Grant Agreement Number: 825225

Safe-DEED

www.safe-deed.eu

Protocols for Privacy-Preserving Data Analytics and Secure Lead-Time Based Pricing v1/2

Deliverable number	*D5.4*
Dissemination level	*Public*
Delivery data	*due 29.05.2020*
Status	*Final*
Authors	*Lukas Helminger, Fabian Schmid*

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825225.

Changes Summary

Date	Author	Summary	Version
08.05.2020	Lukas Helminger, Fabian Schmid	First Draft (for internal review)	0.1
15.05.2020	Alexandros Bampoulidis	Internal Review	0.2
18.05.2020	Evangelos Kotsifakos	Internal Review	0.3
25.05.2020	Lukas Helminger, Fabian Schmid	Final Version (incorporated reviews)	1.0

Executive Summary

This deliverable D5.4 - Protocols for privacy-preserving data analytics and secure lead-time based pricing v1 - is the second software result of Safe-DEED's WP5 in the form of a demonstrator. It is an outcome of the task T5.2 - Specialized protocols. In addition, to the software libraries, this deliverable comprises a document that describes suitable protocols for the use-cases in Safe-DEED's WP6 and WP7. To achieve the privacy requirements in both use-cases latest cryptographic primitives are used - in particular - Private Set Intersection and Multi-Party Computation. The provided demonstrator is a preliminary one, although it is already operative. In the final version, the functionality, as well as the software documentation, will be extended.

Table of Contents

1 Introduction **5**
1.1 Related Documents . . . 5
1.2 Roadmap . . . 5

2 Privacy-Preserving Data Analytics **5**
2.1 Motivation . . . 5
2.1.1 WP6 Task Description . . . 6
2.1.2 Use-Case . . . 6
2.2 Private Set Intersection (PSI) . . . 6
2.3 Demonstrator . . . 8
2.3.1 Design Choices . . . 8
2.3.2 Resources . . . 9
2.4 Performance . . . 9

3 Secure Lead-Time Based Pricing (SLTBP) **10**
3.1 Motivation . . . 10
3.1.1 Use-Case . . . 10
3.2 Multi-Party Computation (MPC) . . . 11
3.3 FRESCO . . . 11
3.3.1 Project Structure . . . 11
3.3.2 The FRESCO core . . . 12
3.3.3 Protocol Suites . . . 12
3.4 Demonstrator . . . 12
3.4.1 Initialization . . . 12
3.4.2 The PriceFinder . . . 13
3.4.3 The Process Flow . . . 13
3.4.4 Secure Channel . . . 13
3.5 Running the Demonstrator . . . 14
3.5.1 Resources . . . 14

4 Conclusion **14**

5 References **14**

List of Figures

Fig.1 PSI with ZIP Codes 7
Fig.2 PSI Benchmarks 9
Fig.3 MPC Protocol for SLTBP 11

Abbreviations

AES	Advanced Encryption Standard
ATP	Available to Promise
CRM	Customer Relationship Management
FRESCO	Framework for Efficient and Secure Computation
GUI	Graphical User Interface
MPC	Multi-Party Computation
PET	Privacy-Enhancing Technology
PSI	Private Set Intersection
SLTBP	Secure Lead-Time Based Pricing

1 Introduction

The purpose of this deliverable is to report on the cryptographic protocols developed for privacy-preserving data analytics and secure lead-time based pricing for use in WP6 and WP7. It provides a demonstrator consisting of two independent software libraries. This document accompanies the libraries with a description of their functionality and the context to Safe-DEED.

1.1 Related Documents

D5.4 is heavily based on the research done in WP5. More concretely, the insights obtained by D5.1, D5.2, and D5.3 were crucial in the design of the protocols in D5.4. On the other hand, D5.4 will serve as the basis for the security protocols used in the demonstrators of WP6 (D6.2) and WP7 (D7.4). Although not required in the agreement, there will be an additional demonstrator for ”hands-on” experimentation. D5.4 also contributes to this demonstrator.

1.2 Roadmap

In Section 2, we first give a motivation for privacy-preserving data analytics in Safe-DEED. After introducing the general idea behind private set intersection (PSI), we give a description of this part of the demonstrator. In Section 3, we explain the use-case of WP7 and the resulting demonstrator.

2 Privacy-Preserving Data Analytics

The main objective of this part of the demonstrator is to develop protocols for privacy-preserving data analytics. In Section 2.1, we will look at the specific motivation for privacy-preserving data analytics in Safe-DEED. In addition, we describe the process of finding the right privacy-enhancing technology (PET) for the use-case in WP6. Thereafter, we give a high-level description of the chosen secure computation protocol (Section 2.2). After that, we are going into the specification of the demonstrator and explain the design choices (Section 2.3). In the end, we report in Section 2.4 the outcomes of the first performance experiment.

2.1 Motivation

We take a brief look at WP6 to better understand the context of the demonstrator. The objective of this work package is to demonstrate that industries can benefit from securely exchange data between companies to perform big data analytics and correlations. We have to develop secure methods that protect the privacy of individuals since companies need to share data that might contain personal information or reveal a company’s activities or strategies. Note that Safe-DEED deliverable D3.2 discusses ”Legal and Ethical Requirements for Personal Data Use Case”.

2.1.1 WP6 Task Description

- Joint data usage within corporate environments: Strict rules often hinder the free exchange of data between different departments (e.g., between marketing and strategy department). Currently, there is a lack of protocols for securely sharing and computing on joint data. Solving this issue could lead to more efficient business decisions.
- Joint data usage between different enterprises in the same domain: A joint analysis of two competitors could be beneficial to both (e.g., joint market analysis). Understandably, the main obstacle here is trust. Also, there is a fear of data leakage in such a collaboration. Therefore, there is a strong need for protocols that are trustworthy by design.
- Joint data usage between different enterprises in different domains: Even if a company owns a large amount of data, these data are highly likely to be very specific. Additional data can be enriched with external data to provide useful insights that will help to choose the proper and most efficient business strategies. So protocols for computation on joint databases with external entities are of great importance.

2.1.2 Use-Case

There was a fruitful exchange of ideas between WP5 and WP6 about concrete use-cases arising from the three tasks described above. The outcome of these discussions was to focus on the intersection of datasets from different parties. More concretely, there is a business interest for two enterprises to find out geographic areas where they both have customers. For instance, such an area can then be targeted by a joint marketing campaign. More generally, this new knowledge can lead to better alignment of common business interests in these areas.

The privacy requirement here is that both enterprises only learn the areas where they both have customers. In contrast, an enterprise should not learn any area where it has no customers, but the other enterprise does. In other words, the input of the enterprises - in this computation - should be kept private. The only information available two both enterprises should be the output. Such functionality can be achieved by Multi-Party Computation (MPC).

In the Safe-DEED deliverable D5.1 we have introduced ”Requirements for secure computations on large datasets with multiple owners”. It discusses the principles and requirements of MPC. Here, we will use a particular MPC protocol called Private Set Intersection (PSI) protocol to accomplish the privacy guarantees outlined in the paragraph above.

2.2 Private Set Intersection (PSI)

PSI is an MPC-protocol that allows two parties to jointly compute the intersection of their datasets. Thereby, neither party learns information from the protocol execution except for the elements in the intersection. In the rest of this section, we explain the high-level idea of PSI protocols. Please note that this description is simplified. State-of-the-art protocols use more sophisticated techniques, mainly for performance reasons.

For this description of PSI, we take two enterprises that want to compute the intersection of their CRM data. For simplicity, we assume that their datasets of customers consist of only one column - the ZIP code of their customers. The objective of PSI in this scenario is to find

the common ZIP codes of the enterprises. The computation can be split into five phases and is depicted in Figure 1.

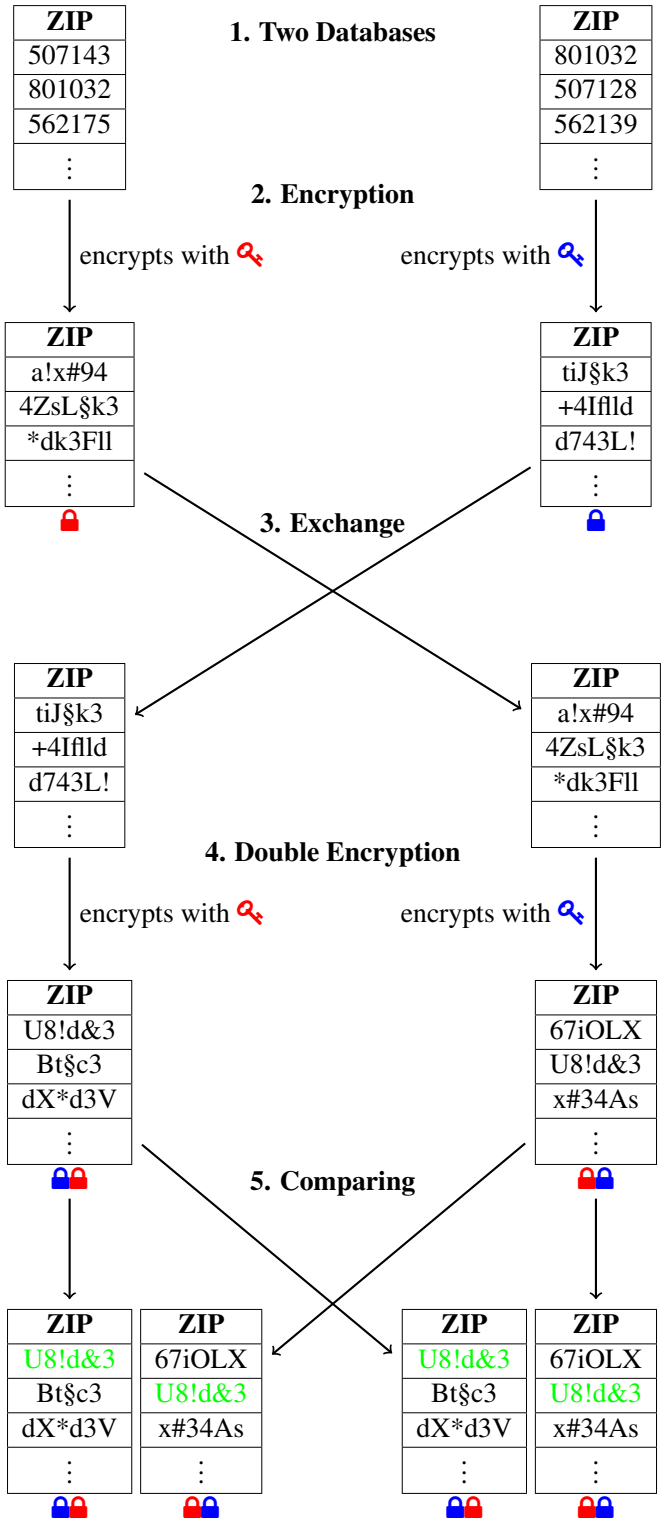

Figure 1: PSI with ZIP Codes

1. **Two Datasets:** Each enterprise has its own table of ZIP codes (corresponding to customers)

2. **Encryption:** The actual computation starts when both enterprises encrypt their data. For encryption, they use similar schemes as AES. Encryption with private keys ensures that the dataset is not decipherable to anyone else. Note that the encryption scheme has to be commutative, i.e., if one encrypts a message with two different keys it does not matter which key is used first.

3. **Exchange:** In this next step, each enterprise sends its encrypted table of ZIP codes to the other enterprise. This is not a privacy risk since the other enterprise is not in possession of the private key. Therefore it can not decipher the table. Encryption schemes are designed in such a way that a brute-force attack is computationally not feasible.

4. **Double Encryption:** Now, both enterprises encrypt the just received table with their private key from phase 2. So each of the tables is encrypted twice. Since the encryption scheme is commutative, the order of encryption does not matter. Therefore the tables are comparable.

5. **Comparison:** After exchanging the double encrypted tables, both enterprises can compare the entries. Each match indicates that they have a ZIP code in common. More precisely, the index where this match occurs tells which ZIP code they have in common. The common entries then get encrypted with the respective private key.

2.3 Demonstrator

This part of the demonstrator consists of a Java PSI library. For the core functionality, we rely on a state-of-the-art PSI protocol [4]. The protocol is secure against a malicious client and a semi-honest server. For encryption, the protocol uses symmetric encryption schemes specially designed for MPC. This allows a significant better throughput as elaborated in Safe-DEED's deliverable D5.2 "Low Complexity Primitives". Note that the core of the PSI protocol is written in C++ for performance reasons.

2.3.1 Design Choices

There have been two major objectives during the development of the PSI library. First, the interface should be as simple as possible. We want that the integration of our PSI library does not require any background in cryptography or privacy. Thereby, we hope to lower the barrier for developing privacy-enhanced applications. Secondly, the library should run on Windows as well as Linux. This, again, should increase the audience for our PSI library.

In this deliverable, we do not provide a graphical user interface (GUI). Our PSI library can be run as a command-line program. In addition, the library will be integrated into a demonstrator in WP6. This demonstrator will cover the business use-case described in Section 2.1. This is further proof that the library is easy to integrate into applications.

Figure 2: PSI Benchmarks

2.3.2 Resources

As already mentioned, this report gives the context of the demonstrator. The source code is available at `https://github.com/Safe-DEED/PSI`. There you also find all the information required for installation and additional documentation.

2.4 Performance

In order to evaluate the performance of our PSI protocol, we decided to compare it to a different Java PSI implementation. We chose to compare our PSI protocol to the implementation found at `https://github.com/aicis/fresco/tree/master/demos/psi`. This PSI implementation uses the general-purpose MPC-framework FRESCO [1] as a building block. The outcomes of this comparison are depicted in Figure 2.

The specifics of the experiment are the following: Both parties posses lists of the same size $(10, 50, 100, 200, 500, 1000, 5000)$, where the last two set sizes were only tested with our implementation (due to performance reasons). The runtime is in seconds and was averaged over 10 executions. We ran our benchmarks on a Linux laptop (Intel® Core™ i5-6200U CPU @ 2.30GHz $\times$ 4) with 12 GB RAM available.

If we look at Figure 2, we see that both protocols scale similarly. However, our implementation is about two magnitudes faster. We believe that this performance difference is two-fold. First, the core of our protocol is written in C++. In addition, we do not rely on a general-purpose MPC-framework but, instead, use optimizations designed for PSI.

3 Secure Lead-Time Based Pricing (SLTBP)

In Section 3.1, we explain the use-case from WP7. Before we are going into the technical description of this part of the demonstrator (Section 3.4), we take a look at the underlying MPC framework for our demonstrator in Section 3.3.

3.1 Motivation

We take a brief look at WP7 to understand the context of the demonstrator better. The goal of this work package is to integrate MPC technology into an existing order management platform where customers of a manufacturer can purchase unallocated Available To Promise (ATP) (stock that has yet to be allocated/sold to a customer) from the manufacturer privately and securely. Details on ATP and lead-time are discussed in Safe-DEED deliverable D7.1 ”Algorithm which allows for extraction of the data”.

Most of the data used in WP7 is non-personal data. Nevertheless, it is of great importance to keep the data secure and private from a business perspective. Note that Safe-DEED deliverable D3.3 discusses ”Legal and Ethical Requirements for non-Personal Data Use Case”.

WP5 and WP7 agreed on extending the use-case described in the agreement. We aim at a solution that allows multiple customers to purchase unallocated ATP from a manufacturer simultaneously. Thereby, we use the full potential of MPC. In addition, we solve a further issue. Small companies (e.g., start-ups) are often not allowed to purchase directly from the manufacturer because they request too few stocks. By allowing customers to aggregate their order, they are able to buy directly from the manufacturer. Usually, this has the effect of lower prices and faster delivery - both crucial to a start-up.

3.1.1 Use-Case

The goal of this part of the demonstrator is to enable small companies to buy from a much larger enterprise as depicted in Figure 3. With MPC, small companies would be able to keep their requested amount secret from their competitors. Even the vendor will only discover the required items and suggested prices when the protocol finishes with success.

In the first version, every customer submits their requested amount and suggests a price. Then everyone participates to secretly compare the required amounts with the available stock of the vendor. In parallel, all participants determine the average offered cost per unit and weigh it to the minimum price of the vendor. If the offers meet the prerequisites of a deal, everyone reveals their distinct amount and the total cost of the order to the vendor. The clients only learn whether the transaction was possible or not, even if they try to attack the protocol.

As a technical foundation for our demonstrator, we use FRamework for Efficient and Secure COmputation (FRESCO). This framework provides the necessary functionality to build new MPC protocols such as ours. We did not only want to write our protocol but also provide a starting point for future work. Providing a starting point seemed necessary to us, since setting up security against malicious adversaries in FRESCO can be quite tedious.

Since FRESCO is available in the public maven repository, we also developed our extension using maven[1]. The use of maven assures ease of integration. In the end we want to have

[1] `https://maven.apache.org/`

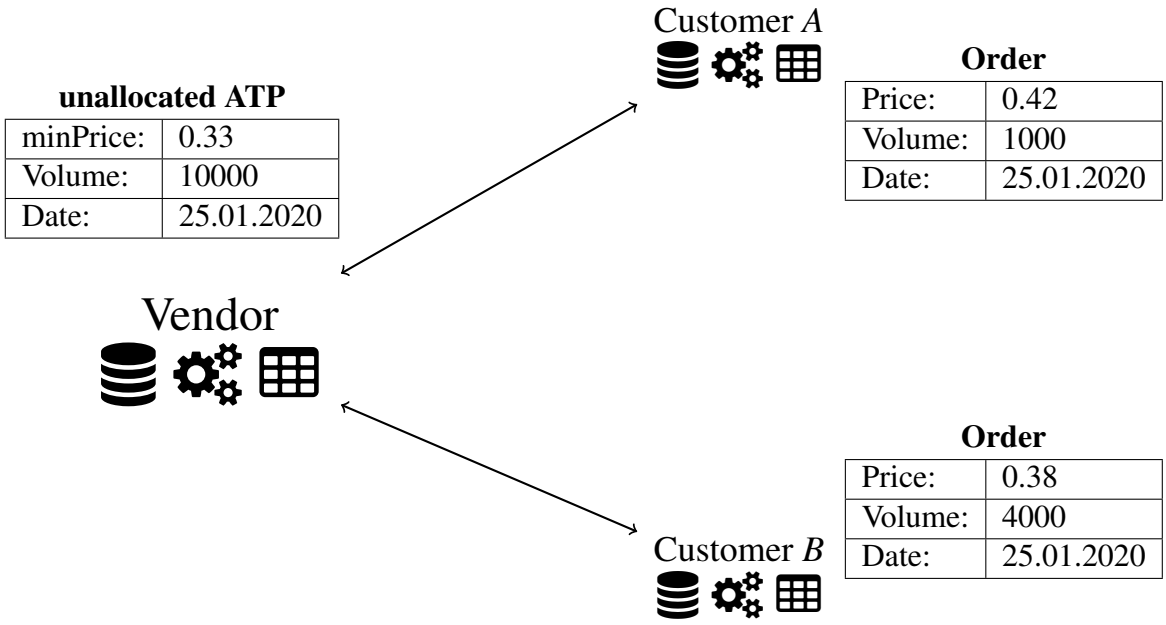

Figure 3: MPC Protocol for SLTBP

everything assembled into a single jar with dependencies called the priceFinder. Parameters on startup define whether the priceFinder acts as the host or the client.

3.2 Multi-Party Computation (MPC)

Multi-Party Computation (MPC) is a technique from cryptography. It enables multiple parties to perform joint data analysis. Thereby, the input data of each party is not revealed - only the result of the data analysis is shared among the involved parties. The goal of MPC is to do data analytics with multiple data providers in a privacy-preserving way. For a more in-depth explanation and several illustrative examples, we refer to Safe-DEED deliverable D5.1 "Requirements for secure computations on large datasets with multiple owners".

3.3 FRESCO

FRESCO can be understood as our main computation engine. We need to setup all the parameters needed and also provide some initialization code to fully enable active security. Having done this groundwork, this demonstrator can now also be used to kick-start other, actively secure, MPC projects, since these steps are use-case independent and can be reused. We now give a quick overview of the main components of FRESCO.

3.3.1 Project Structure

FRESCO is available as a maven project and as a docker image[2]. Including the framework as a maven dependency, as in our case, only takes several lines of code. The framework is separated

[2] https://www.docker.com/

into two main parts, the core project and the protocol suites, which are both separate maven projects. This design allows for additional protocol suites to be developed.

3.3.2 The FRESCO core

FRESCO core provides the general functionality needed to run an MPC program. The main entry point to the library is the `SecureComputationEngine`. This class is the main driver of the computation. Here all the MPC protocols are stored and evaluated over the network. This class heavily relies on the `ResourcePool`, which serves as a container for runtime variables and objects. Specific protocol suites need to implement their version of the `ResourcePool`, so they have all the state variables they need. Finally, there is the protocol suites interface, which has to be implemented externally.

3.3.3 Protocol Suites

There are many known techniques to implement MPC. These techniques are generally known as secure computation protocols. Since there are often a lot of these protocols required for a specific strategy, FRESCO implements all sub-protocols which are related together in so-called protocol suites. These protocol suites contain the code to enable the specific cryptographic technique used to ensure secure multi-party computation.

3.4 Demonstrator

This demonstrator implements the use-case described above, using the both the FRESCO core and SPDZ protocol suite [3, 2]. The project consists of two main entry points, which will, from now on, be referred to as host and client (having no network implications). Both of these can be executed by the main function in the priceFinder, depending on startup parameters. We will elaborate on this design desicion later in this chapter.

3.4.1 Initialization

Upon start, the user can give some customizing input, determining some computation strategies for the application. In the current state, the input necessary is given in the Makefile, and a Command-Line Parser class takes care of it, inside of the demonstrator. The data structure created by this Parser class can easily be generated from different sources, e.g., configuration files from databases. Next to configuration, the input of the program also consists of a price, a volume and a date for each client and the host.

- **Price** The price either indicates the amount of money a client is willing to pay per each unit or, in the case of the host, it indicates the minimum amount of money expected per unit.
- **Volume** The amount that is either requested or provided by a given date.
- **Date** in a later enhancement, several selling windows with different delivery date options are possible. The date either indicates the date of order for the client or indicates the date from which on the units are available.

After creating the required data structure, the initialization begins. This initialization essentially has two main parts. Firstly, the components necessary for the application (i.e., this specific execution) need to be set. This is done by setting a date, price, volume, and also starting up the network and sharing some information with the other participants. Secondly, there are some things to be done to start the FRESCO. Generally, the framework can be started rather quickly - all the needed code can also be found in their demo project. Still, these demos are all not actively secure. They do not use the secure SPDZ protocol but instead they have an insecure method of generating the multiplication triples. To fix this issue, we need to include the MASCOT [5] preprocessing strategy, which requires some boilerplate code for its initialization. After having initialized both the application as well as the framework, we can start our application by running it in the `SecureComputationEngine`.

3.4.2 The PriceFinder

To have a cleaner interface for our demonstrator project, we wanted to be able to start the project as a host or as a client. This led to the executable class of the price Finder. Following up on discussions with our partner, we wanted to be able to quickly edit the logic of the price agreement. In this class we define the price finding method and pass it to our framework. This way, we aim to make this method easily accessible without having to dig into our framework code.

3.4.3 The Process Flow

When running our application, we pass the instance to the `SecureComputationEngine` in the `runApplication` function, alongside our previously instantiated protocol suite and `ResourcePool` objects. Every class implementing the application interface, needs to have a `buildComputation` function. This function is called by the framework and used to build the MPC protocol. Building the protocol is achieved by returning an arbitrary number of lambda functions, which will be executed consecutively later on. In other words, in `buildComputation`, the code of our application is submitted to the framework. This is done because the real computation has to be done over the network, and the function calls to the framework have to be replaced by native protocols.

The logic for this specific application can mostly be found in the `ATPManager`, it also contains the `ATPUnit` subclass. A single `ATPUnit` contains all the info about one order. The stocks provided by the host are also stored as single units per date. This abstraction allows for quick sorting of secretly shared values and to simplify operations and storage.

3.4.4 Secure Channel

After implementing the WP7 specific use-case, we wanted to extract as much functionality as possible into a framework. This framework can be used to prototype FRESCO projects quickly. On the one hand, we already described that a lot of boilerplate code is necessary to enable the security of SPDZ. On the other hand, we provide some network capabilities based on the Java Cryptographic extension. It is, therefore, possible to have a TCP connection with the other parties in the protocol. The Sender and Receiver classes can be handed to the internal structure,

such that FRESCO only uses this authenticated and encrypted communication channel. As it best fits our use-case, at the moment the certificates are distributed beforehand.

3.5 Running the Demonstrator

We tested our protocol using different settings. Therefore the project comes with several predefined demonstrator setups. In all of these there is one party as the host and several parties as clients started in parallel. All of these processes work in different sub directories on the same machine. The Makefile, which provides these demonstration setups, also states the inputs for all the clients. Only the host process reads its input from a json data file. In the end one can see the output of the individual processes in the log file of their respective directory.

3.5.1 Resources

This part of the demonstrator is also open source. It is available at `https://github.com/Safe-DEED/SLTBP`. There you find additional information required for installation and further documentation.

4 Conclusion

In this deliverable, we have presented protocols for privacy-preserving data analytics and secure lead-time based pricing. Besides the implementations of the protocols, this demonstrator provides descriptions, illustrations, and context with regard to Safe-DEED. Despite being an initial version, all protocols are already operative and have been tested on Linux as well as Windows. The first benchmarks leave us confident that the developed technology is suitable for large data sets. On the one hand, we will continue our research for scalable privacy-preserving protocols. On the other, we aim at broadening the suite of protocols in the final version of the demonstrator. We will also intensify our collaboration with the use-case partners to guarantee a smooth integration of our components.

5 References

[1] Alexandra Institute. FRESCO - a FRamework for Efficient Secure COmputation, 2019.

[2] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In *ESORICS*, volume 8134 of *LNCS*, pages 1–18. Springer, 2013.

[3] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, volume 7417 of *LNCS*, pages 643–662. Springer, 2012.

[4] Daniel Kales, Christian Rechberger, Thomas Schneider, Matthias Senker, and Christian Weinert. Mobile private contact discovery at scale. In *USENIX Security Symposium*, pages 1447–1464. USENIX Association, 2019.

[5] Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: faster malicious arithmetic secure computation with oblivious transfer. In *ACM Conference on Computer and Communications Security*, pages 830–842. ACM, 2016.