding (e.g. ASCII))
8. Encrypt each (parsed) cell value v using all the OTPs and encryption function E: $E(v, OTP_{0\text{-}1}, OTP_{0\text{-}2}, OTP_{1\text{-}2})$
9. Pass $E(v, OTP_{0\text{-}1}, OTP_{0\text{-}2}, OTP_{1\text{-}2})$ to node 0, node 1 and node 2.

Calculating secret shares

10. On each node, the OTPs are produced based on the seed shares. For example, node 0 calculates $OTP_{0\text{-}1}$, $OTP_{0\text{-}2}$

[12] Note: This is a slightly simplified version of the procedure. For the actual details, please get in touch.

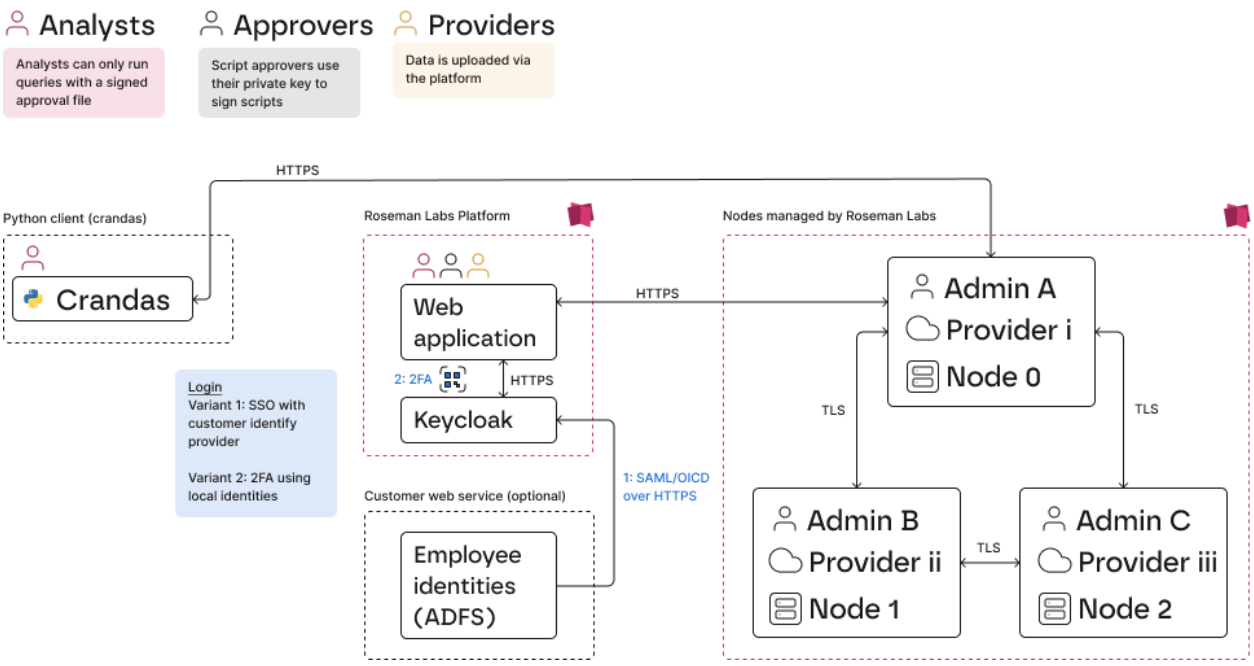
11. Based on the encrypted cell value *v* and the OTPs, each node performs an operation that withdraws the OTPs from the encryption and at the same time calculates their secret share. For example, node 0 calculates secret share s_0 based on $E(v, OTP_{01}, OTP_{02}, OTP_{12})$ and OTP_{01}, OTP_{02}.
12. The nodes can now collaboratively compute on the secret shares they hold using the given protocols.



Roseman Labs

Summary

The diagram below shows the different components of our software and how they interact:



Components

- Platform: web-application for uploading data and approving analyses
- Engine: Set of compute nodes on which the secret-shared data is stored and processed
- Crandas: python package, interacting with the engine, in which analyses are written. Crandas runs locally on the python-environment of the analyst.

Data flow

- Data is uploaded via the input-module, which is an engine component that can be called from either crandas or the platform.
- All communication between crandas and the engine, between the platform and the engine and among the nodes is encrypted.

- Node 0 acts as a forwarder of traffic to the other engine nodes.
- The input module takes care of the secret-sharing procedure in such a way, that neither the platform nor any of the nodes get access to the plaintext data.

Authorization

Roseman Labs can be used to compute aggregated statistics on sensitive (production) datasets. To prevent analysts from accessing information about individual records, we use a script approval system. Analysts first design their script on non-sensitive dummy data. Sensitive data is then uploaded to a different environment, in which a script may only be executed if it was properly authorized by a set of representatives of the organizations providing the data. These representatives are assigned by the platform's primary admin[13] during system initialization. More information about the script approval flow can be found in our documentation[14].

Script signing

Once an analyst wants to run their script on production data, they use the crandas script recording function to “record” all the steps in their script. Each query is then saved in a JSON-file, in the exact order as they occurred in the script. This file also mentions the public key of the analyst. After script approval, the analyst can then authenticate themselves to the engine by referencing their private key (analyst key). The JSON-file can be uploaded to the platform, where it can be accessed by the approver.

A script can be approved in two forms, which can be configured as a variable in the environment. Following the default method, the script may run in the exact sequence of

[13] See also: https://support.rosemanlabs.com/how-to-initialize-an-environment
[14] https://support.rosemanlabs.com/design-and-production

queries in which it was recorded. As an alternative, the analyst can be granted permission to run any query that occurs in the script, regardless of the order.

An approver can approve a script by signing it with their private key. This can be done via the platform. Signing is done by uploading the approver key to the platform, after which the JSON-file is signed using a digital signature. To keep the approver key private, signing is done client-side. The result is a signed file with the ".approved"-extension, which the analyst can download from the platform. Only by reference of the correct analyst key and a valid .approved-file, including the digital signature of all approvers configured in the engine, the queries in the script shall be executed.

Disclosure control

Even when a script has been properly assessed, there might also be scenarios where unwanted disclosure can occur during script execution. These can only be prevented by applying runtime disclosure metrics in the script, to make sure that the result is only computed when the metric holds. To help an analyst to properly implement these metrics, and an approver to assess if they have been sufficiently applied on potentially risky code fragments, the crandas documentation offers a disclosure control manual[15].

[15] https://docs.rosemanlabs.com/latest/gettingstarted/07-guide-for-approvers.html

Architecture & deployment

To deploy, configure, monitor and maintain our software, we use state-of-the-art tools and protocols.

Components

Each Roseman Labs deployment consists of a set of services, which each run on a docker container in a separate pod. These pods are orchestrated on a Kubernetes cluster on one of our virtual machines, which are hosted by a European cloud provider. On environments where segregation-of-duties is active, we ensure that the pods on which an engine node is running are each hosted by a different cloud provider (see also “Software components” > “Engine”). A deployment typically has a pod for the following services:

- Engine nodes
- Platform
- Keycloak
- Postgres database for keycloak
- Postgres database for platform

Routing

To route incoming traffic to the correct pods, we use a load balancer (traefik[16]) offered by the cloud provider. This ensures that an incoming URL ends up at the correct pod of the right environment. All traffic to and from the system pods are encrypted by a TLS-layer, having Let’s Encrypt[17] as the certificate authority.

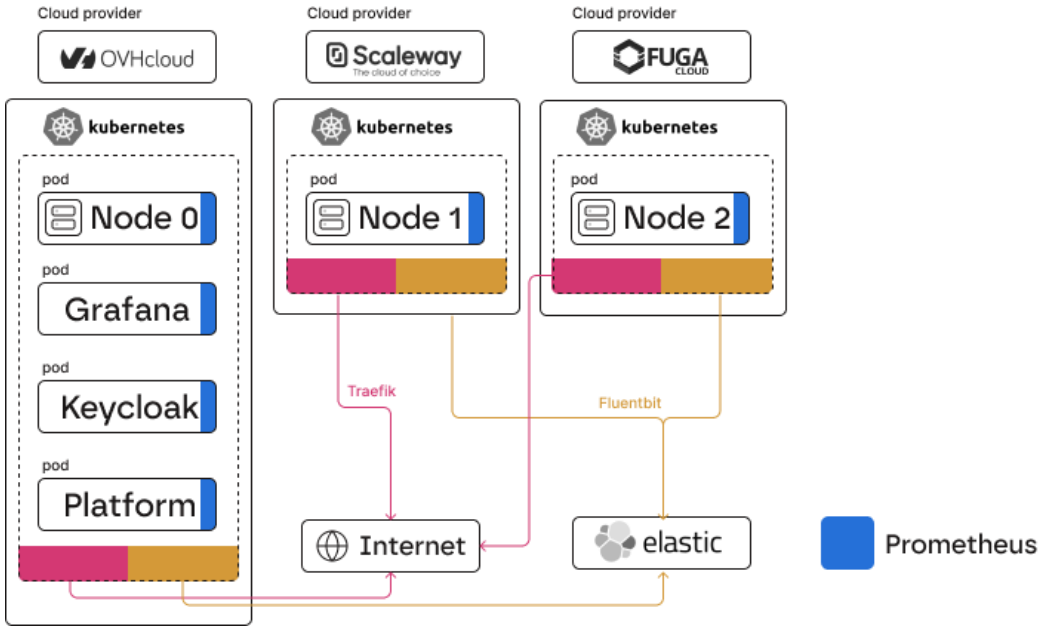
[16] https://doc.traefik.io/traefik/
[17] https://letsencrypt.org/

Infra-as-code

We manage our infrastructure according to "infra-as-code". This means that the deployment of the basic infrastructure of an environment is not managed ad-hoc, but that these can be automatically created and adapted using Terraform code. Besides, for certain services we use Helm charts to lay out the configuration details of the pods (for example for Keycloak and the engine). We use GitHub (GitOps) to manage our continuous deployment pipeline.

Logging

We allow logging of many types of metrics and events in either the platform or the engine. As part of each pod we have a Prometheus[18] component for basic metrics, which is scraped by Grafana[19] for visualization. Besides, fluentbit[20] tracks Kubernetes logs, which can be queried using Elastic.



[18] https://prometheus.io/
[19] https://grafana.com/
[20] https://fluentbit.io/

Roseman Labs

Quality and certification

We take quality and security very seriously. To this end, we have our software regularly assessed and audited[21] by external organizations. Auditability, and the ability for customers to assess the robustness of our product, ensures that systems are secure by both design and implementation.

Certifications

ISO 27001

Every year, we are audited by DigiTrust against the ISO 27001 norm, one of the best-known standards for information security management systems (ISMS).

NEN 7510

As an extension to ISO 27001, we have been certified to conform to NEN 7510. This Dutch standard is tailored to healthcare service providers and organizations processing healthcare data. Additional principles it includes are demands for interoperability and data breach prevention.

Baseline Product Security Assessment (BPSA)

BSPA is a certification scheme developed and maintained by the AIVD (Dutch General Intelligence and Security Service) aimed at the security needs of the Dutch government (and in exceptional cases, also private organizations and the private sector). BSPAs are

[21] More information on our certifications, including the statements of applicability, can be found here https://rosemanlabs.com/en/blogs/our-commitment-to-cyber-security and here https://support.rosemanlabs.com/risk-compliance#certifications

executed by (private) evaluation labs that are supervised by the AIVD. Our product has successfully passed this certification. For more information about the security level and the scope of the BSPA evaluation, please refer to the BSPA deployment advisory[22].

SIG code audit

The Software Improvement Group (SIG) has performed an extensive source code assessment of our software, auditing its maintainability, security and cryptographic properties. As part of this, SIG collaborated with experts in the formal evaluation of cryptographic tools. A rigorous manual code review was also done, including how the implementation of our cryptographic protocols relates to scientific literature. The assessment was performed against the ISO 25010 standard and SIG's Cryptographic trustworthiness model.

Penetration tests

In addition to the above, we apply standard penetration tests, conducted by external parties to test the information security of our product. This is done at least annually. For the reporting of security findings, we invite security researchers to review our Responsible Disclosure Policy[23]. For the latest information about our product updates and bug-fixes, please visit our changelog[24].

[22] https://www.aivd.nl/onderwerpen/informatiebeveiliging/ontwikkeling-en-evaluatie-beveiligingsproducten/bspa-geevalueerde-producten
[23] https://rosemanlabs.com/en/responsible-disclosure-policy
[24] https://docs.rosemanlabs.com/latest/changelog.html

Risk and compliance

Before you start a collaboration on sensitive data, you might have some questions relating to the legal implications. More information on how our product relates to the principles laid out in the GDPR can be found on our help center[25]. Here we also explain how we help drafting a Data Processing Agreement (DPA) and a Data Protection Impact Assessment (DPIA) as part of our service.

[25] https://support.rosemanlabs.com/risk-compliance