D3.3
Proof-of-Concept Prototype for Data Management

Project number:	643964
Project acronym:	**SUPERCLOUD**
Project title:	User-centric management of security and dependability in clouds of clouds
Project Start Date:	1st February, 2015
Duration:	36 months
Programme:	H2020-ICT-2014-1
Deliverable Type:	Demonstrator
Reference Number:	ICT-643964-D3.3/ 1.0
Work Package:	WP3
Due Date:	May 2017 - M28
Actual Submission Date:	31st May, 2017
Responsible Organisation:	FFCUL
Editor:	Alysson Bessani
Dissemination Level:	PU
Revision:	1.0
Abstract:	This deliverable describes the data management software components produced in the SUPERCLOUD project. Besides briefly presenting the components, we describe how these components can be obtained and used.
Keywords:	data management, fault tolerance, security, anonymization, encryption, replication, blockchain, cloud storage

This project has received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement No 643964.

This work was supported (in part) by the Swiss State Secretariat for Education, Research and Innovation (SERI) under contract number 15.0091.

Editor

Alysson Bessani(FFCUL)

Contributors (ordered according to beneficiary numbers)

Mario Münzer (TEC)
Sébastien Canard, Nicolas Desmoulins, Marie Paindavoine (ORANGE)
Marko Vukolic (IBM)
Alysson Bessani (FFCUL)
Daniel Pletea (PEN)

Disclaimer

Executive Summary

This report describes the software components produced in SUPERCLOUD WP3, which is responsible for proposing techniques, methods, and tools for designing data management services in a cloud-of-clouds environment. In this document we describe three types of components: a file storage service built on top of existing public cloud storage services (e.g., Amazon S3, Windows Azure, Google Storage), a framework for developing decentralized blockchain applications on top of state of the art Byzantine consensus algorithms, and tools and libraries for integrating advanced privacy-preserving features on cloud applications. These components comprise most of the research contributions of WP3, many of them published in top conferences and journals.

The presented components can be used either together, complementing each other capabilities, or as standalone software packages integrated to applications.

Besides presenting a brief description of the components, this report also describes how to obtain and use the software, in many cases pointing to open-source repositories.

Contents

List of Figures

Chapter 1 Introduction

The WP3 of SUPERCLOUD is responsible for designing and implementing secure and dependable data management services in a cloud-of-clouds environment. The overall architecture envisioned for the project data management, and the main contributions in this context were described in a previous deliverables [5, 6]. In a nutshell, this architectures considers many different instantiations of data management services. This includes the use of existing public cloud service like Amazon S3 and Rackspace Files for storing files and support cheap disaster recovery, the design of custom blockchain-like multi-cloud services, and new tools and programming libraries for implementing advanced security features in existing applications.
This short report presents the main software components developed in WP3, and is part of D3.3, which is a "demonstrator" deliverable containing this components. More specifically, here we describe the five main software systems that integrate most of the contributions developed in WP3, namely:

- Janus (Chapter 2) is a cloud-backed storage service that can be configured on the web, by specifying workloads and requirements for data storage, and them using a proxy locally to transfer this data to/from a set of cloud storage services selected by the system.
- Hyperledger Fabric (Chapter 3) is a permissioned blockchain framework that employs Byzantine-resilient consensus on its core. Although Hyperledger Fabric is not 100% supported by SUPER-CLOUD, we are contributing with several alternatives for implementing the agreement on the ordering of blocks.
- WP3 also produced a K-anonymization tool (Chapter 4) that can be used to remove privacy-sensible information from datasets that need to be stored in the cloud.
- Trinocchio (Chapter 5) is a secure multi-party computation middleware that can be used to support private computations in distributed applications spanning over mutually untrusted parties.
- SUPERCLOUD-ABE (Chapter 6) is a framework for implementing attribute-based encryption. This allows the encryption of data records in such a way that only parties satisfying certain attributes can decrypt such records.

Each of these chapters gives a short overview of these components (whole descriptions can be found in D3.1 [5] and D3.2 [6]), together with information about how to obtain and use the component. We finish this report with a short conclusion and some notes about how the components can be used together (Chapter 7).

Chapter 2 Janus Storage Service

2.1 Introduction

Janus is a cloud-of-clouds storage system that maintains data in a dependable and secure way using multiple cloud providers as storage backends. The system employs several Byzantine-resilient replication and coding algorithms [8] to spread the stored data in multiple cloud storage services (Amazon S3, Google Storage, Rackspace Files, etc.) in such a way that fault tolerance and confidentiality is preserved even if a fraction of these providers is compromised. These providers are selected and configured by the Janus platform, using the user-specified workloads and/or requirements.

2.2 Component Description

Janus allows the creation of virtual disks (*volumes*) backed by cloud storage services selected in accordance with a set of specifications defined by the users. More specifically, a user could have as much *volumes* as he wants, each one with a different configuration to attend different data requirements. The architecture of the platform – presented in Figure 2.1 – is designed to improve storage efficiency in terms of management effort, costs, and performance, while ensuring the compliance with user requirements.

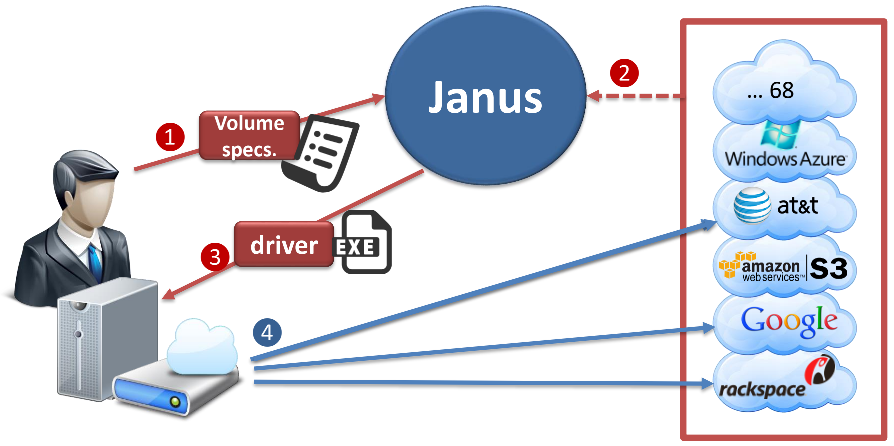
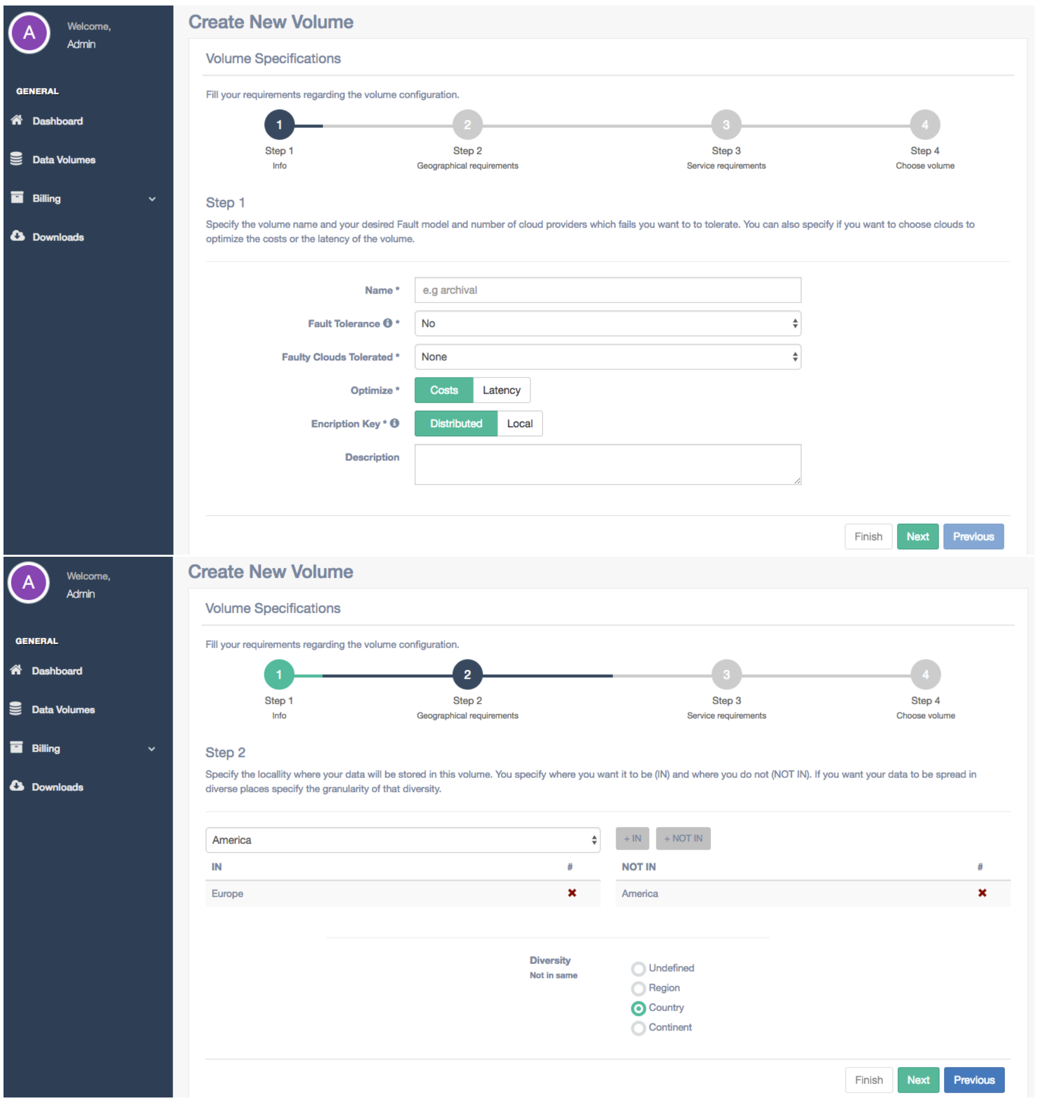


Figure 2.1: Janus architecture.

Figure 2.1 also describes the required steps until a user can get a ready-to-use *volumes*, i.e., a virtual disk with specific storage requirements. First, the user contacts the Janus server to provide his requirements and constraints (1) (part of the interface for this is shown in Figure 2.2). Then, the server finds the best possible storage profiles for that request and shows them to the user (2), which picks one of them. After that, the driver with the chosen profile (which includes the cloud storage accounts generated by Janus) is then downloaded and installed in the client's local machine (3). Lastly, all the data present in the Janus virtual disk is stored in the clouds according to the user-defined requirements (4).

The Janus platform is essentially composed of two components:

- *janus server* is responsible for finding the most adequate storage configurations given the volume specifications. This server is also responsible for the creation and management of the cloud accounts, and for maintaining the information about the available clouds updated (i.e., available locations, latency, prices, etc.). Figure 2.2 shows two of the web forms users interact for defining its requirements when creating a volume.
- *virtual disk driver* accommodates all the different storage *volumes* of the client and is responsible for managing the system's data-plane. The driver can be installed either as an NFS proxy in the same site as the clients or directly on the client's machines (as files systems or dropbox-like applications).



Fill your requirements regarding the volume configuration.

Step 1

Specify the volume name and your desired Fault model and number of cloud providers which fails you want to to tolerate. You can also specify if you want to choose clouds to optimize the costs or the latency of the volume.

Name *	e.g archival
Fault Tolerance *	No
Faulty Clouds Tolerated *	None
Optimize *	[✓ Costs] Latency
Encription Key *	[✓ Distributed] Local
Description	

Fill your requirements regarding the volume configuration.

Step 2

Specify the locallity where your data will be stored in this volume. You specify where you want it to be (IN) and where you do not (NOT IN). If you want your data to be spread in diverse places specify the granularity of that diversity.

America + IN + NOT IN

IN	#	NOT IN	#
Europe	✖	America	✖

Diversity
Not in same

Undefined Region [✓ Country] Continent

Figure 2.2: Janus web interface.

This platform design allows the system to have some interesting capabilities in terms of privacy, performance and fault tolerance. The fact the virtual disk driver is serverless, i.e., it interacts directly with the clouds without contacting the Janus server. This allows the system data-plane to operate directly with the cloud services without requiring any mediation or coordination/security anchor. This enables the virtual disk, in one hand, to be not dependent of the availability of the Janus server, and in the other, to achieve better performance (as there is no server bottleneck). Moreover, since the cloud accounts are created and managed by the Janus server, the clouds will never know the identity of the client. Note that, although Janus has access to the data client stores in the clouds, all the data is encrypted at the client-side with a key only he knows, ensuring that only the client has access to the original (unencrypted) data.

2.3 Code/Component Access

In this section we describe how Janus can be installed as a NFS server inside a Docker container. The server is implemented using Java, so it can be deployed in any operating system that supports Docker. More specifically, we make available a deploy-ready Janus NFS Docker image, that can be easily integrated with other SUPERCLOUD components, in particular with the network virtualization element being developed in WP4. The commands below are for Windows deployment, but they are almost the same for the other OSs. For instance, for Linux based OSs, you should use the .sh shell scripts we provide instead the .bat ones. Moreover, you may need to run the commands with sudo capabilities.
In the following we present a short overview of the steps required for running the system. Nevertheless, before going into the steps, you must first request a Janus account by sending an email to `anbessani@ciencias.ulisboa.pt`.

1. Login into the Janus platform at `http://janus.lasige.di.fc.ul.pt`;
2. Download the Janus NFS Docker image on the “Download” tab on the left side;
3. Uncompress `janus-nfs.zip`;
4. Open the terminal, and go to the uncompressed `janus-nfs` folder;
5. Load the Janus NFS image by running `docker load -i janus-nfs.docker`;
6. Run the container by running `run.bat`;[1]
7. Wait until the output of running status.bat be “RUNNNING”;
8. (On Windows) Install the “Services for NFS” package, which is part of “Windows Features”);
9. Mount the Janus NFS share by running `mount-janus-nfs.bat`;
10. Open “Explorer”, go to “This PC” and there you will find a “Janus” network location which is the Janus NFS share.[2]

Note. All the steps and scripts provided were tested only in Ubuntu 16.04 and Windows 10 (Build 14393). We are working on the Docker container for MacOS 10.12.4. However, the Janus NFS server (undockerized) works in all of these platforms.

2.4 Documentation

A more complete description of the system can be found in Chapter 10 of SUPERCLOUD D3.2 [6]. Further documentation about the software, as how to use a private keystore to encrypt the data or how to add the volumes on the system, is distributed together with the zip file describe above.

[1]This container will have a default volume configured to spread the data across the globe and in different storage providers.

[2]Note that it is impossible to write to the root folder of the Janus NFS folders, you can only work in volumes' folders.

Chapter 3 Hyperledger Fabric

3.1 Introduction

In general, blockchains are distributed ledgers (typically immutable and totally ordered) of transactions pertaining to distributed applications. Distributed applications may be cryptocurrencies (such as Bitcoin) but also general applications (i.e., state machines, sometimes called smart contracts). Permissioned blockchains [12, 13] are those blockchains in which the membership of the nodes that hold copies of the ledger is restricted and managed in some way.
Permissioned blockchains work across multiple administrative domains and are a good match for SUPERCLOUD project goals and its data management requirements. SUPERCLOUD project contributes to the Hyperledger Fabric open-source blockchain project and benefits from it. Hyperledger Fabric (HLF)[1] is an open-source project within the Hyperledger umbrella project under the auspices of the Linux Foundation. HLF is a modular general-purpose permissioned blockchain system which can be also seen as a distributed operating system for permissioned blockchains.
SUPERCLOUD project contributes to HLF project by influencing its overall architecture, including approach to handling non-determinism in the system [2]. It also serves as an integration vector for State-machine replication related components of SUPERCLOUD as described in SUPERCLOUD Deliverable D3.2. Notably these include:

- Component that treats non-determinism when replicating arbitrary applications when replicas can fail in an arbitrary (i.e., Byzantine) way [2].
- Component that introduces a novel model for developing reliable distributed protocols called XFT [4].
- Component that empirically evaluates latency-optimization for state-machine replication in WANs and informing the design of novel state-machine replication protocols [10].
- Component that introduces a generic state-transfer tool for partitioned state-machine replication that enables elasticity [7].

The integration of the last two components is done via integration of HLF and BFT-SMaRt library [1] in which the two mentioned components were implemented.

3.2 Component Description

The architecture of the component is described in the main architecture document `https://github.com/hyperledger/fabric/blob/master/proposals/r1/Next-Consensus-Architecture-Proposal.md`. This overall architecture has been developed in part in the SUPERCLOUD project.
Other lower level architecture design documents are available at `https://wiki.hyperledger.org/community/fabric-design-docs`. With the exception of the SBFT consensus, which was developed in the context of the SUPERCLOUD project, other components have been devloped outside SUPERCLOUD, by the community.

[1]`https://github.com/hyperledger/fabric`

3.3 Code/Component Access

The Hyperledger Fabric v1 code is accessible at `https://github.com/hyperledger/fabric/`. The integration between Hyperledger Fabric v1 and BFT-SMaRt is still under work[2]. Meanwhile, the code for BFT-SMaRt is available on `http://bft-smart.github.io/library/`.

3.4 Documentation

The documentation is available at `https://hyperledger-fabric.readthedocs.io/en/latest/`

[2]Recall that HLF is not only a SUPERCLOUD component, but a large and complex project with many stakeholders. Therefore, adding a new component in its codebase is complex process which requires the development of several performance and integration tests.

Chapter 4 Anonymization

4.1 Introduction

Anonymization techniques open the possibility of releasing personal and sensitive data, while preserving individual's privacy. Therefore, data anonymization guarantees that revealed data cannot be assigned to a natural person nor inferences to user's identity can be made. There are several techniques known, which are applicable for data anonymization, as described in the SUPERCLOUD deliverable D3.2 [6]. The characterized data anonymization tool within this chapter is among others based on k-anonymity, whereby the focus is put on the irreversibility of released data.
The aim of the data anonymization tool is to meet the goal of k-anonymity regarding irreversibility. As a result of this, the disclosure of sensitive data (e.g. health data) is impossible and the opening of data is enabled, whereby each data record is at least $k-1$ from other records indistinguishable with respect to the quasi-identifier. However, the tool is not simply performing calculations on medical data and anonymizing them. Moreover, the tool aims to calculate the best solution for the given data in terms of cost-efficiency. This is done by means of so-called cost metric calculation as well as the Optimal Lattice Anonymization (OLA) algorithm, as described in D3.2 in detail.
Privacy-enabling mechanisms for untrusted cloud(s) represent an explorative subdomain of the SUPERCLOUD architecture. Therefore, data anonymization techniques, such as k-anonymity, were from the beginning of the project under consideration in the overall architecture design (depicted in SUPERCLOUD's deliverable D3.1 [5]). In the architecture proposal of WP3 of SUPERCLOUD, the data anonymization tool is used to enable the release of medical-related sensitive data. In order to guarantee a secure storage as well as a trusted health data exchange, the tool will be integrated within the SUPERCLOUD data management architecture, and in particular with the Janus storage service.

4.2 Component Description

As already mentioned, the data anonymization tool is among others based on k-anonymity. Therefore, generalization and suppression is used, as described in detail in the previous deliverable D3.2 [6]. While suppression deletes uniquely identifying attributes, generalization is necessary in order to obtain k-anonymity, respectively to obtain k-identical values by generalizing the pre-selected attributes (a set of these attributes is called quasi-identifier in the following). One further element of the data anonymization tool is the precision cost metric algorithm by Sweeney [11]. Besides the achievement of the anonymity level (k) for given medical data, the precision of each applicable node has to be calculated. As a result, the height and depth of the generalization hierarchy will be considered and the ratio between applicable and total generalization steps determined. As mentioned previously, the main goal of the data anonymization tool is to (besides the irreversible anonymization and opening of data) determine the best solution in terms of cost-efficiency with most minimal information loss for the given data. Hence, a potential solution is given by its k and precision. However, there could be several potential solutions available with same characteristics based on anonymity level and precision only. Therefore, the data anonymization tool includes one more important element in order to determine the optimal solution for the provided medical data. The OLA algorithm [3] is the last element of the anonymization tool and is responsible for determining efficiently the optimal solution, respectively the optimal node in the so-called lattice, by means of divide-and-conquer technique. The detailed steps of

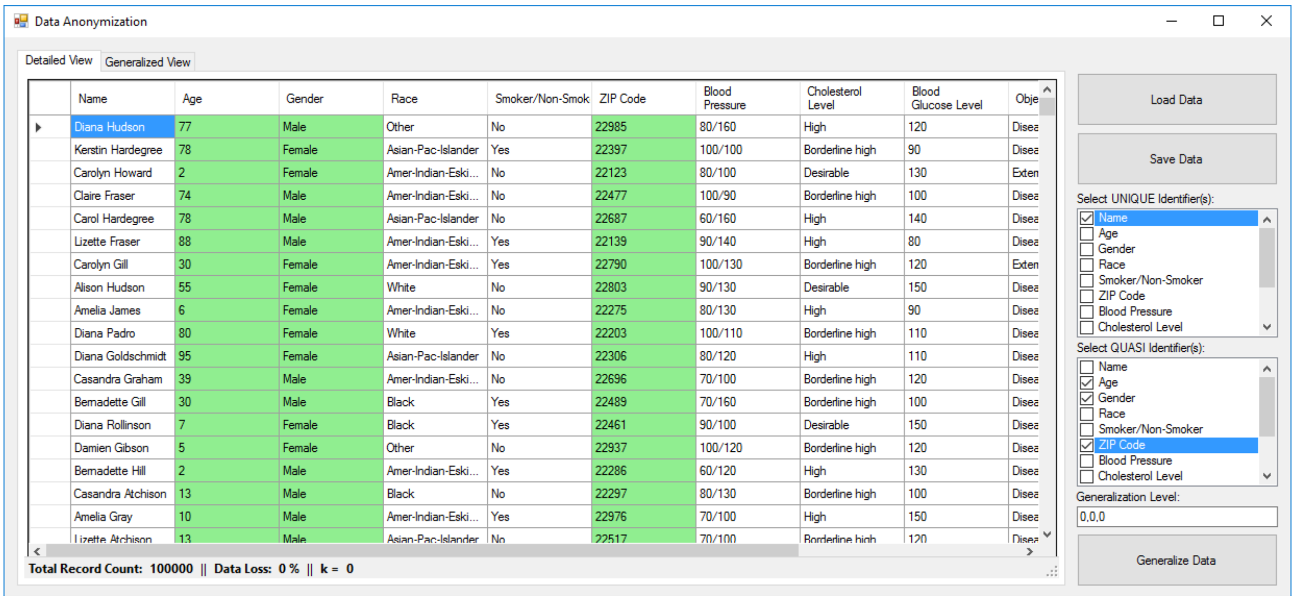
the Optimal Lattice Anonymization as well as a pseudo code of all including functions of the algorithm can be found in deliverable D3.2 [6].
Since the OLA algorithm has to traverse through all possible solutions, a list of nodes, also known as lattice, is required as input. The lattice represents a stepped generalization of the given data in form of a node list and contains the current generalization step, the k and the information loss, respectively the cost metric precision. The lattice itself is built automatically within the anonymization tool based on the quasi-identifier and its total possible generalization steps, respectively the total generalization step of each attribute. The OLA algorithm traverses through the lattice according to the divide-and-conquer principle and marks all non-applicable as well as already traversed nodes with tags for best efficiency. The detailed procedure of the lattice traverse can be found in the previous SUPERCLOUD's deliverable D3.2 [6] as well.

4.2.1 Software Development

The software development of the data anonymization tool was completely done in Microsoft's object-oriented programming language C#. Therefore, a graphical user interface (GUI) was made as well in order to support the user's input and control. Since the software tool was built by means of Microsoft's Visual Studio default libraries (included in .NET 4.x framework), such as *System.IO*; *System.Windows.Forms*; *System.Threading*; *System.Data*; etc, there is no further external or third-party library necessary to run the program. The final software tool is resulting in an executable program (*.exe), which is supported on computers with Windows operating system only. Further details about the tool access can be found at the end of this chapter (section 4.3).

4.2.2 Procedure

The data anonymization tool is composed of three main components: k-anonymity calculation; cost metric computation; OLA algorithm. Therefore, the procedure from the input of plain health data to the output of irreversible anonymized data records is straightforward. The tool accepts as input plain data records in comma-separated values (CSV) file format. Given a valid source file, the user then has to select the unique-identifying attribute(s) as well as the quasi-identifier, as it is depicted in Figure 4.1 (selected quasi-identifier attributes highlighted in green).



Name	Age	Gender	Race	Smoker/Non-Smok	ZIP Code	Blood Pressure	Cholesterol Level	Blood Glucose Level	Obje
Diana Hudson	77	Male	Other	No	22985	80/160	High	120	Disea
Kerstin Hardegree	78	Female	Asian-Pac-Islander	Yes	22397	100/100	Borderline high	90	Disea
Carolyn Howard	2	Female	Amer-Indian-Eski...	No	22123	80/100	Desirable	130	Exten
Claire Fraser	74	Male	Amer-Indian-Eski...	No	22477	100/90	Borderline high	100	Disea
Carol Hardegree	78	Male	Asian-Pac-Islander	No	22687	60/160	High	140	Disea
Lizette Fraser	88	Male	Amer-Indian-Eski...	Yes	22139	90/140	High	80	Disea
Carolyn Gill	30	Female	Amer-Indian-Eski...	Yes	22790	100/130	Borderline high	120	Exten
Alison Hudson	55	Female	White	No	22803	90/130	Desirable	150	Disea
Amelia James	6	Female	Amer-Indian-Eski...	No	22275	80/130	High	90	Disea
Diana Padro	80	Female	White	Yes	22203	100/110	Borderline high	110	Disea
Diana Goldschmidt	95	Female	Asian-Pac-Islander	No	22306	80/120	High	110	Disea
Casandra Graham	39	Male	Amer-Indian-Eski...	No	22696	70/100	Borderline high	120	Disea
Bernadette Gill	30	Male	Black	Yes	22489	70/160	Borderline high	100	Disea
Diana Rollinson	7	Female	Black	Yes	22461	90/100	Desirable	150	Disea
Damien Gibson	5	Female	Other	No	22937	100/120	Borderline high	120	Disea
Bernadette Hill	2	Male	Amer-Indian-Eski...	Yes	22286	60/120	High	130	Disea
Casandra Atchison	13	Male	Black	No	22297	80/130	Borderline high	100	Disea
Amelia Gray	10	Male	Amer-Indian-Eski...	Yes	22976	70/100	High	150	Disea
Lizette Atchison	13	Male	Asian-Pac-Islander	No	22517	70/100	Borderline high	120	Disea

Total Record Count: 100000 || Data Loss: 0 % || k = 0

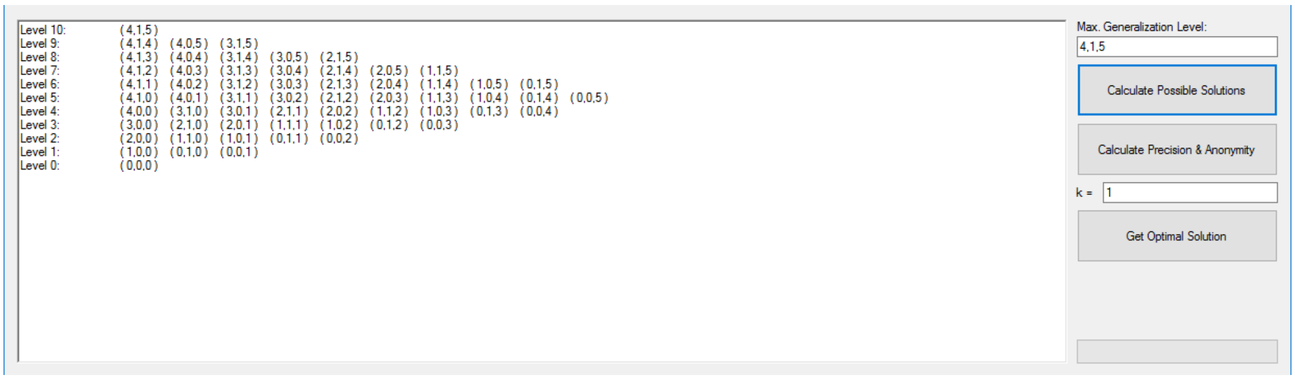
Select UNIQUE Identifier(s):
- [x] Name
- [ ] Age
- [ ] Gender
- [ ] Race
- [ ] Smoker/Non-Smoker
- [ ] ZIP Code
- [ ] Blood Pressure
- [ ] Cholesterol Level

Select QUASI Identifier(s):
- [ ] Name
- [x] Age
- [x] Gender
- [ ] Race
- [ ] Smoker/Non-Smoker
- [x] ZIP Code
- [ ] Blood Pressure
- [ ] Cholesterol Level

Generalization Level: 0,0,0

Figure 4.1: Selection of unique- and quasi-identifier attributes

By means of look-up tables (LUTs), the tool automatically checks the total generalization level of each selected quasi-identifier attribute[1]. Based on the quasi-identifier and its total generalization level, the node list (lattice) can be built, which is depicted in Figure 4.2.



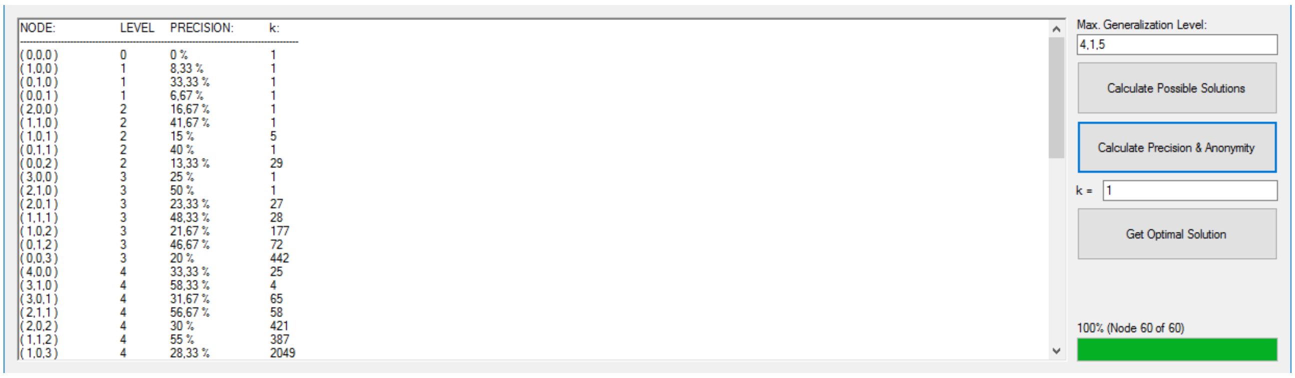
```
Level 10:    ( 4,1,5 )
Level 9:     ( 4,1,4 )  ( 4,0,5 )  ( 3,1,5 )
Level 8:     ( 4,1,3 )  ( 4,0,4 )  ( 3,1,4 )  ( 3,0,5 )  ( 2,1,5 )
Level 7:     ( 4,1,2 )  ( 4,0,3 )  ( 3,1,3 )  ( 3,0,4 )  ( 2,1,4 )  ( 2,0,5 )  ( 1,1,5 )
Level 6:     ( 4,1,1 )  ( 4,0,2 )  ( 3,1,2 )  ( 3,0,3 )  ( 2,1,3 )  ( 2,0,4 )  ( 1,1,4 )  ( 1,0,5 )  ( 0,1,5 )
Level 5:     ( 4,1,0 )  ( 4,0,1 )  ( 3,1,1 )  ( 3,0,2 )  ( 2,1,2 )  ( 2,0,3 )  ( 1,1,3 )  ( 1,0,4 )  ( 0,1,4 )  ( 0,0,5 )
Level 4:     ( 4,0,0 )  ( 3,1,0 )  ( 3,0,1 )  ( 2,1,1 )  ( 2,0,2 )  ( 1,1,2 )  ( 1,0,3 )  ( 0,1,3 )  ( 0,0,4 )
Level 3:     ( 3,0,0 )  ( 2,1,0 )  ( 2,0,1 )  ( 1,1,1 )  ( 1,0,2 )  ( 0,1,2 )  ( 0,0,3 )
Level 2:     ( 2,0,0 )  ( 1,1,0 )  ( 1,0,1 )  ( 0,1,1 )  ( 0,0,2 )
Level 1:     ( 1,0,0 )  ( 0,1,0 )  ( 0,0,1 )
Level 0:     ( 0,0,0 )
```

Max. Generalization Level: 4,1,5

k = 1

Figure 4.2: Generation of lattice based on maximum generalization level [4,1,5]

Since the OLA algorithm requires for the calculation of the best/optimal solution the k (anonymity level) and the precision (information loss), a further computation has to be done. The resulting output can be found in Figure 4.3



```
NODE:          LEVEL   PRECISION:    k:
--------------------------------------------------
( 0,0,0 )      0       0 %           1
( 1,0,0 )      1       8,33 %        1
( 0,1,0 )      1       33,33 %       1
( 0,0,1 )      1       6,67 %        1
( 2,0,0 )      2       16,67 %       1
( 1,1,0 )      2       41,67 %       1
( 1,0,1 )      2       15 %          5
( 0,1,1 )      2       40 %          1
( 0,0,2 )      2       13,33 %       29
( 3,0,0 )      3       25 %          1
( 2,1,0 )      3       50 %          1
( 2,0,1 )      3       23,33 %       27
( 1,1,1 )      3       48,33 %       28
( 1,0,2 )      3       21,67 %       177
( 0,1,2 )      3       46,67 %       72
( 0,0,3 )      3       20 %          442
( 4,0,0 )      4       33,33 %       25
( 3,1,0 )      4       58,33 %       4
( 3,0,1 )      4       31,67 %       65
( 2,1,1 )      4       56,67 %       58
( 2,0,2 )      4       30 %          421
( 1,1,2 )      4       55 %          387
( 1,0,3 )      4       28,33 %       2049
```

Max. Generalization Level: 4,1,5

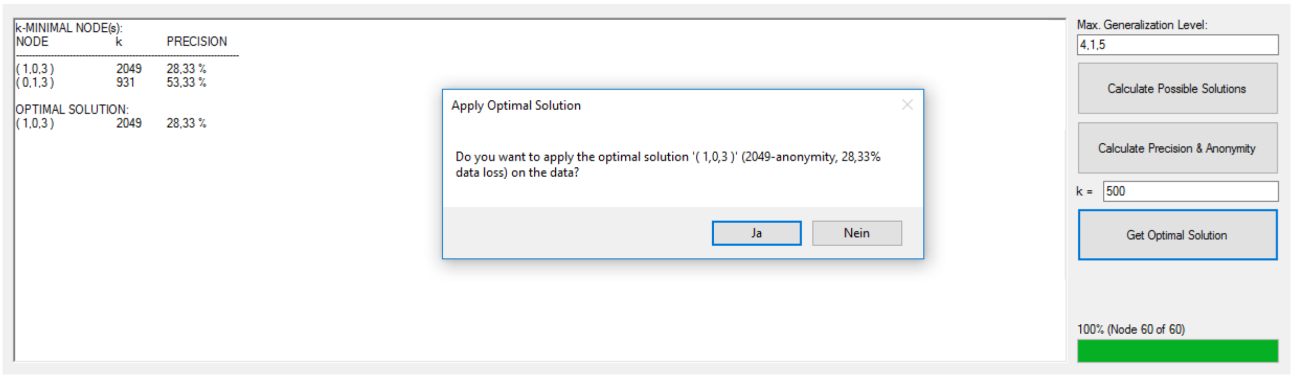
k = 1

100% (Node 60 of 60)

Figure 4.3: Calculation of anonymity level and precision based on medical data records and its maximum generalization level [4,1,5]

Besides the plain health data and identifier selection, the user has to select the lower bound for the anonymity level (k). In the end of the procedure, the OLA algorithm based on the valid inputs (source file with data records; quasi-identifier; total generalization level; lattice; anonymity level boundary) can be applied. At this stage, the OLA algorithm computes the best-fitting node of the lattice based on the provided characteristics and the optimal solution will be displayed. However, the calculation is performed on the lattice only, so no anonymization on the provided data will take place at this time. Therefore, the user has now the possibility to apply the optimal solution on all loaded data records or decline the result, as seen in Figure 4.4.

[1]If there is no pre-defined LUT for the selected quasi-identifier attribute, a default value will be assumed



```
k-MINIMAL NODE(s):
NODE          k        PRECISION
------------------------------------
( 1,0,3 )     2049     28,33 %
( 0,1,3 )     931      53,33 %

OPTIMAL SOLUTION:
( 1,0,3 )     2049     28,33 %
```

Apply Optimal Solution

Do you want to apply the optimal solution '(1,0,3)' (2049-anonymity, 28,33% data loss) on the data?

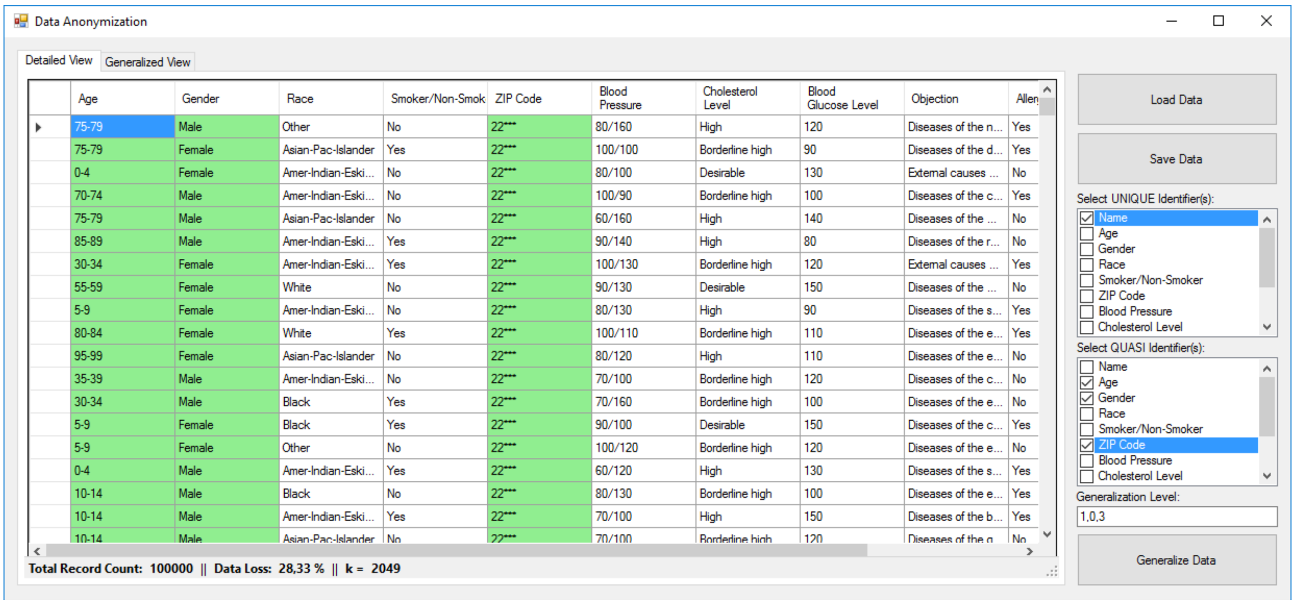
Max. Generalization Level: 4,1,5

k = 500

100% (Node 60 of 60)

Figure 4.4: Calculation of optimal anonymization node by means of OLA algorithm based on anonymity level boundary of $k = 500$

The final outcome of the data anonymization tool is illustrated in Figure 4.5. Within the stated example, the OLA algorithm resulted for anonymization level boundary of $k = 500$, the node [1,0,3] is resulting. Since the quasi-identifier is composed of *Age*, *Gender* and *ZIP Code*, consequential the age is generalized once, the ZIP code three times and the gender not once at all. Therefore, the information loss is about 28% and the anonymization level is at $k = 2049$. Thus, there exist (at least) 2049 (out of 100,000) data records, which are not distinguishable from each other (considering the selected quasi-identifier attributes only).



Data Anonymization

Age	Gender	Race	Smoker/Non-Smok	ZIP Code	Blood Pressure	Cholesterol Level	Blood Glucose Level	Objection	Aller
75-79	Male	Other	No	22***	80/160	High	120	Diseases of the n...	Yes
75-79	Female	Asian-Pac-Islander	Yes	22***	100/100	Borderline high	90	Diseases of the d...	Yes
0-4	Female	Amer-Indian-Eski...	No	22***	80/100	Desirable	130	External causes ...	No
70-74	Male	Amer-Indian-Eski...	No	22***	100/90	Borderline high	100	Diseases of the c...	Yes
75-79	Male	Asian-Pac-Islander	No	22***	60/160	High	140	Diseases of the ...	No
85-89	Male	Amer-Indian-Eski...	Yes	22***	90/140	High	80	Diseases of the r...	No
30-34	Female	Amer-Indian-Eski...	Yes	22***	100/130	Borderline high	120	External causes ...	Yes
55-59	Female	White	No	22***	90/130	Desirable	150	Diseases of the ...	No
5-9	Female	Amer-Indian-Eski...	No	22***	80/130	High	90	Diseases of the s...	Yes
80-84	Female	White	Yes	22***	100/110	Borderline high	110	Diseases of the e...	Yes
95-99	Female	Asian-Pac-Islander	No	22***	80/120	High	110	Diseases of the e...	No
35-39	Male	Amer-Indian-Eski...	No	22***	70/100	Borderline high	120	Diseases of the c...	No
30-34	Male	Black	Yes	22***	70/160	Borderline high	100	Diseases of the e...	No
5-9	Female	Black	Yes	22***	90/100	Desirable	150	Diseases of the c...	Yes
5-9	Female	Other	No	22***	100/120	Borderline high	120	Diseases of the e...	No
0-4	Male	Amer-Indian-Eski...	Yes	22***	60/120	High	130	Diseases of the s...	Yes
10-14	Male	Black	No	22***	80/130	Borderline high	100	Diseases of the e...	Yes
10-14	Male	Amer-Indian-Eski...	Yes	22***	70/100	High	150	Diseases of the b...	Yes
10-14	Male	Asian-Pac-Islander	No	22***	70/100	Borderline high	120	Diseases of the a	No

Total Record Count: 100000 || Data Loss: 28,33 % || k = 2049

Select UNIQUE Identifier(s):
- [x] Name
- [ ] Age
- [ ] Gender
- [ ] Race
- [ ] Smoker/Non-Smoker
- [ ] ZIP Code
- [ ] Blood Pressure
- [ ] Cholesterol Level

Select QUASI Identifier(s):
- [ ] Name
- [x] Age
- [x] Gender
- [ ] Race
- [ ] Smoker/Non-Smoker
- [x] ZIP Code
- [ ] Blood Pressure
- [ ] Cholesterol Level

Generalization Level: 1,0,3

Figure 4.5: Representation of the final outcome of the data anonymization tool by applying the node [1,0,3]

The subsequent listing sums up the necessary steps of the data anonymization tool, as described in this chapter.

1. Input of valid data records
2. Selection of unique- and quasi-identifier attributes
3. Generation of node list (lattice)
4. Calculation of anonymity level (k) and information loss (precision)
5. Set of lower anonymity level boundary
6. Calculation of optimal anonymization by means of OLA algorithm

4.3 Code/Component Access

The data anonymization tool is accessible as an executable software at SUPERCLOUD's project website: `https://supercloud-project.eu/project-results/toolbox`. The tool provided within the *Toolbox* of the project website includes the executable file (.exe) as well as sample data files (.csv) serving as the (plain) data input, which are required by the tool.

4.4 Documentation

As already mentioned, a more detailed description of all included elements of the data anonymization tool can be found in Chapter 13 of SUPERCLOUD's deliverable D3.2 [6].

Chapter 5 Trinocchio

5.1 Introduction

Verifiable computation allows a client to outsource computations, while receiving a cryptographic proof of correctness of the result. Recently, the Pinocchio system achieved faster verification than computation in practice, but Pinocchio and other efficient verifiable computation systems require the client to disclose the inputs to the workers performing the computations, which is undesirable when preserving the privacy of the owners' data is at stake. Trinocchio is a system that distributes Pinocchio to three (or more) workers without the workers learning the inputs that they are computing on. Trinocchio fully exploits the almost linear structure of Pinocchio proofs, letting each worker perform the work for a single Pinocchio proof, while verification by the client remains the same. We created a SUPERCLOUD prototype for this approach and integrated it within the SUPERCLOUD architecture.

5.2 Component Description

In a connected and collaborating system like SUPERCLOUD, the Trinocchio workers are be deployed in the compute resources of the cloud providers. The data that is sent to the workers are the shamir shares of the input (e.g. [[s]], [[t]]) (step 1 in Figure 5.1). Next, the secure multi-party computation component from each of the workers is performing the needed computations on the received input data (step 2). Within the same VM of the cloud provider, the data is transferred as shamir shares to the Pinocchio System for computing shares (e.g. [[y]], $[\pi]$) of the cryptographic proof of the computation (step 3). The cryptographic proof is generated using the evaluation key received from the pinocchio key generation component (step 0). Later the data owner, which outsourced the computation to the workers, reconstructs the results of the requested computations using the shamir secret shares that he received from the workers ([[y]], step 4). For verifying the outsourced computation, the data owner uses the verification key received from the pinocchio key generation component (step 0).

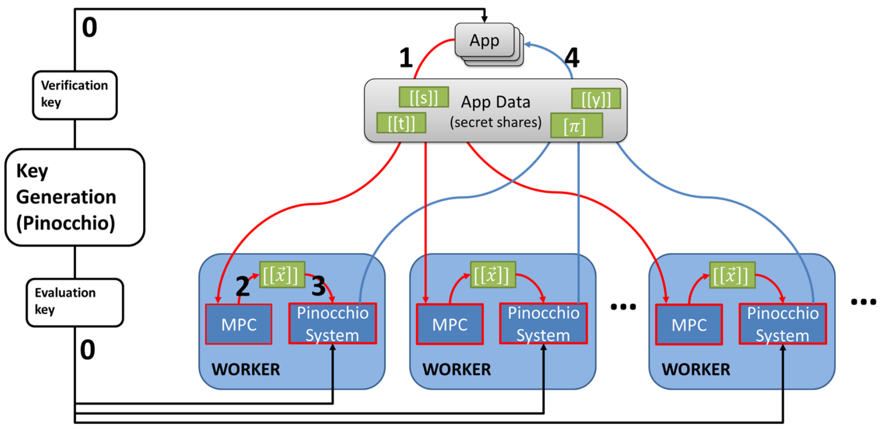


Figure 5.1: Trinocchio data flow

5.2.1 Tools

5.2.1.1 `genkey`

Usage: `genkey <qapfile>` Generate evaluation and verification keys from a given QAP (quadratic arithmetic program); output both to standard output.

5.2.1.2 `lpqap`

Usage: `lpqap <n> <m> <l>` Generates a QAP to prove optimality of the solution to a n-by-m linear program, using bitlength l.
Given a LP (A, b, c) and solution x, p, q to the LP and its dual, the proof consists of, in order: the $(n+1)$-by-$(m+1)$ tableau, consisting of A, b, and c [9]; the values of q, x, and p; a bit decomposition of q; partial sums of $cx-pb$; partial sums of $qb-Ax$; bit decompositions of $qb-Ax$; bit decompositions of x; partial sums of $qc-pA$; bit decompositions of $qc-pA$; and bit decompositions of p.

5.2.1.3 `combine`

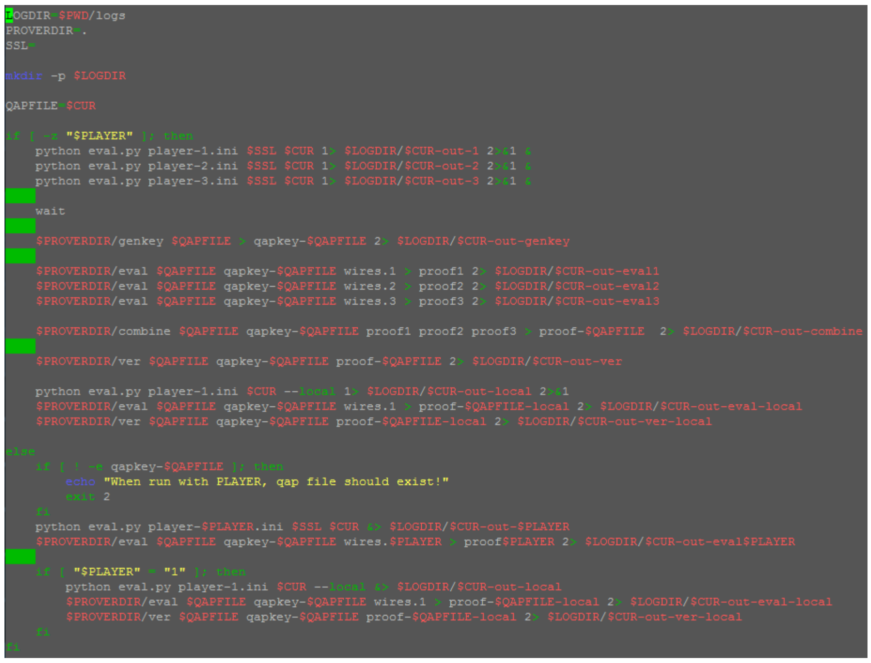
Usage: `combine` Read proofs `proof1`, `proof2`, and `proof3`, and combine them into one overall proof. This combines all blocks from the parts given in the respective proof files.

5.2.1.4 `eval`

Usage: `eval <qap> <evalkey> <wires>` Produces a ZK-QAP proof based on the given wire values and randomness. `<qap>` is a QAP file, and `<evalkey>` an evaluation key produced by `genkey`. `<wires>` contains the values for all wires of the QAP, followed by 19 randomness values: shares of $\delta_{v,i}$ for $i = 1, 2, 3$; shares of $\delta_{w,i}$ for $i = 1, 2, 3$; shares of $\delta_{y,i}$ for $i = 1, 2, 3$; and shares of $\delta_{v,4}$, $\delta_{w,4}$ and $\delta_{y,4}$; and finally, seven shares of zero used to randomise the circuit wires. Prints the proof to standard output.

5.2.1.5 `ver`

Usage: `ver <qap> <evalkey> <proof>` Verify a QAP proof. Currently takes as input an evaluation key because `genkey` does not separately output the verification key, but this could be easily changed.



```
LOGDIR=$PWD/logs
PROVERDIR=.
SSL=

mkdir -p $LOGDIR

QAPFILE=$CUR

if [ -z "$PLAYER" ]; then
    python eval.py player-1.ini $SSL $CUR 1> $LOGDIR/$CUR-out-1 2>&1 &
    python eval.py player-2.ini $SSL $CUR 1> $LOGDIR/$CUR-out-2 2>&1 &
    python eval.py player-3.ini $SSL $CUR 1> $LOGDIR/$CUR-out-3 2>&1 &

    wait

    $PROVERDIR/genkey $QAPFILE > qapkey-$QAPFILE 2> $LOGDIR/$CUR-out-genkey

    $PROVERDIR/eval $QAPFILE qapkey-$QAPFILE wires.1 > proof1 2> $LOGDIR/$CUR-out-eval1
    $PROVERDIR/eval $QAPFILE qapkey-$QAPFILE wires.2 > proof2 2> $LOGDIR/$CUR-out-eval2
    $PROVERDIR/eval $QAPFILE qapkey-$QAPFILE wires.3 > proof3 2> $LOGDIR/$CUR-out-eval3

    $PROVERDIR/combine $QAPFILE qapkey-$QAPFILE proof1 proof2 proof3 > proof-$QAPFILE  2> $LOGDIR/$CUR-out-combine

    $PROVERDIR/ver $QAPFILE qapkey-$QAPFILE proof-$QAPFILE 2> $LOGDIR/$CUR-out-ver

    python eval.py player-1.ini $CUR --local 1> $LOGDIR/$CUR-out-local 2>&1
    $PROVERDIR/eval $QAPFILE qapkey-$QAPFILE wires.1 > proof-$QAPFILE-local 2> $LOGDIR/$CUR-out-eval-local
    $PROVERDIR/ver $QAPFILE qapkey-$QAPFILE proof-$QAPFILE-local 2> $LOGDIR/$CUR-out-ver-local

else
    if [ ! -e qapkey-$QAPFILE ]; then
        echo "When run with PLAYER, qap file should exist!"
        exit 2
    fi
    python eval.py player-$PLAYER.ini $SSL $CUR &> $LOGDIR/$CUR-out-$PLAYER
    $PROVERDIR/eval $QAPFILE qapkey-$QAPFILE wires.$PLAYER > proof$PLAYER 2> $LOGDIR/$CUR-out-eval$PLAYER

    if [ "$PLAYER" = "1" ]; then
        python eval.py player-1.ini $CUR --local &> $LOGDIR/$CUR-out-local
        $PROVERDIR/eval $QAPFILE qapkey-$QAPFILE wires.1 > proof-$QAPFILE-local 2> $LOGDIR/$CUR-out-eval-local
        $PROVERDIR/ver $QAPFILE qapkey-$QAPFILE proof-$QAPFILE-local 2> $LOGDIR/$CUR-out-ver-local
    fi
fi
```

Figure 5.2: Trinocchio experiment qap-2-2 script

5.2.2 Execution example

To reproduce the experiments from [9], the following three experiments, accessible via the link mentioned in section 5.3.

```
sh timings-qap-2-2.sh
sh timings-qap-5-8.sh
sh timings-qap-5-10.sh
```

These scripts generate log files in the `logs` subdirectory from which timing information can be extracted. In the first phase of the first experiment, each of the workers evaluates a random polinomial of degree 2 and with 2 different variables (as depicted in Figure 5.2). Next the key material is generated with the genkey tool and the proofs are generated. Later the proofs are combined into one proof which is verified.

5.3 Code/Component Access

The Trinocchio code is accessible at `https://zenodo.org/record/60295#.WSUyl8akJaQ`. The Trinocchio software component was prototyped and tested on Linux, mingw-w64.For installation the following steps should be followed:

- Installing TUeVIFF-local (`http://security1.win.tue.nl/ meilof/files/verifiability/local.tar.g`
- Installing `ate-pairing` following the instructions from `https://github.com/herumi/ate-pairing`.
- Setting the file locations in the first lines of the `Makefile`
- Compiling need to be updated and compiling is done using the command `make`.
- Testing the Trinocchio component using QAPs of various sizes contained in the Trinocchio distribution: `qap-2-2`, `qap-5-8`, `qap-5-10`.

5.4 Documentation

A more complete description of the system can be found in Chapter 2 of SUPERCLOUD D3.2 [6]. Further documentation about the software is distributed together with the zip file describe above.

Chapter 6 Attribute-based Encryption

6.1 Introduction

SUPERCLOUD-ABE is a Scala cryptographic library which implements an Attribute Based Encryption (ABE) scheme, in which the encryption and decryption are based on some user's attributes. More precisely, in such a system, each end user possesses some characteristic attributes. When uploading a new data, the depositary chooses an access control policy, based on the existing attributes, and related to the data. Then, only users having the set of attributes verifying the defined access control policy will be able to decrypt and read the stored data. Concerning the access structure, the library permits a fine-grained access control with a disjunctive normal form (DNF). More precisely, an access policy is of the form $B_1 \vee B_2 \vee \cdots \vee B_m$, where each B_k is a clause of the form $b_{k,1} \wedge b_{k,2} \wedge \cdots \wedge b_{k,m_k}$, where $b_{k,j}$ corresponds to an attribute.

6.2 Component Description

We have defined, within the SUPERCLOUD-ABE library, a set of classes and methods that can be used to manage a complete ABE system.
At first, there are three main classes that are available within the library.

- `ABEManager`: entity which creates the system and manages attributes and users. It can add or remove an attribute to a member and send him an update private key. For that purpose, it manages a master secret key. Note that the ABE Manager has the knowledge of the member private key.
- `ABEUser`: user with a set of attributes, who can encrypt and decrypt according to its attributes.
- `ABETool`: some utility methods related to the `ABEUser`. This class is mainly useful for tests.

Most classes have `toBinary()` and `fromBinary()` methods to serialize data to/from array of bytes.
We then detail the other classes of the library.

- `ABEAttribute`: it represents an attribute managed by the `ABEManager`. In this library, each attribute label is associated to a random curve point (cf. Chapter 12 of SUPERCLOUD D3.2 [6]). As this is just a library, the attribute configuration should then be done in accordance with the use case for which SUPERCLOUD-ABE will be used.
- `ABEAttributesUniverse`: it represents the set of all the attributes known within the system.
- `ABECipher`: ABE cipher which will be sent in the cloud.
- `ABEManagerElements`: this is the result of `ABEManager` key and element generation. It contains the `ABEManagerPrivateKey`, which must be known ONLY by the `ABEManager`, and `ABEPublicElements` which contains public elements.
- `ABEManagerPrivateElements`: it corresponds to the `ABEManager` private key (a.k.a. as the master secret key).

- `ABEPublicElements`: it contains all the public elements needed by anyone using the system
- `ABEUserPrivateKey`: this is the ABE user's private key, which can only be generated by `ABEManager`.
- `ABEUserPrivateKeyManagerPart`: this corresponds to some elements of `ABEUserPrivateKey` that should be stored by the `ABEManager`. These elements can then be used, later on, in order to update, if relevant, the user's private key. This is relevant if the set of attributes for the corresponding user needs to be changed.
- `AesAndABEencryptedKey`: this is the result of data encryption. It contains a 128 bits symmetric key (typically for AES symmetric cipher) which is used to encrypt the data and the corresponding `ABECipher` to be shared.

The library also contains a set of methods that can be used after its installation. At first, the package `params` contains classes representing keys, ciphers, etc. related to the ABE scheme.

6.3 Code/Component Access

Before going into the installation process, you should request the .jar library by sending an email to `sebastien.canard@orange.com` or `nicolas.desmoulins@orange.com`. Before installing the library, one needs to install a Java virtual machine 7 or earlier. The library and all its dependencies are provided in the `lib` folder. Some Java source code is provided, containing `JUnit` tests and a main method (`JUnit` jar is provided at the root folder and is only necessary for tests).
To compile the test class, one has to execute:

- `javac -cp ./*:lib/* com/orangelabs/crypto/ABETestJ.java`

Then, to run the test class, one has to execute:

- `java -cp ./*:lib/*: com.orangelabs.crypto.ABETestJ`

The sources are also provided, and the compilation can be done with SBT (`http://www.scala-sbt.org/`). The compilation is done by exeuting

- `sbt compile`

Then, the generation of the jar file (stored in target/scala-2.11) is done by executing

- `sbt package`

Finally, the generation of the zip archive containing library and all dependencies (stored in target/universal) necessitates to execute:

- `sbt universal:packageBin`

6.4 Documentation

We provide in Figure 6.1 some code sample that shows how the library can be used in practice.
A more complete description of the system can be found in Chapter 12 of SUPERCLOUD D3.2 [6]. It also provides the way SUPERCLOUD-ABE can be integrated in the SUPERCLOUD framework. Further documentation is also distributed with the library.

```java
import org.junit.Test;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import com.orangelabs.crypto.abe.*;
import com.orangelabs.crypto.abe.params.*;
import com.orangelabs.math.PairingParams;
import com.orangelabs.math.pairing.PairingParametersFp12_BNAte;
import com.orangelabs.tools.ToolsByte;
import com.orangelabs.tools.CipherManager;
import scala.Option;
import scala.Tuple2;
// ...
// Fixed pairing parameters used for crypto (use a BN-256 curve -> ~128 bit security)
PairingParametersFp12_BNAte pairParams = PairingParams.getParams_BN1_pairing();
SecureRandom rand = new SecureRandom();
// Define attribute universe
static String[] attU = new String[] { "team1", "team2", "team3", "project1", "project2", "boss" };
ABEManagerElements elems = ABEManager.generateParameters(pairParams, "SHA-256", rand);
ABEManagerPrivateElements abePriv = elems.privElem();
ABEPublicElements pubElem0 = elems.pubElem();
// also test binary serialization of ABEPublicElements
ABEPublicElements pubElem = ABEPublicElements.fromBinaryJ(pairParams, pubElem0.toBinary());
// create instance of ABEManager
ABEManager abeManager = new ABEManager(abePriv, pubElem);
// add attributes
abeManager.addAttributesToUniverse(attU);
// The ABEUserPrivate will have to be sent to the user (in a confidential way!).
// The manager will have to keep the ABEUserPrivateKey and the ABEUserPrivateKeyManagerPart, which
// are needed to add (or remove) new attributes to a user.
Tuple2<ABEUserPrivateKeyManagerPart, ABEUserPrivateKey> user1Keys = abeManager.generateUserSecretKey(
"user1", new String[] { "boss", "project1" }, rand);
ABEUserPrivateKeyManagerPart manUser1 = user1Keys._1();
ABEUserPrivateKey user1PrivKey0 = user1Keys._2();
// also test private key binary serialization
ABEUserPrivateKey user1PrivKey = ABEUserPrivateKey.fromBinaryJ(pairParams, samPrivKey0.toBinary());
Tuple2<ABEUserPrivateKeyManagerPart, ABEUserPrivateKey> user2Keys = abeManager.generateUserSecretKey(
"user2", new String[] { "team1", "team2", "project1" }, rand);
ABEUserPrivateKeyManagerPart manUser2 = user2Keys._1();
ABEUserPrivateKey user2PrivKey = user2Keys._2();
// create ABEUser instances (normally on each client side)
ABEUser user1 = new ABEUser(user1PrivKey, pubElem);
ABEUser user2 = new ABEUser(user2PrivKey, pubElem);
// encrypt some data
// define access policy: boss or (team1 & team2)
String[][] accessPolicy = new String[][] { new String[] { "boss" }, new String[] { "team1", "team2" }};
AesAndABEencryptedKey resEnc = user1.genEncryptionKeys(accessPolicy, rand);
byte[] aesKey1 = resEnc.aesKey(); // aes key in the form of a 16-bytes array.
ABECipher abeCipher1 = resEnc.abeCipher();
// AES key must be used to encrypt the data
// ECB mode for AES should not be used. Other modes like CBC are better. But in this case
// the initialization vector must be a fixed value in the system.
// It is not a problem here because each file or set of data to encrypt will use a random AES key.
//
// The class CipherManager can help for AES encryption but is not mandatory.
byte[] data = ... // contain the data to encrypt
byte iv = new byte[16]; // 16 bytes '0' array can be used for initialization vector.
byte[] aesEncData = CipherManager.encrypt(data, aesKey1,iv);
// abeCipher1 is the corresponding ABECipher which can be shared on cloud with the data encrypted with AES
// also test serialization of cipher
ABECipher abeCipher1Enc = ABECipher.fromBinaryJ(pairParams, abeCipher1.toBinary());
Option<byte[]> aesKey1Dec = user2.decryptAESKey(abeCipher1Enc);
if (aesKey1Dec.isDefined()) {
byte[] key = aesKey1Dec.get();
// key is normally identical to aesKey1 and can be used to decrypt the data encrypted with AES.
byte[] decData = CipherManager.decrypt(data, key, iv); // using CipherManager class
} else {
// if we are here it means the user had not the appropriate attribute(s).
}
```

Figure 6.1: SUPERCLOUD-ABE code example

Chapter 7 Conclusions

This report described the main software components produced in WP3. These components include most of the research contributions produced in this work package. These components instantiate several elements described in SUPERCLOUD data management architecture (see Figure 7.1), namely: Janus can be used either through data accessors (DA) or proxies, exploiting cloud storage services; Hyperledger, Trinochio, and SUPERCLOUD-AB must be instantiated as servers on top of SUPERCLOUD VMs, tightly integrated on distributed cloud applications; finally the K-anonymization tool can be used in the client side to remove privacy-sensitive data from datasets to be stored in non-trusted clouds.

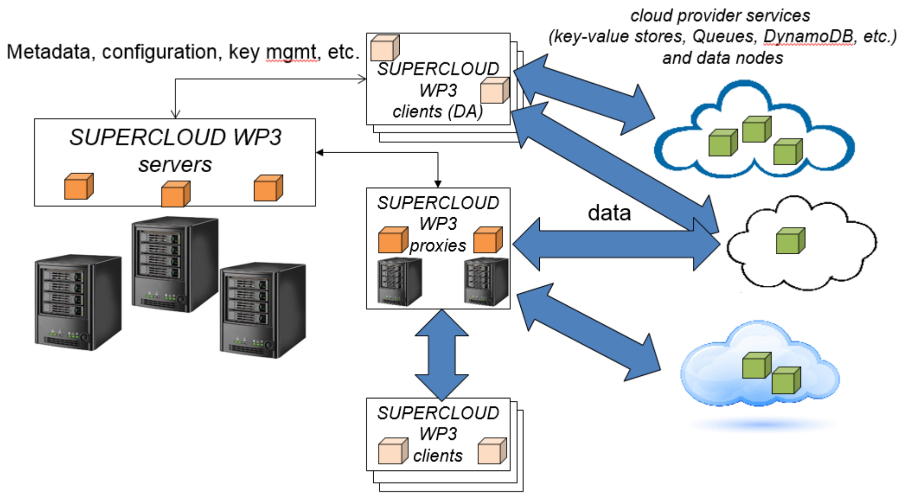

cloud provider services
(key-value stores, Queues, DynamoDB, etc.)
and data nodes
Metadata, configuration, key mgmt, etc.
SUPERCLOUD WP3 clients (DA)
SUPERCLOUD WP3 servers
SUPERCLOUD WP3 proxies
data
SUPERCLOUD WP3 clients

Figure 7.1: SUPERCLOUD data management architecture.

All these components are implemented on the operating system level or above, therefore they can be easily integrated with the VMs and containers being developed in the SUPERCLOUD project (WP2 and WP4). This integration will be demonstrated in the use cases under development in the project (WP5).

Bibliography

[1] Alysson Bessani, Joao Sousa, and Eduardo Alchieri. State machine replication for the masses with BFT-SMaRt. In *Proc. of the IEEE/IFIP International Conference on Dependable Systems and Networks – DSN 2014*, June 2014.

[2] Christian Cachin, Simon Schubert, and Marko Vukolić. Non-determinism in byzantine fault-tolerant replication. In *20th International Conference on Principles of Distributed Systems, OPODIS 2016, December 13-16, 2016, Madrid, Spain*, pages 24:1–24:16, 2016.

[3] K. El Emam, F. K. Dankar, R. Issa, E. Jonker, D. Amyot, E. Cogo, J. P. Corriveau, M. Walker, S. Chowdhury, R. Vaillancourt, T. Roffey, and J. Bottomley. A globally optimal k-anonymity method for the de-identification of health data. *Journal of the American Medical Informatics Association*, 16(5):670–682, 2009.

[4] Shengyun Liu, Paolo Viotti, Christian Cachin, Vivien Quéma, and Marko Vukolić. XFT: practical fault tolerance beyond crashes. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016.*, pages 485–500, 2016.

[5] Mario Münzer, Sébastian Canard, Marie Paindavoine, Alysson Bessani, Caroline Fontaine, Krysztof Oborzyńsky, Meilof Veeningen, and Paulo Sousa. D3.1 - Architecture for Data Management. *SUPERCLOUD*, 2015.

[6] Mario Münzer, Sébastian Canard, Marie Paindavoine, Andre Nogueira, Antonio Casimiro, João Sousa, Joel Alcântara, Tiago Oliveira, Ricardo Mendes, Alysson Bessani, Christian Cachin, Simon Schubert, Caroline Fontaine, Daniel Pletea, Meilof Veeningen, and Jialin Huang. D3.2 - Specification of Security Enablers for Data Management. *SUPERCLOUD*, 2016.

[7] Andre Nogueira, Antonio Casimiro, and Alysson Bessani. Elastic state machine replication. *IEEE Transactions on Parallel and Distributed Systems*, March 2017. Accepted for publication.

[8] Tiago Oliveira, Ricardo Mendes, and Alysson Bessani. Exploring key-value stores in multi-writer byzantine-resilient register emulations. In *Proc. of the 20th International Conference On Principles Of DIstributed Systems – OPODIS?16*, December 2016.

[9] Berry Schoenmakers, Meilof Veeningen, and Niels de Vreede. Trinocchio: Privacy-preserving outsourcing by distributed verifiable computation. In *Applied Cryptography and Network Security - 14th International Conference, ACNS 2016, Guildford, UK, June 19-22, 2016. Proceedings*, pages 346–366, 2016.

[10] Joao Sousa and Alysson Bessani. Separating the WHEAT from the chaff: An empirical design for geo-replicated state machines. In *Proc. of the 34th International Symposium on Reliable Distributed Systems – SRDS'15*, September 2015.

[11] Latanya Sweeney. K-anonymity: A model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):557–570, October 2002.

[12] Marko Vukolić. The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication. In *Open Problems in Network Security - IFIP WG 11.4 International Workshop, iNetSec 2015, Zurich, Switzerland, October 29, 2015, Revised Selected Papers*, pages 112–125, 2015.

[13] Marko Vukolić. Rethinking permissioned blockchains. In *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*, BCC '17, pages 3–7, 2017.