Alexander Grass, BSc

Evaluation of Private Selective Aggregation in the Web and its Application in Real-World Scenarios

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisors

Univ.-Prof. Dipl.-Ing. Dr.techn. Christian Rechberger

Dipl. Ing. Lukas Helminger, BSc

Institute of Applied Information Processing and Communications

Graz, March 26, 2021

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date, Signature

Acknowledgements

Throughout the writing of this thesis, I have received a great deal of support.

First, I would like to thank my mentor Lukas Helminger who was always open to any questions I had along the way. Your feedback helped me to identify a lot of sloppiness I initially introduced into the thesis. Additionally, it pushed me to further reflect on my way of scientific thinking. Thank you for the time you invested in my issues.

I would also like to thank Christian Rechberger, who quickly helped me out in tough times, especially during the Corona pandemic. I would have gone through a lot of struggle if it weren't for you. I acknowledge your kindness and the fact that you kept being open to my requests even in busy times.

Next, I would like to thank Nick Angelou, who is the maintainer of the node-seal library. You were helping me out, no matter what problem I faced in your library. I would have had such a hard time implementing all those additional features I needed. Thanks for all the support. I hope to see you someday to thank you personally.

I would like to thank Chiara Letter, who was proof-reading my thesis patiently and pointed out a whole bunch of improvements. Moreover, I would like to thank you for your mental support during this time. A lot of issues stacked, and it would have been so much harder without your assistance.

Last, I would like to thank the rest of my friends and my family who were there for me during this time. It would have been hard to relax my mind after hours of research without all of your support.

Abstract

Although information security is increasingly becoming the focus of society and many problems have been solved in recent years, there are still many critical areas where appropriate solutions are lacking. Securely retrieving data from a server is a problem that has been frequently addressed in the past. The main focus of common schemes such as Private Information Retrieval (PIR) or Oblivious Transfer (OT) is to hide the client's request. However, these schemes do not aim to protect individual entries in the database, although securely retrieving information about individual items could help construct entirely new systems. Especially in the cloud, we have seen tremendous data growth in recent years, leading to the need for new systems to emerge that process data efficiently and, most importantly, securely. Such systems could be used to build information service platforms that enable information sharing without violating individual privacy. In this work, we evaluate Private Selective Aggregation (PSA), a protocol for securely querying aggregated data that targets both client, server, and individual privacy. We provide a generic library to help build PSA web applications and study their performance. In addition, we present business problems that can be solved using PSA and our library, and provide a Proof of Concept for two of the cases. We also examine various parameters of these applications and provide recommendations for the correct settings.

Keywords: Information Security, Homomorphic Encryption, Differential Privacy, Private Selective Aggregation

Kurzfassung

Obwohl die Informationssicherheit zunehmend in den Fokus der Gesellschaft rückt und viele Probleme in den letzten Jahren gelöst wurden, gibt es immer noch viele kritische Bereiche, in denen geeignete Lösungen fehlen. Das sichere Abrufen von Daten auf einem Server ist ein Problem, das in den letzten Jahren häufig adressiert wurde. Der Schwerpunkt gängiger Schemata wie Private Information Retrieval (PIR) oder Oblivious Transfer (OT) liegt darin, die Anfrage des Clients zu verbergen. Diese Schemata zielen jedoch nicht darauf ab, einzelne Einträge in der Datenbank zu schützen, obwohl der sichere Abruf von Informationen über einzelne Elemente helfen könnte, völlig neue Systeme zu konstruieren. Besonders in der Cloud haben wir in den letzten Jahren ein enormes Datenwachstum erlebt, was dazu führt, dass neue Systeme entstehen müssen, die Daten effizient und vor allem sicher verarbeiten. Solche Systeme könnten zum Aufbau von Informationsdienstplattformen verwendet werden, die den Austausch von Informationen ermöglichen, ohne die Privatsphäre des Einzelnen zu verletzen. In dieser Arbeit evaluieren wir Private Selective Aggregation (PSA), ein Protokoll zur sicheren Abfrage von aggregierten Daten, das sowohl die Privatsphäre des Clients und des Servers als auch die des Einzelnen berücksichtigt. Wir stellen eine generische Bibliothek zur Verfügung, mit deren Hilfe PSA-Webanwendungen erstellt werden können. Darüber hinaus messen wir die Leistungsfähigkeit der Bibliothek und stellen Probleme in der Wirtschaft vor, die mit PSA und unserer Bibliothek gelöst werden können, und liefern ein Proof-of-Concept für zwei der Fälle. Wir untersuchen auch verschiedene Parameter dieser Anwendungen und geben Empfehlungen für die richtigen Einstellungen.

Schlagwörter: Informationssicherheit, Homomorphic Encryption, Differential Privacy, Private Selective Aggregation

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cryptography increasingly becomes the focus of attention in times where multinational organizations collect massive user data. Often, companies use collected datasets as input to machine learning algorithms, which generate new information based on the given data. The output of such algorithms turn out to be quite shocking sometimes, which was shown by a case where a company knew that a female customer was pregnant before her father did [48]. Technology progresses very fast, and the advancements are raising the question of adequate security among modern systems. Many solutions to common privacy problems are already present in the academic field. However, only a few of them are applied to the industry due to a lack of people identifying the issues and insufficient privacy awareness. We approach the problem by providing a new system for secure querying of data and dedicated tools for easy integration. In this chapter, we discuss the foundation of our work and outline the scope.

1.1 Private Data Queries

The idea of privately querying data is not new but was already studied quite some time ago. Initial work in the field [62, 38, 6] was leveraged later to introduce the notion of Private Information Retrieval (PIR) [27]. Retrieving data from a server poses a security risk to the client. A curious database manager could investigate which entries were requested by the client and infer what the client is after. PIR protocols are a way of privately retrieving information from one or many untrusted servers without leaking any information about what was retrieved. In this work, we mainly focus single-server variants [1, 18, 31, 50]. The concept of PIR is very similar to *oblivious transfer* [60] where the server remains oblivious about what and if anything was retrieved at all. Many advancements happened to oblivious transfer like 1-out-of-2 OT [36] and 1-out-of-n OT [15]. The latter can be seen as a stronger version of PIR since there is also the requirement that the client learns only the queried item and nothing else. PIR is tackling the privacy problem from the user's point of view by masking the query. Server privacy is neglected totally in the case of PIR since a curious client could query more items than required. In 1-out-of-n OT, the server's data privacy is protected to some degree due to the additional requirements. However, none of these techniques protect individual entries in a database. This can be accomplished by returning aggregated results. Unfortunately, retrieving an aggregated result from desired entries in a database requires special indexing, often

parametrized with sensible data. Therefore, the need for a system with additional privacy features arises.

1.2 Our Contribution

Our goal is to enhance privacy in systems that involve retrieving data from a server. We focus on systems that benefit from data generated by aggregation. We want to give rise to modern, secure applications by offering a protocol that can be incorporated in many ways. The protocol originates from previous work [16] where it serves as the core of a system that privately connects mobility to infectious diseases. We decided to name this protocol Private Selective Aggregation (PSA) and concluded that it suites well for more than this particular use-case because of its generic definition. We started to develop ideas for future scenarios and tools to make integrations easier. Currently, the work regarding PSA comprises the protocol definition itself, and an implementation of the said use-case in C++ [2]. All relevant information can be found in the original paper. We are using this work as a basis and extend it in many directions in this thesis. PSA enables private information requests towards aggregated results. Since aggregated data is returned, the privacy of individual database entries is protected, provided that all recommended security mechanisms are used. We initially give a rough overview of PSA and discuss the protocol in more detail in chapter 3.

The Target Domain. In recent years, data became one of the main drivers of the economy. According to a study [29] of the International Data Corporation (IDC)[1], we will have a total of 175 Zettabytes of data across the globe from which 49% will reside in the cloud by the year 2025. Additionally, as predicted by Gartner Inc.[2], annual revenue for Software as a Service (SaaS) platforms is increasing rapidly [39] as shown in Figure 1.1. Inherently, we would benefit further from cloud systems utilizing this massive amount of data. PSA can be used to implement a cloud data service, exposing aggregated information about a data pool. We define two parties in our protocol, which we refer to by *Client* and *Server*. The Server holds a data set in the form of a matrix in which the Client is interested. To protect privacy of individual entries, the Client cannot simply query them individually but is required to provide information about the indices in the form of an encrypted vector, such that the Server can multiply that vector with its matrix and return the encrypted result back to the Client. The multiplication inherently results in the aggregation of the data. The Server operates only on encrypted data and uses, therefore, a technique known as Homomorphic Encryption (HE). PSA is the perfect tool when two parties cannot share their data directly due to mistrust or prohibiting regulations like the European General Data Protection Regulation (GDPR)[3]. Nevertheless, it is possible to use PSA even if the two parties have no problems sharing

[1] https://www.idc.com/
[2] https://www.gartner.com/en
[3] https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679

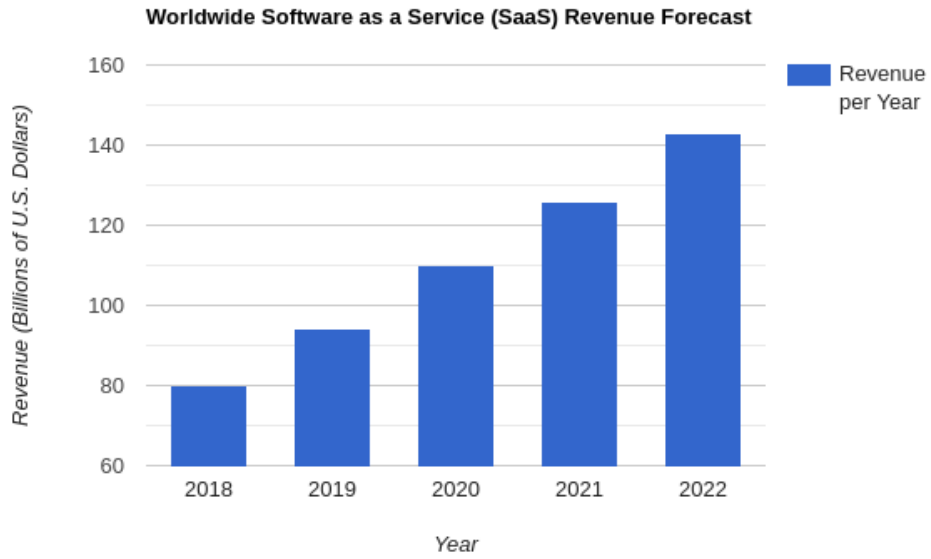


Figure 1.1: Worldwide Software as a Service (SaaS) revenue forecast as predicted by Gartner Inc. [39].

their data. However, the computational overhead will diminish the protocol's utility in terms of the cost-benefit ratio.

Providing the Right Tools. New technology can be leveraged more efficiently when the tools for straightforward integrations are available. Since PSA is a generic protocol that can be applied to a vast array of areas, we are very confident that a library for the easy integration of PSA into future applications would help the protocol to thrive. Therefore, we decided to provide such a library for the promising cloud environment. Companies and institutions that plan to implement a service to share information privately will benefit from the library considerably. In chapter 4 we talk in-depth about its internals, proper usage, and design decisions along the way. Additionally, we provide performance benchmarks and compare our implementation with one in a native language to give a feeling about trade-offs in the web.

Identifying Industry Problems. A new system gets overlooked easily when its intention is not initially clear, or the problem domain is defined sloppy. Therefore, in chapter 5, we identify various industry problems that can be solved using PSA. We describe the scenarios in detail and provide a PoC for two of the cases. The implementations are based on our library and show the protocol's viability in the web. Additionally, we study the privacy of the two implemented cases and analyze various parameter settings.

PSA is equipped with privacy mechanisms that require proper parametrization. Every system based on PSA handles different data in different situations. Therefore, a thorough analysis of the parameters is required. We study the proper setting of said parameters by defining measures to estimate the privacy level.

Chapter 2

Preliminaries

Our work builds on top of various research fields we want to discuss and outline in this section before diving deeper into them. Additionally, we are making use of mathematical expressions and notations during this work. To make this as straightforward as possible, we are trying to follow well-established conventions in this field.

2.0.1 Notation

We denote scalars as regular lower case letters, vectors as bold lower case letters and matrices as regular upper case letters. If we draw a vector $\boldsymbol{x}$ of dimension n randomly from the set of integers, we write $\boldsymbol{x} \xleftarrow{\$} \mathbb{Z}^n$ and if we assign the value $\boldsymbol{y}$ to this vector, we denote it as $\boldsymbol{x} \leftarrow \boldsymbol{y}$. Elements in the vector $\boldsymbol{x}$ are denoted as x_i, where i is in the range $1, ..., n$ in this particular case. The element-wise product of vector $\boldsymbol{x}$ and $\boldsymbol{y}$ is denoted as $\boldsymbol{x} \circ \boldsymbol{y}$ and is known as the *Hadamard product*. We write the inner product of $\boldsymbol{x}$ and $\boldsymbol{y}$ as $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ and denote the *euclidean distance* between them as $\|\boldsymbol{x} - \boldsymbol{y}\|$. For $m \in \mathbb{N}$ and $x \in \mathbb{Z}$, we write $\boldsymbol{x}^m$ to denote the vector of powers of x: $\boldsymbol{x}^m = (1, x^1, ..., x^{m-1})$. The ℓ_1 norm of $\boldsymbol{x}$ is denoted as $\|\boldsymbol{x}\|_1$. We denote $negl(\kappa)$ for a negligible function in the argument κ.

2.0.2 Homomorphic Encryption

Homomorphic Encryption is the idea of performing arbitrarily complex operations on a ciphertext. A ciphertext is an encrypted version of the raw data itself. The result will be encrypted and matches the results of operations to be performed on plain text. This idea was initially proposed by Rivest et al. [64]. However, the system was not capable of performing computations of arbitrary complexity, but was limited to a finite number of operations on the ciphertext. It was not sure if such a system even existed, until Gentry [41] came up with his revolutionary work, theoretically enabling an infinite amount of operations. This invention led to the field of Fully Homomorphic Encryption, and was followed by a series of improvements [12, 10, 14, 11, 53, 8, 42, 24, 13] by the community.

Definition. A Homomorphic Encryption scheme HE consists of the four algorithms $\mathsf{HE.KGen}$, $\mathsf{HE.Enc}$, $\mathsf{HE.Dec}$, and $\mathsf{HE.Eval}$. $\mathsf{HE.KGen}$ takes a security parameter κ as its input and outputs a key-pair $(\mathsf{sk}, \mathsf{pk})$, where sk is the secret key and pk the public key that defines the plaintext space $\mathcal{P}$ and the ciphertext space $\mathcal{C}$. $\mathsf{HE.Enc}$ takes as an input

the public key pk and a plaintext $m \in \mathcal{P}$ and outputs a ciphertext $c \in \mathcal{C}$. $\mathsf{HE.Dec}$ takes the secret key sk and a ciphertex $c \in \mathcal{C}$ and outputs m. The algorithm $\mathsf{HE.Eval}$ takes the public key pk, a circuit C from a permitted set of circuits C_{HE}, a tuple of ciphertexts $(c_1, c_2, ..., c_t)$ (one for every input bit of C), and outputs a new ciphertext c. Note that in some definitions of Homomorphic Encryption there is an evaluation key evk used as input to $\mathsf{HE.Eval}$. In this definition it is part of pk.

Security. A Homomorphic Encryption scheme HE is said to be *Indistinguishability Under Chosen-Plaintext Attack (IND-CPA)* secure [10] if for any polynomial-time adversary $\mathcal{A}$ it holds that:

$$|Pr[\mathcal{A}(\mathsf{pk}, \mathsf{HE.Enc}(\mathsf{pk}, 0)) = 1] - Pr[\mathcal{A}(\mathsf{pk}, \mathsf{HE.Enc}(\mathsf{pk}, 1)) = 1]| = negl(\kappa)$$

where $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{HE.KGen}(\kappa)$.
In other words, the scheme is said to be semantically secure if an adversary does not have a better chance than guessing with 50% probability wether a given ciphertext is the encryption of either one of two equally likely distinct messages. If that holds true, one can assume that no information about the content of a given ciphertext is learned by someone who has no access to the secret key.

Correctness. A Homomorphic Encryption scheme HE is correct [40] if for any t-input circuit C from C_{HE}, any key-pair $(\mathsf{sk}, \mathsf{pk})$ obtained from $\mathsf{HE.KGen}(\kappa)$, any t plaintext bits $(m_1, m_2, ..., m_t)$ and any ciphertexts $c = (c_1, c_2, ..., c_t)$ with c_i obtained from $\mathsf{HE.Enc}(\mathsf{pk}, m_i)$ it holds, that:

$$\mathsf{HE.Dec}(\mathsf{sk}, \mathsf{HE.Eval}(\mathsf{pk}, C, c)) = C(m_1, m_2, ..., m_t)$$

Example. A popular application for Homomorphic Encryption is a data owner who wants his data D to be processed by a cloud service provider. The data owner does not trust the provider, and so he generates a secret key sk and encrypts all of his data using the HE scheme HE. Next, he uploads the resulting ciphertext c to the provider's platform. The provider is aware of public key pk, and a circuit C used to evaluate c. Evaluation can be regarded as computing a function on the ciphertext c. This function is represented by the C. When receiving the c, the provider applies all the relevant computations and sends it back to the data owner, who will decrypt it eventually. The service provider did not learn anything about the data since he had no access to sk. More generally spoken, on the one hand, there is a client providing encrypted data c. On the other hand, there is a server evaluating c by only knowing pk and C and hands the result back to the client once he is finished. This process is depicted in Figure 2.1.

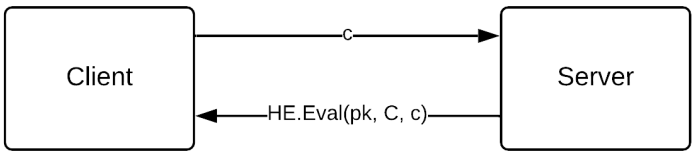


Figure 2.1: A high level view of Homomorphic Encryption.

2.0.3 Homomorphic Encryption Types

Homomorphic encryption schemes can vary quite a bit. However, they can be categorized into multiple types. To explain the different types, we first introduce the notion of a circuit.

2.0.3.1 Boolean Circuits

Expressing any Boolean circuit with only two types of gates, namely addition and multiplication, is not a trivial problem but was shown a long time ago. Every gate has two inputs. Moreover, such a circuit can express arbitrary functions. If a homomorphic algorithm can evaluate such a circuit, it can theoretically evaluate any function. Seeing functions as Boolean circuits makes it, therefore, easier to find cryptographic constructions. However, the downside is that functions have to be converted to Boolean circuits before such an algorithm can evaluate them. The function $f(x, y) = x + x \cdot y$ has two inputs x and y. To construct a boolean circuit given this function, one will need two gates - one for addition and one for multiplication. The circuit is depicted in Figure 2.2. For

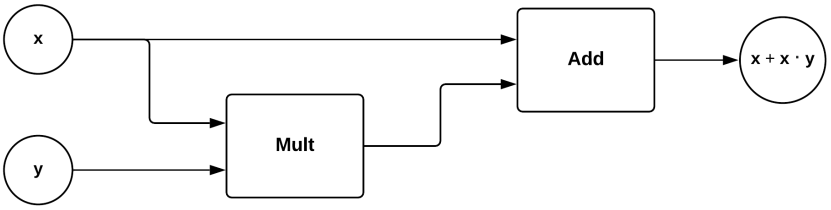


Figure 2.2: A simple Boolean circuit.

this thesis, there are two properties of interest, while there are many more in general. The first property is the *size*, which is simply the number of gates in the circuit. The second property is the *depth*, which is the maximal length of a path from an input to the

output gate. A Boolean circuit's depth can be described easier by visualizing the circuit as a directed, acyclic graph. That is, a graph where the vertices and edges are directed in a way such that following them will never form a closed loop. The circuit's depth is then the maximal length from an input gate to the output gate (the number of edges along the way). In this example, the size is 2 since there are two gates, and the depth is 2 since there are two gates from input y to output $x + x \cdot y$ along the way.

2.0.3.2 Types

There are three common types of Homomorphic Encryption schemes [69]:

- Partially Homomorphic Encryption (PHE)
- Somewhat Homomorphic Encryption (SWHE)
- Fully Homomorphic Encryption (FHE)

They are different in being homomorphic with respect to either the addition or the multiplication operation or both. Moreover, they differ in the computational power needed to run them.

Partially Homomorphic Encryption. Encryption schemes that are Partially Homomorphic can evaluate circuits only with respect to one operation, either addition or multiplication. Such schemes do not restrict the size or depth of the circuit to be evaluated. However, one should keep in mind that not every function can be transformed into a circuit if a fundamental operation is not available. Among the famous examples of Partially Homomorphic Encryption schemes are the Paillier [58] cryptosystem, which is additively homomorphic, ElGamal [35] and the textbook variant of RSA [63], which have a multiplicative homomorphic property. RSA encryption is defined as:

$$E(m) \coloneqq m^e \mod n$$

The homomorphic property is:

$$\begin{aligned} E(m_1) \cdot E(m_2) &\coloneqq m_1^e \cdot m_2^e \mod n \\ &= (m_1 \cdot m_2)^e \mod n \\ &= E(m_1 \cdot m_2) \end{aligned}$$

where E is the encryption function, $n \in \mathbb{N}$ the modulus, $m, m_1, m_2 \in [0, ..., n-1]$ the message, and $e \in \mathbb{N}$ the private key.

Somewhat Homomorphic Encryption. Schemes that are homomorphic with respect to addition and multiplication but are restricted by the depth of the circuit are called Somewhat Homomorphic Encryption schemes. The BGN cryptosystem, named after Boneh, Goh, and Nissim [7], was the first of its kind, allowing an arbitrary number of additions but only a single multiplication. In SWHE (and FHE) schemes, there is a so-called noise budget that must not run out before having all computations finished. The noise is a small error added into the ciphertext to guarantee the security of the scheme. The parameters determine the size of the budget and also the amount spent with each homomorphic operation. Once it runs out and exceeds a certain threshold, the ciphertext cannot be decrypted correctly anymore. In general, Somewhat Homomorphic Encryption schemes get noisier with every operation on the ciphertext, and multiplications are more expensive than additions. Gentry's original idea is used frequently [30, 67] to turn Somewhat Homomorphic Encryption schemes into Fully Homomorphic Encryption. The original work of Gentry was based on ideal lattices and covered three parts: as a first step he constructs a SWHE scheme, then simplifies its decryption function (called squashing), and then he homomorphically evaluates the decryption function to lower the noise budget (called bootstrapping).

Fully Homomorphic Encryption. Fully Homomorphic Encryption (FHE) schemes can homomorphically evaluate both addition and multiplication gates and are not restricted in either size or depth. In reality, however, they are usually extremely inefficient, and any application should be thoroughly evaluated before usage of such a scheme. Nevertheless, FHE schemes get more and more efficient, and thanks to Gentry, we know how to transform a SWHE scheme that can evaluate its own encryption circuit to a FHE scheme by a process called bootstrapping. However, bootstrapping comes with a high computational cost, making Fully Homomorphic Encryption obsolete for some applications. Depending on the application, it is wiser to choose a SWHE scheme and set the parameters accordingly. The most important FHE scheme for this work is the Brakerski/Fan-Vercauteren (BFV) [12] scheme. The two operations in the scheme are addition and multiplication. Both operations consume the noise budget at a rate also determined by the parameters of the scheme. The addition is nearly free to use, while multiplications are very expensive. The scheme's security is based on the Ring-Learning-with-Errors [54] hardness assumption, which is the ring variant of Learning-with-Errors (LWE) [61].

2.0.4 Differential Privacy

The goal of Differential Privacy [33] is to make the overall output of an algorithm operating on a (statistical) database as independent as possible from a single entry of that database. That is, it protects individual privacy from being exposed to some external entity receiving the output of the algorithm and prevents the (additional) harm individuals would take due to the information generated by the difference between being in the dataset and not being in the dataset. A randomized algorithm $\mathcal{A}$ is ϵ-differential

private if for all adjacent datasets D_1 and D_2, and all $\mathcal{S} \in \text{range}(\mathcal{A})$ it holds that:

$$Pr[\mathcal{A}(D_1) \in S] \leq e^{\epsilon} \cdot Pr[\mathcal{A}(D_2) \in \mathcal{S}] \tag{2.1}$$

for $\epsilon > 0$. Intuitively, it means that an algorithm is differentially private if one cannot tell if any individual's data was included in the original dataset by looking at the output. Differential Privacy identifies the relationship between the change of a single data entry and the output of a query. The maximum magnitude of change in the output arising when altering a single data entry that is evaluated over all possible combinations of entries is called the *sensitivity* Δq of the system. It is defined as:

$$\Delta q = \max_{D_1, D_2} \|q(D_1) - q(D_2)\|_1$$

where $\mathcal{D}$ is a collection of adjacent datasets, $D_1, D_2 \in \mathcal{D}$, and $q : \mathcal{D} \mapsto \mathbb{R}$. The maximum is taken over all pairs of D_1 and D_2. The sensitivity determines how much uncertainty must be introduced in the final result to hide the participation of a single entry and can be exactly computed for every application. Additionally, there is the privacy parameter ϵ. A popular method to meet the criteria described in equation 2.1 is to add noise drawn from a zero-centered Laplace distribution with scale $b \in \mathbb{R}$ and the probability density function:

$$Lap(x|b) = \frac{1}{2b} e^{-\frac{|x|}{b}}, \qquad \text{with } b = \frac{\Delta q}{\epsilon}$$

The variance σ^2 of the distribution is $2b^2$ [34]. Therefore, choosing higher values for the privacy parameter ϵ will result in less variance and, thus, in noisy outputs closer to the true value. This will inevitably lead to more utility and lower privacy levels. Special care has to be taken in applications using real-valued numbers since floating-point implementations can vary and destroy this mechanism due to rounding. In integer cases, this function's result should just be rounded to the nearest integer or rounded up if the value is strictly in between.

Choosing the right epsilon. It turns out to be quite challenging to choose an appropriate privacy parameter ϵ. Although it is the privacy parameter, it does not directly adhere to some privacy standards and is not an absolute measure but rather a relative measure. A lower value of ϵ implies more noise and, therefore, less chance for an adversary to identify a single entry. Higher values mean less noise and more chances for an adversary. It is hard to quantify the privacy of a system and individual data entries in a database since this highly depends on the application and the data itself. If ϵ is set too high, privacy might suffer. If it is set too low, the utility of the application decreases. Nevertheless, choosing the best ϵ is the central part of an application based on Differential Privacy. Unfortunately, there is no default setting for ϵ, which is clearly shown by the literature [56, 22, 52, 5, 55, 49] where ϵ is chosen within the range of 0.01 to 7, oftentimes without proper justification. Therefore, we will try to figure out the best ϵ by experimenting within that range in our implementations.

Chapter 3

Private Selective Aggregation

After explaining the fundamental concepts, we now discuss Private Selective Aggregation (PSA) in more detail. PSA is a two-party protocol that allows privacy-preserving querying of a database if all privacy recommendations are followed. The querying party does not learn individual items in the database since an aggregated result is returned [16].

3.1 Introduction

In PSA, the querying party called the *Client* can privately query the database of the other party called the *Server* to obtain an aggregated result. The database resides in the form of a matrix. The Client does not learn individual data entries in the Server's matrix if privacy parameters are set accordingly. The query is encrypted in the beginning, and even computations on it happen in the encrypted domain. A simplified depiction is given in Figure 3.1. In straightforward terms, PSA is a multiplication of a query vector and a data matrix in the encrypted domain. The vector serves as a list of indices to the corresponding matrix rows. These indices are pointing to the data that is aggregated into a final value and collected in an output array. In a real-world scenario, the Client could be a small business or institution in need of specific information only to be found in a cloud provider's database who is the Server in this scenario. The protocol is of most use if the Client's query and the Server's matrix must be kept private. One might see that the Client can only conduct a reasonable query if the data in the Client's vector somehow correlates to the data stored in the Server's matrix. The order of their entries has to match so that every entry correlates to an entry in the opposite party's data structure. The Client and Server have to place the data in the same sequence. However, sometimes the Server holds more data in his matrix than the Client in his vector. To solve this problem, the Client and the Server have to calculate their Private Set Intersection (PSI) before running the PSA protocol. There are many efficient PSI solutions developed already [23, 59, 51]. PSI is a protocol with two parties, each holding a set, where the goal is to calculate the intersection of the sets without revealing anything else than the intersection itself. The individual sets are kept private and are, therefore, not exposed to the other party. PSI is not in this thesis's scope but should be considered when using PSA. Since the PSA core protocol suffers from a few privacy issues, it can be extended with privacy-enhancing tweaks like masking and Differential Privacy [33] that eliminate these issues. The whole Chapter 3 follows the work in the original paper [16]. We will

discuss the internals of PSA and the additional privacy mechanisms in the following sections.

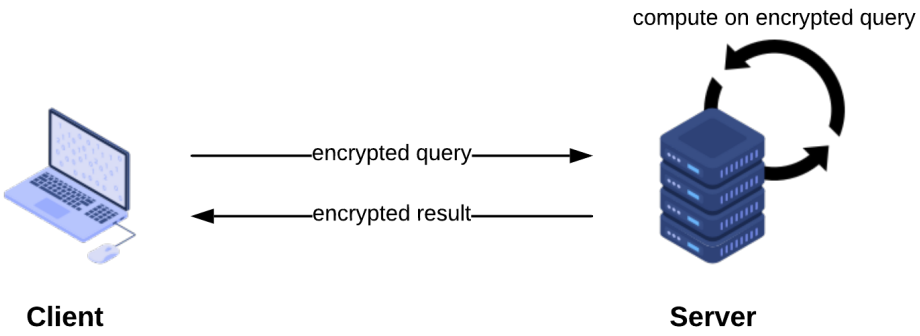


Figure 3.1: A high level depiction of Private Selective Aggregation. The Client can privately query a database for an aggregated result. Neither the query nor individual entries in the Server's data matrix are learned by the Client.

3.1.1 Protocol

In the protocol and during this writing we are using a Homomorphic Encryption scheme $\mathsf{HE} = (\mathsf{HE.KGen}, \mathsf{HE.Enc}, \mathsf{HE.Dec}, \mathsf{HE.Eval})$. In the beginning, the Client generates a secret key sk and an evaluation key evk by running $\mathsf{HE.KGen}$. The evaluation key is needed to perform operations on encrypted data. Thus, the server also needs evk, but transmitting it is not part of the protocol and remains an implementation detail. The query of length N is encoded into a vector $\boldsymbol{x} \in \mathbb{Z}^N$ and encrypted to obtain the ciphertext $\boldsymbol{c} \leftarrow \mathsf{HE.Enc}_{\mathsf{sk}}(\boldsymbol{x})$. This ciphertext is sent to the server who holds a data matrix $Z \in \mathbb{Z}^{N \times k}$. The server in turn performs the *compute* step. That is, multiplying the matrix Z with the encrypted input vector $\boldsymbol{c}$. For computation in the encrypted domain, the evaluation algorithm $\mathsf{HE.Eval}$ is required. It uses the evaluation key evk and outputs $\boldsymbol{h}^* \leftarrow \mathsf{HE.Eval}_{\mathsf{evk}}\left(\boldsymbol{c}^T \cdot Z\right)$. After the computation, various privacy preserving techniques are applied to $\boldsymbol{h}^*$. These techniques will be discussed in the next section. $\boldsymbol{h}^*$ is sent back to the client who uses sk for decryption and obtains the result $\boldsymbol{h} \leftarrow \mathsf{HE.Dec}_{\mathsf{sk}}(\boldsymbol{h}^*)$.

3.1.2 Privacy

As already mentioned, the goal of PSA is to provide the Client with aggregated data obtained from the Server's data pool without leaking any information other than that. The input vector provided by the Client is not exposed to the Server in plain form. The

Server does not learn any information about the vector, and the Client learns nothing about the individual entries in the Server's matrix since the values are aggregated in the result vector. Nevertheless, when using the protocol in the wrong way, information leaks can occur. Assume an input vector $\boldsymbol{x} \in \mathbb{Z}^n$ with $n-1$ 0s and a single 1:

$$\boldsymbol{x} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{3.1}$$

Such a vector is highly possible. If the input vector holds only zeros except for a single one, then individual privacy is in danger since this single one from the input vector will lead to the aggregation of a single matrix entry in the result and is, therefore, visible to the Client. Even if the Client does provide a multiple non-zero entries vector, the information could still be leaked if the entries are, for example, tied to different contexts and would allow an attacker to interpret the result in a way such that specific information is leaked for every context individually. The Corona Heatmap from section 5.2.2 serves as an example. The final output is a heatmap showing the geographical footprint of a set of infected people. If the input vector is chosen so that there is a single 1 in the vector for each geographical region, respectively, the footprint of each region's person will be visible in the output. To circumvent that, there are a few mitigation techniques a PSA implementation should support.

3.1.2.1 Masking

There are a few issues in the plain PSA protocol presented. First, in some cases, a non-binary query vector could render the resulting data invalid since this leads to duplicate values in the aggregation and, therefore, to a distorted result. Second, a Client can target single entries in the Server's matrix by using a vector like the one in equation 3.1. As an example, we again point to the Corona Heatmap from section 5.2.2 where a non-binary query vector would lead to a heatmap where some of the infected individuals contribute more than others, although there is no definition of a biased output. A vector as the one described above would allow querying single individuals. To prevent said issues, the idea is to optionally add masks to the result vector before sending it back to the client. The masks are zero-valued if the Client follows the rules agreed on. If the Client is dishonest, the result vector is randomized and of no use anymore. The simplified concept is presented in Figure 3.2. All masking happens on the server-side after the main computation has finished. The masks are added to the result vector and are not obligatory but remain an option, although if no explicit statement is declared, we recommend using them by default.

Non-Binary Masking. A non-binary query vector applies weights to specific entries in the Server's matrix and has the potential to enhance the utility of the protocol. However, in some cases, such a query can distort the result in a way that would lead to

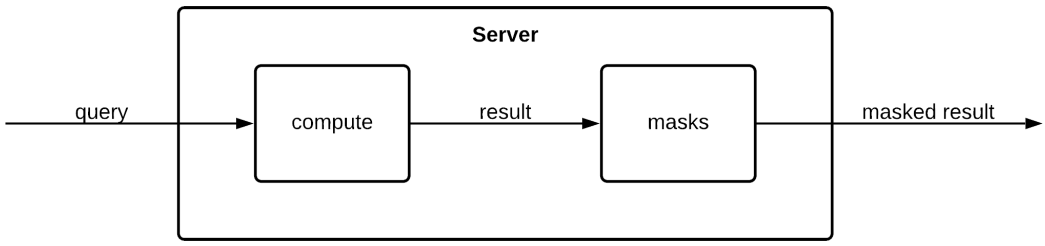


Figure 3.2: High-level depiction of the masking process on the server side.

a sabotaged or simply wrong output regarding the specific application. To circumvent that, we use a mask $\mu_{\texttt{bin}}$. The mask asserts that only zeros and ones can be used in the query vector. If the Client is dishonest and uses non-binary numbers, the resulting vector will be randomized, and thus, the output rendered useless. To see if the query vector $\boldsymbol{x}$ is indeed binary, the Server cannot simply check it since only the encrypted version $\boldsymbol{c}$ is transmitted and cannot trivially be inspected. However, by using a trick very close to the one used in the non-interactive zero-knowledge proof protocol Bulletproofs [17], the vector can be checked. Since Homomorphic Encryption is used in this protocol, the Server can compute in the encrypted domain. Even though $\boldsymbol{c}$ is a ciphertext, HE enables us to do regular computations with it. The idea is to calculate $\boldsymbol{d} = \boldsymbol{c} - \mathbf{1}$ and then the inner product $\langle \boldsymbol{c}, \boldsymbol{d} \rangle$, since this will result in a value of zero if $\boldsymbol{c}$ is indeed binary. If it is binary, then $\boldsymbol{d}$ will hold a 0 at every index where $\boldsymbol{c}$ is 1 and $\boldsymbol{d}$ will hold a (-1) at every index where $\boldsymbol{c}$ is 0. Therefore, computing the inner product will result in a scalar with the value 0. Since the Client still has some chance to cheat, by constructing $\boldsymbol{c}$ in a way such that some entries cancel each other out, a random integer y is introduced and multiplied (Hadamard product) with $\boldsymbol{d}$ before calculating the inner product. The security is further increased by independently checking for a binary $\boldsymbol{c}$ twice. The resulting binary mask will then be:

$$\mu_{\texttt{bin}} = \langle \boldsymbol{c}, (\boldsymbol{d} \circ \boldsymbol{y_1}^N) \rangle \cdot r_1 + \langle \boldsymbol{c}, (\boldsymbol{d} \circ \boldsymbol{y_2}^N) \rangle \cdot r_2$$

where $r_1, r_2 \xleftarrow{\$} \mathbb{Z}_t \setminus \{0\}$ are two random integers. The Server can apply this masking without and further interaction with the user. Fortunately, this means that the Server can check for a non-binary vector without permitting the Client to influence this decision, let alone notice it.

Small or Wrong Hamming Weight Masking. If the Client sends a vector filled with only 0s except for a single 1, the privacy of the Server's data is not guaranteed. Such a vector has a *Hamming Weight* of 1. The Hamming Weight (HW) of a vector is defined as the sum of all elements in that vector. Such a vector would inherently lead to the leakage of individual entries in the Server's matrix. We prevent this by using

another mask $\mu_{\texttt{HW}}$. For this purpose, we modify the protocol and introduce more Client interaction. The Client calculates the Hamming weight w of the query vector and sends it to the Server together with the encrypted version of the vector $\boldsymbol{c}$. To check if the Client was honest and did not send a wrong w, the Server needs to compute it too, even though the query vector transmitted is encrypted. The HW can be calculated in the encrypted domain by the inner product of the input vector $\boldsymbol{c}$ and a vector $\mathbf{1}^N$ holding only 1s. Next, the Hamming Weight w, provided by the Client, is subtracted from this value to obtain the mask:

$$\mu_{\texttt{HW}} = \langle \boldsymbol{c}, \mathbf{1}^N \rangle - w \,.$$

If the Client is honest, $\mu_{\texttt{HW}}$ will result in 0. In the dishonest case, it will be a non-zero value. It will later be multiplied by some randomness, unknown to the Client and added to the Server's final result. Thus, the final result will be rendered useless if the Client was dishonest about w.

Final Mask Computation. After the masks are calculated individually, they have to be combined in the last step, and some randomness must be added for further security improvement. The two masks are added together to obtain the final mask:

$$\boldsymbol{\mu} = (\mu_{\texttt{bin}} + \mu_{\texttt{HW}}) \cdot \boldsymbol{r}$$

where $\boldsymbol{r} \xleftarrow{\$} (\mathbb{Z}_t \setminus \{0\})^k$ is a random vector. This vector assures that a different random value is added to every element in the final mask to prevent the Client from learning anything about the mask if any values of the final result are known in advance.

3.1.2.2 Differential Privacy

Even if all communication is encrypted, the final result aggregated, and masks applied in the end, there is still a security issue, e.g., in the context of mobility data when looking at distinct geographical locations [70]. Again, in the Corona Heatmap from section 5.2.2, the Client can form a query to target individuals of distinct geographical locations and gain knowledge about these individuals. Therefore, it makes sense to introduce Differential Privacy to PSA. It targets the unlinking of single entries in the Server's matrix from the output of the protocol by introducing noise accordingly. The details about Differential Privacy are discussed in section 3.1.2.2.

Combining Masks and Differential Privacy. As a general rule, we recommend using masking as well as Differential Privacy for PSA applications. However, in a few scenarios, it might result in a too tight restriction. A user of Private Selective Aggregation must carefully consider the circumstances and evaluate which privacy mechanisms fit the application. We want to stress that all mechanisms are optional, and every combination between them is possible. A depiction of the complete control flow related to the privacy opt-ins is given in Figure 3.3. The query vector reaches the Server, and the *compute* step is executed immediately. After retrieving the result, it can then be masked by either the

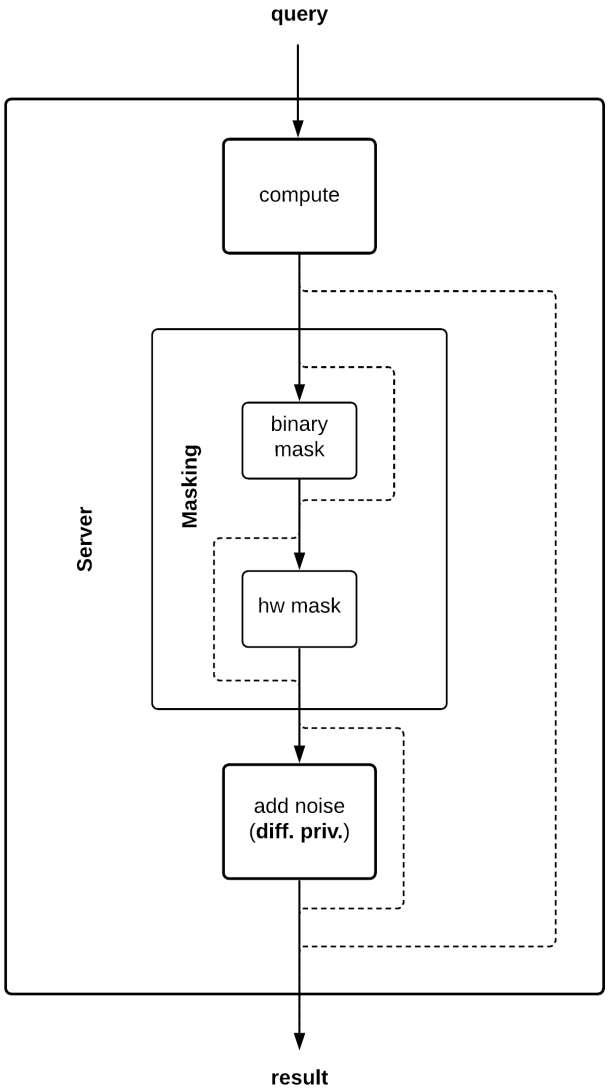


Figure 3.3: High-level depiction of the optional privacy extensions in Private Selective Aggregation. A dashed line indicates an optional path.

binary or the hamming weight mask, or both. Further, differential privacy can be applied right after *compute* or directly after the masking. It is possible to use neither masking nor Differential Privacy in rare cases where the plain PSA protocol already guarantees full privacy, e.g., individual privacy is already implemented at the data-entry level.

3.1.2.3 Security Analysis

So far, we described the PSA protocol and the mechanisms required to make it secure. On top of that, we can now define what we mean by *security* and justify our claims. First, we describe the terminology used in the following:

Semi-honest/malicious adversaries: We study the security of the protocol by using two types of attackers. First, we have *semi-honest* adversaries who will follow the protocol but may inspect any transcripts of it after execution, with the goal to learn anything they should not. Second, there are *malicious* adversaries who might deviate from the protocol specification arbitrarily and change inputs and outputs in their aim to cheat.

Real-ideal paradigm: The real-ideal paradigm [43] is a security definition for multi-party protocols. It compares the real protocol execution to an "idealized" version of the protocol where all computations happen honestly and channels are perfectly secure. The standard definition of the idealized version today [19] defines no communication among the parties directly, but the presence of an incorruptible third party acting as a mediator. All of the other parties' inputs are sent to this third party, which in return handles the computation of functions locally and transmits possible outputs to the corresponding party. A real-world protocol is said to be secure if any attack carried out on the real protocol could also be carried out in the idealized one. This type of security definition is also called *simulation-based* since a real-world protocol execution is compared to a simulated version.

One-sided simulatability: First considered in the context of Oblivious Transfer[57] and later formalized [47], one-sided simulatability is a relaxation of the simulation-based definition. It defines full simulation for one party and privacy via computational indistinguishability (see paragraph 2.0.2) for the other party. Privacy can be proven by showing that the corrupted party cannot distinguish between inputs.

PSA achieves simulation-based security against semi-honest adversaries while preserving the privacy of the Client's vector against a malicious Server. This is proven by defining the ideal execution of the protocol and checking it against a real execution.

Security Against Semi-Honest Adversaries. The proof works by showing that a real-world execution with a semi-honest adversary is indistinguishable from the idealized execution run by a simulator. This can be achieved by defining a system that breaks the IND-CPA security (see paragraph 2.0.2) of HE. That is, we reduce the security of PSA in the semi-honest adversary setting to the security of IND-CPA. The full proof can be found in the original paper [16] in Appendix B.

Security Against Malicious Adversaries. It is not trivial to achieve simulation-based security against a malicious adversary since we cannot control what is sent by either party. Therefore, we focus on the privacy of the Client's vector $\boldsymbol{x}$ by proving its

privacy in a malicious Server setting. Since the vector is homomorphically encrypted before transmission, the Server needs to break HE's semantic security. Therefore, we again reduce the security to IND-CPA security of HE. That is, PSA achieves privacy of $\boldsymbol{x}$ against a malicious Server. The full proof can also be found in the original paper.

Chapter 4

Bringing PSA to the Web

Systems running PSA rely on Homomorphic Encryption. Therefore, they require an underlying implementation to power the HE part. There is already a considerable amount of libraries available of which a large proportion is implemented in C++ or other languages restricted to execution in native environments. Popular libraries are e.g. HElib[1], SEAL[2], and TFHE[3]. A current implementation of an PSA application, called the *Corona Heatmap* [4] is based on SEAL. There is also a library called node-seal[5] written in Javascript, utilizing the WebAssembly[6] API. By using this library, we can bring PSA to the web, supporting completely new kinds of applications. The user does not have to carry out an installation process but instead can simply open the browser, head over to a URL, upload data, and execute the protocol with no prior setup. The enhancement in usability eases the adoption of the technology. In this chapter, we discuss the frameworks and libraries used in PSA, its general performance, caveats, its proper usage and where to get it.

4.1 Technology in PSA Lib

To bring PSA to the web, we utilize a variety of different libraries and frameworks. The most important one is node-seal which builds on top of SEAL by Microsoft and offers support for Homomorphic Encryption in javascript. Additionally, we make heavy use of Node.js, on which the library depends. Nevertheless, the PSA library can run in a browser through a Node.js powered frontend library like, e.g., React or Angular.

4.1.1 Node.js

Node.js[7] is an open-source Javascript runtime that enables Javascript applications outside a browser. It uses a non-blocking, event-driven I/O model to be lightweight. Originally, Javascript was developed for the web to manipulate the internal representation of HTML

[1] https://github.com/homenc/HElib
[2] https://github.com/microsoft/SEAL
[3] https://tfhe.github.io/tfhe/
[4] https://github.com/IAIK/CoronaHeatMap
[5] https://github.com/morfix-io/node-seal
[6] https://github.com/IAIK/CoronaHeatMap
[7] https://nodejs.org/en/

documents in the browser, which enables interactive web pages and dynamic content rendering. In the area of web development, JavaScript gave rise to full-stack web developers working on both the client and the server-side of an application [66]. The community is growing fast which leads to a continuous increase in the number of available packages for Node. Popular frontend libraries like React or Angular are using Node for, e.g., their building and bundling process or to spin up a local web server for easier testing.

4.1.2 SEAL

SEAL is one of the most popular open-source Homomorphic Encryption libraries. It was developed at Microsoft and has been adopted by both academic [16, 32] and industry professionals[8][9]. The goal is to ease the complexity of Homomorphic Encryption and make it accessible to people unaware of the cryptographic internals. It is written in C++ with no external dependencies and continuously updated by Microsoft. Since Homomorphic Encryption is a non-trivial topic, it includes many easy examples with extensive commentary to better understand the library.

4.1.3 Node-SEAL

Node-seal[10] is an open-source homomorphic encryption library in JavaScript which builds on top of Node.js and SEAL by utilizing Web Assembly. It brings homomorphic encryption to any JavaScript environment, including browsers. The name *node-seal* is misleading to some degree since it suggests that only Node.js environments can run the library. Node-SEAL is very similar to SEAL and requires no additional dependencies. The usage feels very close to SEAL, since it directly calls the C++ functions via the WebAssembly API. Web Assembly is a portable low-level byte code that was developed for the web and is cross-browser compatible. WebAssembly has a low-level syntax that can be automatically compiled from C++, C, or any other supported language. The resulting file has the ending *.wasm*, which stands for *WebAssembly*. Optionally, one could write web assembly manually in a *.wat* file, which is the *WebAssembly Text* format. The syntax is very similar to regular assembly language. However, it should be viewed more like the smallest set of commonalities between all common assembler languages. Compiling a supported language can be done with Emscripten[11], a compiler toolchain built on top of LLVM and focused on the web platforms. The resulting WebAssembly code can be much faster than conventional JavaScript code since tasks like memory management and object references are performed manually and under the developer's control. The byte code can then be loaded with JS by so-called *glue-code*, very much like a module and executed within a browser. Figure 4.1 depicts the process of transforming native code to a module that a browser can run.

[8] https://www.intel.com/content/www/us/en/artificial-intelligence/posts/he-transformer-for-ngraph-enabling-deep-learning-on-encrypted-data.html
[9] https://techmonitor.ai/techonology/cloud/homomorphic-encryption-microsoft
[10] https://github.com/morfix-io/node-seal
[11] https://emscripten.org/

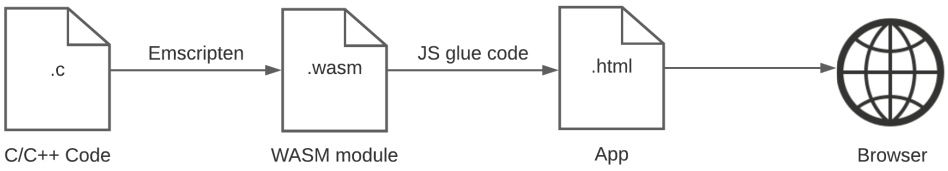


Figure 4.1: The web assembly pipeline: Starting with regular C/C++ code through to its execution in a browser.

4.2 PSA Library

To build arbitrary web applications relying on PSA, we released a publicly accessible library in JavaScript[12]. The library is standalone and does not need any dependencies other than node-seal. We focused on building it in a way that makes it easy to adopt and provide a variety of adjustment options to the user. The PSA library can be finetuned according to the particular application. The parameters of the library are passed down the hierarchy to the underlying SEAL instance. Users can choose the plaintext modulus, the polynomial degrees, the security level, the compression method used by SEAL internally, masking, and differential privacy. Choosing suitable parameters heavily depends on the underlying application and should include trial and error to some degree.

4.2.1 Performance

Since server and browser environments run JavaScript, it is the optimal language to make it widely accessible. The PSA library can be used in everyday applications without the overhead of installing software on the local machine. Even in server environments it is easy to set up the toolchain using the Node Package Manager (NPM). For browsers it is only required to visit a particular site referenced by its URL. However, since JavaScript code is not native and has to be interpreted or JIT-compiled first, an application could be slower than a similar one written in C or C++ by multiple factors. To clarify how this implementation performs and compares to an implementation in a more low-level language, we ran a variety of benchmarks on a cluster node with 20 logical cores, an Intel Xeon E5-2660 v3 @ 2.60GHz CPU and 32GB RAM. Unfortunately, the implementation cannot fully utilize the multi-core environment since JavaScript follows a single-threaded design. We could spawn so-called *Worker threads* which are very similar to regular threads. However, spawning threads in JavaScript breaks its fundamental single-threaded design, and the computational overhead for doing so is not worth it. In this section, we provide detailed information about the PSA library's performance and compare it to an implementation in C++.

[12]https://www.npmjs.com/package/psa-lib

4.2.1.1 Preliminaries for Reading the Results

In order to gain meaningful test results, we need to measure the right aspects of the library. Using a HE scheme in practice has some implications which we want to discuss before considering the test results. We use BFV for homomorphic operations in our implementation since it is the most appropriate scheme for PSA offered by node-seal. In BFV, certain parameters must be chosen before doing any computations. We will discuss the parameters in further detail in section 4.2.4.2. For now, the most important parameters are the plaintext modulus p, the ciphertext modulus q, the security level κ, and the degree of the polynomial modulus n that determines the so-called *slot size*. If a user of SEAL wants to use an array of length N for computation, the user must first encode and then encrypt it. N is required to be smaller or at most equal to n. The array will be accessible in the encrypted domain in the form of a matrix with dimensions $2 \times \frac{n}{2}$. Various operations like addition, multiplication, and rotation can be applied to it. Internally, the library is using the diagonal method [44] with the baby-step giant-step technique [45, 46] applied to it for multiplying the input vector $\boldsymbol{c} \in \mathbb{Z}^N$ with the matrix $Z \in \mathbb{Z}^{N \times k}$. This techique requires rotations of $\boldsymbol{c}$. Since $\boldsymbol{c}$ is split into two rows in the encrypted domain and we can only perform the rotation for each row separately, we cannot simply do a multiplication between $\boldsymbol{c}$ and Z directly, but have to do the multiplication for each row in $\boldsymbol{c}$ separately. As a consequence, we have to split Z into submatrices. The total number of vector-matrix multiplications is given by $\#\texttt{MatMul} = vecs_n \cdot mats_n$, where $vecs_n = \lceil \frac{N}{n} \rceil$ and $mats_n = \lceil \frac{2k}{n} \rceil$, and $vecs_n$ and $mats_n$ are rounded up to the next multiple of n if they are not already a multiple. We will occasionally use $\#\texttt{MatMul}$ in relation to execution time when measuring the library's performance.

4.2.1.2 Runtime

The PSA library's runtime is influenced the most by the internal encrypted multiplication between the input vector and the matrix. Although many more operations are executed next to the multiplication, it is sufficient to measure only this particular part to gain accurate insights into the overall library's performance. All other tasks are neglectible in terms of runtime. Table 4.1 lists different performance test runs for various parameter sets and indicates the number of needed matrix multiplications and the consumed time. One can observe a considerable performance loss when using larger parameter sets, i.e., increasing the reduction polynomial n. This helps to allow more computations until the noise budget is finally consumed but comes with the drawback of high computational cost and larger ciphertexts. Regarding memory consumption, one needs to be cautious since the matrix provided as an input to the library can already exceed the limits of the machine running the library. Consider Nr. 12 in Table 4.1, where the matrix is of dimensions $N = 163840$ and $k = 40960$. The size of this matrix in memory, provided that the JavaScript's Number type has 8 bytes, is 50 GB. The processing of this amount of data was only possible by generating the matrix values separately during the computation. For future work based on the PSA library, we recommend to proceed in the same way. The library itself does not use unexpected amounts of memory for its internal datatypes.

Nr.	BFV $\log_2(p)$	BFV $\log_2(q)$	BFV n	BFV κ	Matrix N	Matrix k	#MatMul total	Runtime min
1	33	218	8192	128	8192	4096	1	1.61
2	33	218	8192	128	40960	4096	5	8.18
3	33	218	8192	128	81920	4096	10	17.10
4	33	218	8192	128	163840	4096	20	32.45
5	33	218	8192	128	40960	20480	25	40.40
6	33	218	8192	128	98304	32768	96	154.69
7	42	438	16384	128	16384	8192	1	10.48
8	42	438	16384	128	81920	8192	5	53.27
9	42	438	16384	128	163840	8192	10	109.67
10	42	438	16384	128	327680	8192	20	217.65
11	42	438	16384	128	81920	40960	25	269.77
12	42	438	16384	128	163840	40960	50	542.44

Table 4.1: Runtime in minutes for the encyrypted matrix multiplication for different parameters.

However, SEAL is using a particular compression method named *ZSTD* [65]. The implementation of the compression had a bug[13] which was present until version 3.6.1. Due to this bug, memory consumption was very high in node-seal and, therefore, in the library. Earlier versions of the PSA library use the buggy version of SEAL. This was fixed in the latest release where memory consumption is now stable. However, users of the library should ensure to always use the latest version.

4.2.1.3 Linear Dependency

The runtime of the protocol depends on the size of the matrix. It does not matter which of the two dimensions is increased. The runtime will eventually increase roughly by the same amount for both dimensions. Figure 4.2 shows the linear increase of the runtime depending on the number of matrix multiplications. This allows a prediction of the protocol's runtime for a particular set of dimensions N and k.

4.2.1.4 Performance Compared to C++ Implementation

Comparing the JavaScript version to the one in C++, we conclude that JavaScript is slower by a factor of approximately six. Fortunately, it is consistently slower by this factor, which makes it possible to predict the runtime of applications with parameters that have not been tested before. This result also indicates that C++ is suited better for very large applications and JavaScript for smaller ones where the time difference is negligible. The difference in runtime becomes more significant as the application's

[13] https://github.com/microsoft/SEAL/issues/248

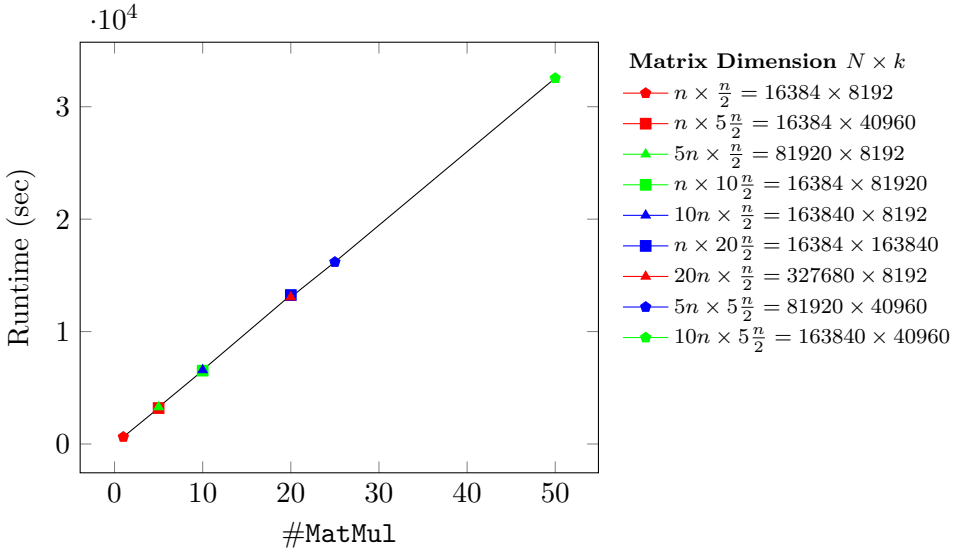


Figure 4.2: Linear dependency of the runtime of the matrix multiplication and the number of `MatMul` evaluations. BFV parameters used: $\log_2(p) = 42$, $\log_2(q) = 438$, $n = 16384$, $\kappa = 128$.

complexity increases. An application running six seconds rather than one second may seem tolerable. However, the impact becomes more clear when running an application for six hours instead of one hour. In the following, we present Table 4.2 and Figure 4.3, showing directly how the JavaScript implementation compares to the C++ one.

4.2.2 Tools Used

We were trying to keep the library's dependencies as small as possible to minimize the bundle size. We accomplished using node-seal as the only dependency since the library would not work without it. Besides that, we used code prettifier, testing, and debugging tools. However, they were exclusively used during development and had no impact on the execution of the library. The library was developed using Node.js version 14.15.1. However, users are free to choose an older version from Node 10 and above.

4.2.3 Compatibility

Developing JavaScript libraries that are meant to work in browsers as well as server environments is a bit tricky since those two environments differ a lot in their structure and context. Even the JavaScript engines used to execute code are not the same in some cases. Every browser has its own implementation. For JavaScript on server environments, Node.js serves as a runtime. However, Node's development was not always in sync with the advances of the browser-operated JavaScript implementations. When it comes to the concept of modules, the two environments differ quite a bit. For a very long

Nr.	Matrix N	Matrix k	JS min	C++ min
1	16384	8192	10.48	1.70
2	16384	40960	53.27	8.43
3	81920	8192	54.68	8.45
4	16384	81920	108.75	16.88
5	163840	8192	109.67	16.88
6	16384	163840	220.79	33.75
7	327680	8192	217.65	33.76
8	81920	40960	269.77	42.20
9	163840	40960	542.44	84.35

Table 4.2: Runtime in minutes for both the JavaScript and C++ implementation for the parameter set: $\log_2(p) = 42$, $\log_2(q) = 438$, $n = 16384$, $\kappa = 128$.

time, JavaScript code had to be written in a single file since there was no option to include other JavaScript files in an existing file. The arrival of a combination of modules and a defining standard was a long time coming which led to a variety of different implementations for every environment. It differed not only in the syntax but also in the way the modules were loaded. In browser implementations, modules were loaded asynchronously, while in server environments, they were loaded synchronously. Browsers, in general, used RequireJS[14] based on the AMD (Asynchronous Module Definition) spec to import JavaScript modules, while Node.js used the CommonJS[15] module spec. The two syntaxes are not compatible with each other. Nevertheless, it is possible to support both environments. We wrote our files in CommonJS Syntax and used Babel[16] to transpile them to RequireJS syntax. In the bundle, we provided both files so that a specific environment can use the corresponding file. We can support even older environments with this solution since newer ones implement the ES6 Modules standard.

4.2.4 Usage

The library is publicly available and accessible by the Node Package Manager[17]. It can be used in any node-powered environment that supports package management by any common managers like NPM or yarn. This includes React, Angular, Express.js applications, and many more. The PSA library is installed by running `npm i psa-lib` from the command line so that the bundle gets registered in the according package.json file. The interface of the library is straightforward and has a high usability. The source code, together with an API documentation, is available in the official GitHub repository[18]. In

[14] https://requirejs.org/
[15] http://www.commonjs.org/
[16] https://babeljs.io/
[17] https://www.npmjs.com/package/psa-lib
[18] https://github.com/Safe-DEED/PSA

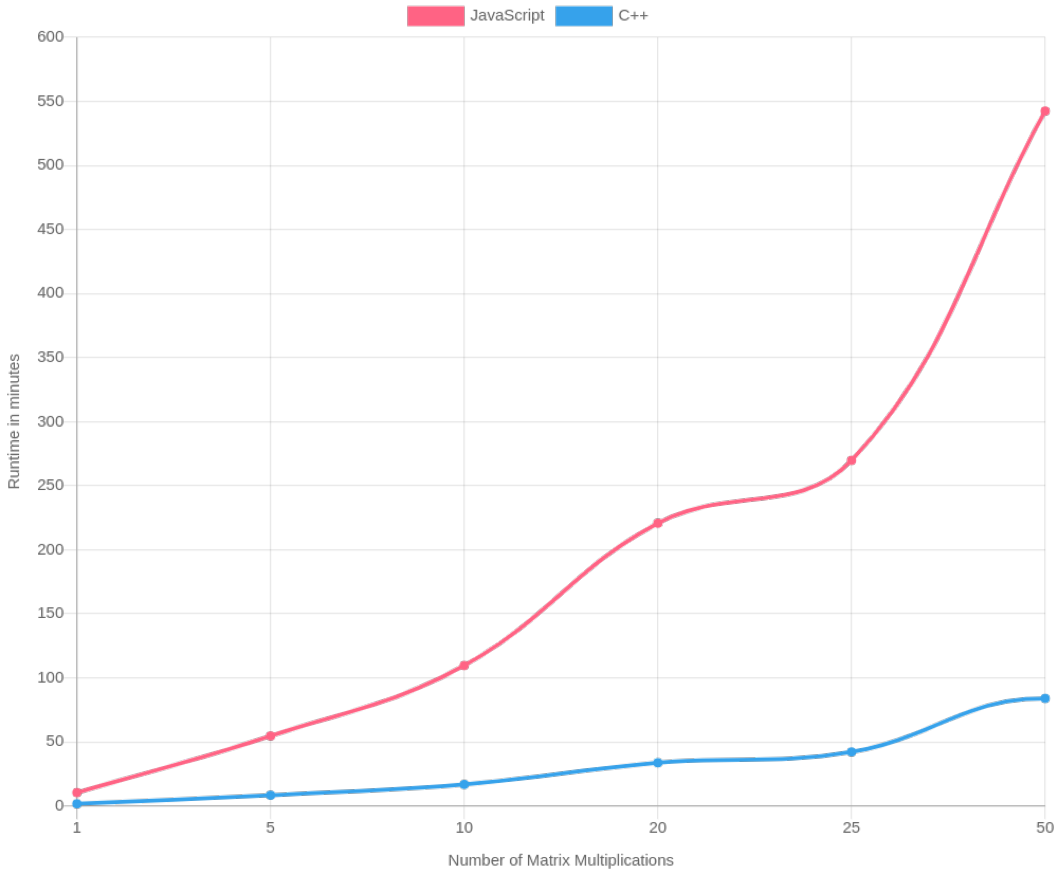


Figure 4.3: PSA runtime of JavaScript implementation compared to C++: time in minutes required to do a certain amount of `MatMul` evaluations using a parameters: $\log_2(p) = 42$, $\log_2(q) = 438$, $n = 16384$, $\kappa = 128$.

the Readme, we discuss the library's installation procedure and give a detailed description about implementation details. If a user already knows which application to build on top of the PSA library, all relevant information can be found in the repository. However, if a user is unsure which problems are best solved with this library, the following section gives an overview of the possibilities. We summarize the most important informations in respect of developing applications with the library in more detail:

1. Client and Server agree on a set of parameters in advance
2. Client and Server instantiate the library on their end (e.g., `import PSA from 'psa-lib'`)
3. Client and Server create an corresponding context with the parameters agreed on
4. Client passes his data to the library for encryption

5. Client transmits resulting encrypted vector to the Server
6. Server passes his matrix and the encrypted Client vector to the library and computes the result vector
7. Server sends result vector back to the Client
8. Client decrypts the resulting vector to obtain the result in plain form

4.2.4.1 Logical separation

To understand the mechanics and in further consequence the proper application of the PSA library, it is most helpful to seperate the library into logical components. The very constituents are states, environments, inputs, and outputs. A depiction of the mentioned seperation is given in Figure 4.4

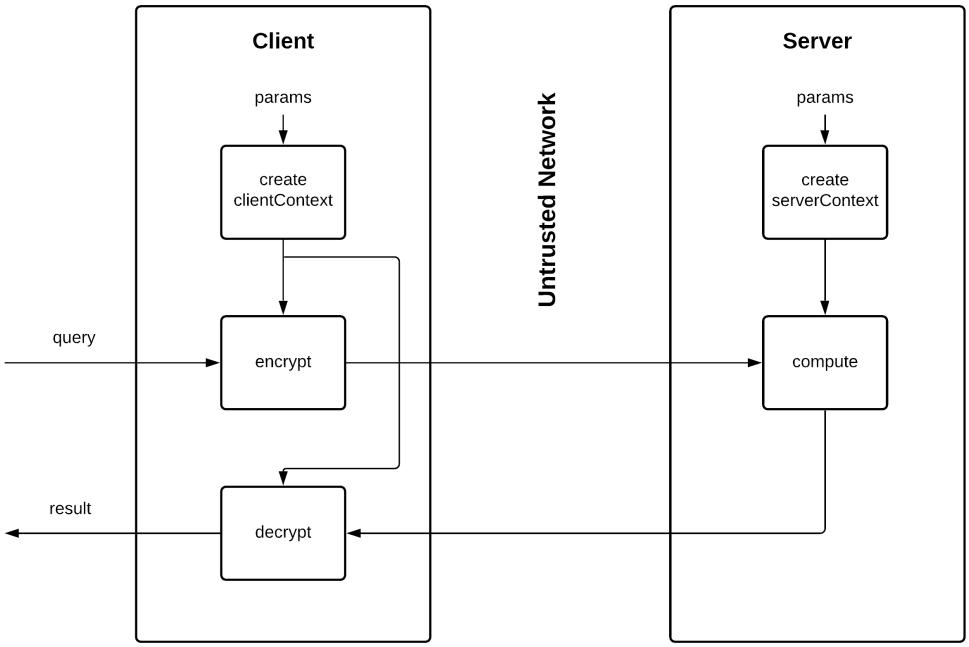


Figure 4.4: The PSA library seperated into its logical components. Here one can clearly see how states are transitioning across environments.

States. There is a direct control flow from the start of the protocol to the end on a very high abstraction level. In between, there are the following states which mostly depend on a previous state:

1. Context Creation

2. Encryption
3. Computation
4. Decryption

Every state is associated with executing a set of tasks that are not crucial at this abstraction level. Although all state transitions happen sequentially, they do not happen in the same environment.

Environments. An environment is a context in which a particular instance of the library is running. This differs from Client to Server. The Server takes a whole different part in the protocol compared to the Client, and the underlying executing platform can be technically different. The user only needs to know whether a particular instance of the library should run in a client or server environment. The library then handles the technical discrepancy. The library's user does not have to worry about running the code in the browser or on a Node.js platform.

Inputs and Outputs. Since PSA is a protocol that demands interaction by two parties, state transitions and their outputs depend on specific inputs. Obtaining the required inputs need inter-environment communication, which is often performed on untrusted networks. The library does not handle the network part since this is highly dependent on the application and should remain the library user's responsibility. The library's function calls yield objects which should be marshaled and sent to the opposing party.

4.2.4.2 Determining Parameters and Execution Environment

Choosing suitable parameters for the library is not trivial and requires careful consideration of the application's characteristics and edge cases. There are some parameters for which the right adjustment can be found very easily and some which may involve trial and error. Since the PSA library works across environments, a user must also decide whether the application will be executed in a browser or on a node environment. In this section, we will describe the parameters as well as the environments and give suggestions on the optimal usage based on our experience.

Parameters. The PSA library needs a block of configuration variables in JavaScript Object Notation (JSON) form. The object is passed into the functions that create the Client and Server context, respectively. The keys in the object have to be chosen carefully. Some of them correspond to the parameters used in BFV [37], the underlying Homomorphic Encryption scheme. We give a description and recommendations of all keys below:

polyModulusDegree: Determines the degree of the polynomial modulus used in BFV. We recommend a value of 4096 when the application runs without any masking and holds limited data. It has to be either 4096, 8192, or 16384. Choosing higher

values for the polynomial modules' degree enables more complex computations but makes computations slower and results in a larger ciphertext size.

plainModulusBitSize: Bit size of the plaintext modulus used in BFV. It must be set according to the maximum values needed in the application. Increasing the modulus results in a larger plaintext size and more noise budget consumption in multiplications. It should be kept as low as possible.

securityLevel: The security level in bits. We recommend 128 bit as this is usually sufficient. A higher security level means higher security overall but slower performance.

compression: The compression method used internally for serializing and deserializing. It can be either *zstd*, *zlib*, or *none*. We recommend using zstd since it is the fastest. However, it may use slightly more memory than zlib.

maskHW: A boolean value determining if hamming weight masking should be applied. If active, the client query vector's hamming weight must have the value of the next parameter *minHW* at minimum. We recommend activating the hamming weight mask, otherwise individual matrix entries can be queried directly.

minHW: If *maskHW* is enabled, minHW determines the HW value the client query vector must have at a minimum. This parameter heavily depends on the application and can be adjusted accordingly.

maskBin: A boolean value determining if binary masking should be enabled, that is restricting the Client to use only zeros and ones in the query vector. If the application requires weighted query entries, we recommend disabling this mask.

createGkIndices: Internally, the library uses a so-called *Galois Key* for rotation operations. When creating the Galois key at the creation of the Client's context, one can enter the rotation indices to make the rotation operations faster. This results in more RAM usage but increases performance in very large applications. We recommend setting this value for anything working with *polyModulusDegree* higher than 4096.

diffPriv: Boolean value determining if differential privacy measures should be applied. This results in a security-utility trade-off. Values drawn from a Laplace-Distribution are mixed into the final result to protect individual data entries. This process is parametrized with *sensitivity* and *epsilon*. We recommend setting this value to true. For detailed information about Differential Privacy and its parameters, we refer back to section 3.1.2.2.

sensitivity: The sensitivity refers to the largest possible difference that a change in data can make. Such a change might expose individual data and should be mitigated in all applications. Sensitivity can be computed exactly for every application.

epsilon: The level of noisiness is controlled with epsilon. Smaller values result in more privacy. However, the utility of the protocol might suffer from a too-small epsilon. Therefore, it should be chosen in a way that results in maximum privacy while still having sufficient utility.

Execution Environment. There are two different environments in which the PSA library can be executed. On one hand there is the browser, on the other hand the Node.js environment. These two environments differ significantly. However, in the case of running a PSA application, everything can be broken down into a few key differences.

Browser. The browser is a computationally weak environment since it usually runs on a personal computer. It offers a graphical user interface and needs user interaction. A browser fits very well when operating as a PSA client. If the data returned by the Server is meant to be visualized, this can be done smoothly in the given environment. It is straightforward to load data from the local filesystem into the browser, and usually the Client's vector is a few factors smaller than the Server's matrix. All the heavy computations are done on the server end, therefore the browser is the optimal environment for Clients with limited memory and computational power but high interactional requirements. A browser as a server will be limited in the variety of applications. It is probably only an option for small PSA applications with low computational requirements since machines with optimized efficiency are usually not set up with a browser, let alone a graphical user interface. Such machines are heavily used in cloud computing and backend environments but rarely to browse the web. Usually, if heavy computational work has to be done in the web, the browser requests help from such a server and relais the computation. This could also be done with PSA. A weak machine, set up with a browser, could be used as the interface to upload data and configure PSA on the server-side. As soon as the PSA client sends its query vector, the request is relayed to a powerful backend server to execute the compute step, optionally transferred back to the server-side browser environment and then sent back to the PSA client. The data can also be sent from the backend directly to the PSA client. Figure 4.5 shows how this is done on a high level. The process of relaying the heavy computation can be made entirely invisbile for the PSA Client.

Node. Node is an environment commonly used to power web backends and other server environments. It does not need any graphical user interface or intervention by a user. In web applications, it usually powers a web server and handles database communication. It was made for computationally intensive tasks and to initially solve the I/O scaling problem [25]. A user cannot interact with Node via a graphical user interface directly but can program it to be as dynamic as required. In the browser scenario, input files and process configurations can be loaded into the browser via the File API. If the files are static or known before the protocol's execution, then a PSA environment based on Node is probably the better and faster choice. Node performs well on I/O tasks. It is possible to set up the PSA client library instance as well as the Server one in a Node

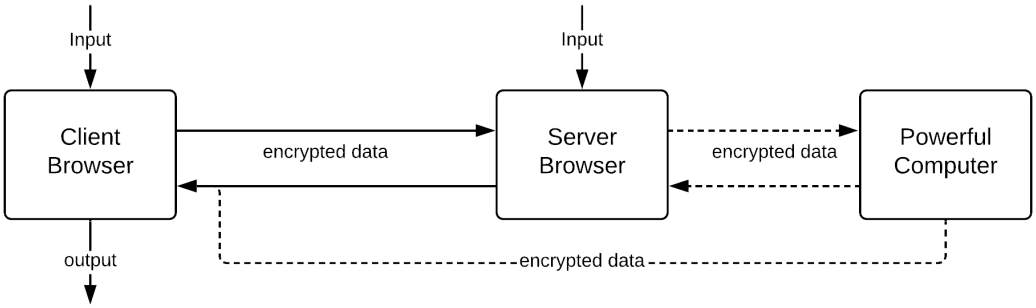


Figure 4.5: A PSA client browser communicating with a PSA server browser. The heavy computation can be relayed to a powerful computer.

environment and not use any browser at all since the library is environment-independent for both the Client and Server end.

4.2.5 Infrastructures in Implementations

After discussing the PSA library's technological basis, its performance, and the fundamental environments in which the PSA library can run, we want to give some recommendations about a real-world infrastructure and communication between Client and Server. Since there are two environments in which the library is able to run, namely browser and Node environments, it follows that there are four setups that are equally valid and should be discussed for future implementations. Additionally, the PSA protocol does not specify a transportation layer or states how to connect the environments. In an actual application, the two environments have to communicate. One option to realize this is via a WebSocket server. This server cannot be hosted in a browser and might need the involvement of a third party. We will shortly discuss the security implications of a third party and introduce the four setups.

4.2.5.1 Security Implications of a Third Party

It should be kept in mind that the WebSocket server only acts as the mediator between the PSA Client and Server since they cannot communicate directly in certain setups. All communication is encrypted, and the third party is not able to read any messages. However, some problems arise as the third party turns out to be malicious:

Denial of Service: The third party could hinder Client and Server from communicating and deny the service. This can be very problematic in applications where, e.g., human lives depend on the output of the protocol. This problem can only be circumvented by placing the WebSocket server at one of the two parties.

Impersonation: Since the Server does not authenticate itself and prove its identity, the malicious third party can impersonate it. This can only be circumvented by introducing authentication into the transport layer or by placing the WebSocket server at one of the two parties.

Replay attack: The original protocol defines a new secret key for every run. However, this can easily be overlooked in implementations. The key could be reused and sockets held open to "improve efficiency". If the secret key is reused, a malicious third party can reuse an old input vector to query the Server or return an old result to the client. This can be circumvented by assuring correct implementation and, therefore, a new key for every protocol run.

4.2.5.2 Browser-to-Browser communication

The browser-to-browser scenario is the only one in which no direct communication can be established since a WebSocket server cannot be hosted in a browser environment. Communication between two browsers must happen through a third party. This schematic is depicted in Figure 4.6. It is an option to host a Node environment separately at the

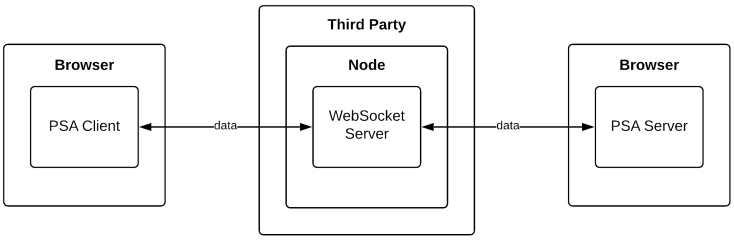


Figure 4.6: The browser-to-browser setup: Communication cannot happen directly since neither of the browser environments can host a WebSocket Server. A third party is needed.

Client or Server side, although this defeats the purpose of this easy-to-use scheme. In this case, it is not anymore as easy as just heading to a website, since either Client or Server needs to execute a Node environment running a WebSocket server in addition to the browser environment. Nevertheless, the data always remains encrypted on its way, and the third party only acts as a mediator. This setup allows for adoption by a really broad field and is very user-friendly. Another option is to make the connection peer-to-peer and let both browsers communicate directly. However, setting up peer-to-peer connections requires loads of configurations and is not trivial in case vast amounts of data are sent.

4.2.5.3 Browser-to-Node communication

This case is probably the most common one since the heavy computation is done on the server end. As a consequence, a solid Node environment on the server side seems to be

the most efficient solution. A big advantage of this setup is that since there is already a Node environment at the Server's end, the WebSocket server can also be placed in this environment, and communication happens directly between Client and Server without the need of a third party. Figure 4.7 gives an idea of this scenario.

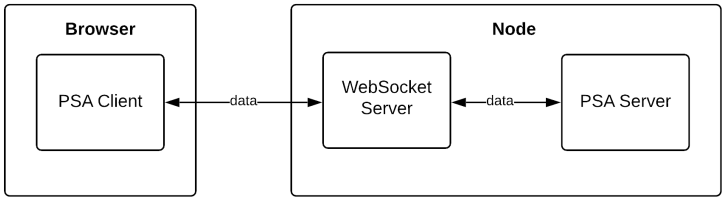


Figure 4.7: The browser-to-node setup: Communication happens directly between the two parties since the WebSocket server is placed in the PSA-Server Node environment.

4.2.5.4 Node-to-Browser communication

The node-to-browser case is probably not the most practical one since the PSA client usually only provides an array to the Server and is not involved in heavy computations. Nevertheless, this case should be discussed. Having a PSA server in a browser at the Server's end seems legit in some cases when uploading of the required files should happen manually. It enables technology laymen to handle the Server's end via the Graphical User Interface (GUI). Fortunately, since the Client is running a Node environment, the WebSocket server can be placed in this environment, and direct communication between the two parties can happen without bringing in a third party. This setup is shown in Figure 4.8.

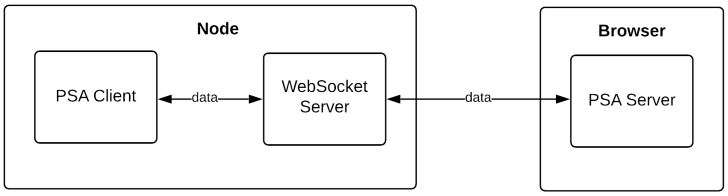
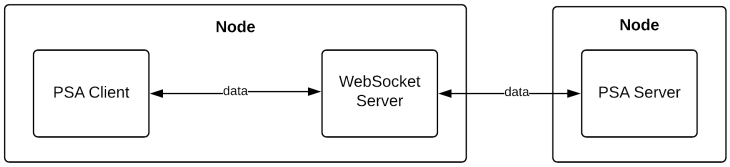


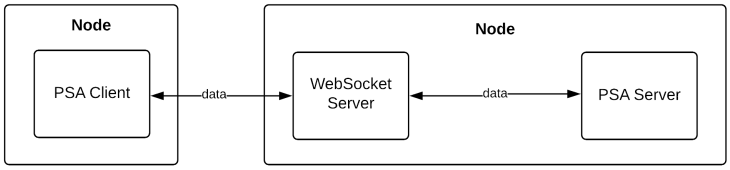
Figure 4.8: The node-to-browser setup: The WebSocket server is placed at the PSA-Client's Node environment such that communication happens directly between the two parties.

4.2.5.5 Node-to-Node communication

Communication between two Node environments is an option, e.g., when the Client and Server are highly automated or perform a high number of I/O operations. In this case, the Web Socket server could be placed at either the Client or the Server. These two scenarios are depicted in Figure 4.9. Again, in this scenario, no third party is required,



(a) WebSocket server at the Client side



(b) WebSocket server at the Server side

Figure 4.9: The node-to-node setup: The WebSocket server is placed either at the Client (a) or the Server side (b). No third party is involved, and applications in this setup can be made highly efficient and dynamic.

and communication happens directly. This setup allows complex constructions and is flexible since Node supports many packages by its package manager acrshortnpm and enables users to build solid APIs around its applications.

4.2.6 Caveats

There are few limitations imposed by the library which users should keep in mind during the development. In the following, we provide some information about constraints and limitations:

BigInt limitations: JavaScript's built-in *Number* primitive type only supports numbers not larger than $2^{53} - 1$. For numbers higher than that, the BigInt[19] interface provides a way to represent whole numbers larger than that. The library utilizes BigInts internally and would therefore allow numbers of arbitrary size. However, the library's node-seal core was compiled with Emscripten, which only supports the wasm32 datatype that cannot handle numbers larger than $2^{32} - 1$. This means,

[19]https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/BigInt

when data is sent from the JavaScript layer to the WebAssembly layer of the application, data gets corrupted if it surpasses the limit. The way this was solved in node-seal is that BigInts are marshaled into strings, passed to the WebAssembly layer, and then, in C++, unmarshalled and converted into a 64-bit data type. It follows that, although a BigInt can theoretically hold numbers of arbitrary size, it is limited to a maximum of 64 bit in the PSA library.

Polynomial modulus limit: We believe that using a polynomial modulus degree higher than 16384 is out of the scope for most of the applications using the JavaScript implementation of PSA and thus do not offer such. Nevertheless, if a library user wishes to use higher degrees, they can be implemented easily.

Chapter 5

PSA In Real World Scenarios

The PSA protocol can be generically applied in many different areas. However, sometimes it is not initially clear if using PSA on a specific problem is the right choice since there is not enough reference available. In this chapter, we comprise some practical PSA approaches with concrete examples to develop an understanding of proper PSA utilization in the reader. We describe specific industry problems and demonstrate solutions based on Private Selective Aggregation. For two of the scenarios, we developed a webapp in JavaScript, utilizing the PSA library.

5.1 Preliminaries

For all PSA scenarios, it is required that Client and Server store their data entries in the same order since they have to match when computing the internal vector-matrx multiplication during the execution of the protocol. There are multiple ways to achieve this. The most secure one would be to use a Private Set Intersection (PSI) protocol. We shortly discussed PSI in section 3.1. For the description of the use cases below, we assume already matching data indices and focus only on the data itself. Additionally, the result of the protocol will be represented by $PSA(\boldsymbol{x}, Z, p)$, where $\boldsymbol{x} \in \mathbb{Z}^N$ is the Client's input vector, $Z \in \mathbb{Z}^{N \times k}$ is the Server's matrix and p is the PSA instance's parameter set. We will not specify the parameter set any further since it depends on the size of the data and the application itself.

5.2 Connecting Mobility to Infectious Diseases

As of April 2020, the world faces an exceptional state due to the SARS-CoV-2 virus and the resulting corona disease. On March 11 2020, the World Health Organization (WHO) declared it as a pandemic[1]. The Austrian government is working in cooperation with mobile carriers[2][3] to track people and monitor the spread of the disease. The scientific community is developing many strategies, like contact tracing [28, 4, 21], anonymous

[1] https://www.euro.who.int/de/health-topics/health-emergencies/coronavirus-covid-19/novel-coronavirus-2019-ncov

[2] https://www.reuters.com/article/us-health-coronavirus-europe-telecoms/european-mobile-operators-share-data-for-coronavirus-fight-idUSKBN2152C2

[3] https://www.trendingtopics.at/coronavirus-a1-liefert-bewegungsprofile-der-bevoelkerung-an-regierung/

collocation discovery [20], decentralized contact tracing [9] and other approaches with location data. The Corona Heatmap[4] is another contribution based on PSA. Currently, there is a C++ implementation [2]. We contribute an implementation in JavaScript based on our PSA library. We will describe the JavaScript implementation in detail later in this section and want to stress, that the implementation is not an official application but rather a showcase pointing out how this system could be used.

5.2.1 Corona Heatmap Scenario

To monitor the spread of the disease, one idea is to privately link mobility to individual infection. The idea is based on the fact that both, Mobile Carrier (MC) and the Ministry of Health (MH), possess a list of phone numbers with a certain intersection. This intersection is of great significance since, for every phone number in this set, the MH knows whether it is linked to a positively tested person, while the MC possesses a comprehensive geographical footprint of this person. This is enough information to draw a heatmap depicting the spread of the disease. However, due to user privacy protected by the General Data Protection Regulation (GDPR) and restrictions along with it, this is not a trivial problem. It would pose a privacy intrusion if two datasets holding information about individuals were linked naively. This process has to be done privately without exposing any data. Private Selective Aggregation is of optimal use here since it allows for a confidential generation of the said heatmap. The MH queries the MC for the geographical footprint of positively tested people by providing information about them in encrypted form. The MC generates a list with one entry per person in his network. Each entry holds the duration a given person was registered by any cell tower among the observed ones. The list remains encrypted at any point in time, such that the MC does not learn anything. In the end, the list is sent to the MH, which decrypts it and uses it to generate a heatmap based on the cell tower locations and the total durations spent at each location. This process is shown schematically in Figure 5.1.

5.2.1.1 Ministry of Health

The Client in this scenario is the Ministry of Health. If people are infected with the virus and tested positively, the ministry has information about this. As the query vector of length N, we define $\boldsymbol{x} = (x_1, x_2, ..., x_N)$, where $x_i = 1$ if the patient at index i is infected. If the patient at index i is not infected or i is an index generated due to the preceding execution of an PSI protocol or another process to get matching indices, then $x_i = 0$.

5.2.1.2 Mobile Carrier

The Server is the Mobile Carrier that holds a matrix Z with dimensions $N \times k$. Every row corresponds to a client c_i, where i is in the range from $1, ..., N$ and every column

[4] https://covid-heatmap.iaik.tugraz.at/en/

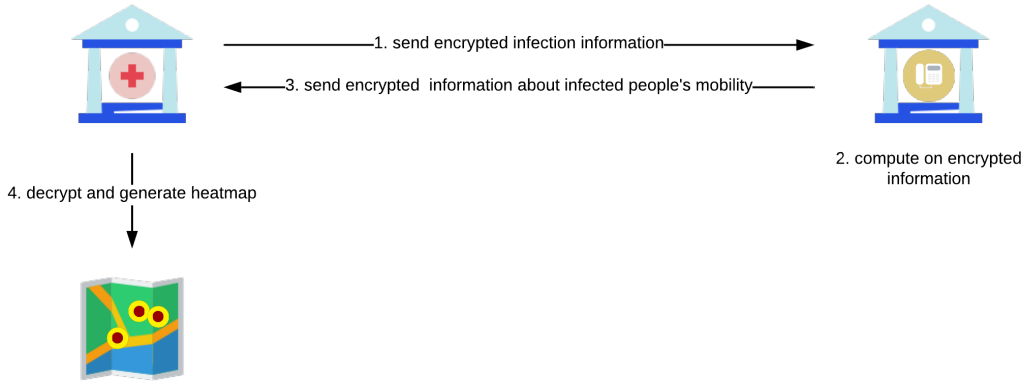


Figure 5.1: A high level depiction of the steps required to generate the Corona Heatmap.

corresponds to a cell tower t_j, where j is in the range $1, ..., k$. An entry $z_{i,j}$ in the matrix tells the exact duration a client c_i was present at a specific cell tower t_j.

5.2.1.3 Result

When computing on the query vector and the matrix, the result is a vector holding the aggregated presence durations at towers $t_1, ..., t_k$. In means of this vector, a heatmap can be drawn. Cell towers with higher presence durations hint towards a hotspot of infected people.

5.2.2 Implementation

For this thesis's purpose, we implemented a showcase for the Corona Heatmap by utilizing JavaScript in combination with the PSA library. We chose to use the browser-to-browser setup from section 4.2.5. On top, we built an infrastructure for communication between Client and Server based on Node.js and added a front-end based on the popular front-end library React[5]. In this section, we are describing general design decisions and implementation details. This serves as a guide for possible future projects evolving from our work. If peers decide to extend this project or simply use it, this section will give the necessary technical information.

5.2.2.1 Frameworks

The web is a fast-paced environment. One should evaluate the decision of which frameworks to use and which technology to use very carefully. We intended to build the Corona

[5]https://reactjs.org/

Heatmap with the most modern and most easy-to-use tools and still assure to some level that that those tools will be relevant and in use for a longer period of time. It is a very tiring process to learn an outdated framework to be able to work on an old project. The most trivial idea is not to use any frameworks at all. However, the cost of developing and maintaining an application based on the plain web languages JavaScript, HTML, and CSS, rises very fast with the project size. Therefore, we decided to use a few libraries for the sake of easy maintenance.

Front-end implementation details. For the development of the front-end, we had to choose between React, Angular, and Vue. Those are the most used frameworks and have been well established for a few years. Angular had many breaking changes in the past and is a very heavy framework. Although Google supports it, it would be too much of an overhead to use it here because of all the boilerplate code needed to run a simple application. Vue is very lightweight but not as well established as React and Angular. Many developers nowadays head for either React or Angular, so the probability of losing Vue as a web framework is much higher than React or Angular. We chose React as the best fitting option since it is supported by Facebook and easy to use due to its component-based nature. Our focus heavily relies on separating implementation concerns to reuse code and make application maintenance as easy as possible. React is the optimal tool for that. In addition, we use Material-UI[6] as a components library. This library provides web components for easier building of modern websites based on Google's design guideline Material Design and works in conjunction with React. It offers ready-to-use buttons, navbars, sliders, and much more.

Back-end details. The front-end requires a server to handle requests, provide files, and store data. This unit is called the back-end. There are many options in our case. One possibility is to use a Java back-end with the popular Spring framework, Python with Django or Flask, or Node with Express.js. Spring is well established and is well supported by its online community. Python is a very easy-to-use language, and Django works quite well. However, we decided to use Express.js because more and more developers are using it, which leads to a higher chance for long-time adoption. It works very well with heavy I/O work. The development happens in JavaScript. It follows that a developer is only required to learn JavaScript to work on the front- and back-end. These reasons lead us to the decision to go for Node with Express.js. For the communication between Client and Server we set up a WebSocket server based on the NPM package *websocket*[7]. This library provides a pure implementation of the WebSocket standard.

5.2.2.2 Application Infrastructure

When executing the Corona Heatmap, Client and Server communicate with each other. The PSA protocol itself does not specify a transport layer, so implementations have

[6] https://material-ui.com/
[7] https://www.npmjs.com/package/websocket

to specify this layer independently. The client, as well as the Server, can either be implemented as browser or Node end. In this implementation, we chose to use a browser-to-browser case. In our opinion, this case is the most remarkable one since users do not have to install anything and can simply head to a website. However, since we chose the browser-to-browser case, communication between the two parties is not trivial. We use the WebSocket standard, which is built on top of TCP. Unfortunately, using Web sockets implies that Client and Server cannot communicate directly in the browser-to-browser case. Communication happens through a server that is hosted either by the Client, the Server, or a third party and is encrypted all the way. In the browser-to-browser case, an additional Node environment must be installed at the Client or Server end since the WebSocket server cannot be hosted in a browser environment directly but needs an own Node environment on the local machine. This WebSocket Server acts as a mediator between the two parties. In case the PSA users are not familiar with setting up a Node application, a third party should provide the WebSocket server. Communicating directly from one browser to another is called peer-to-peer. There is already a standard called WebRTC[8]. However, setting up peer-to-peer connections with WebRTC still requires a server to distribute session info and handling errors. Moreover, the setup is not as straightforward as in the case of WebSockets. The extension of the Corona Heatmap with a peer-to-peer feature could be addressed in future research projects. For the Corona Heatmap showcase, we chose the browser-to-browser setup since it allows easy execution from everywhere and offers the implementation of a Graphical User Interface. However, all of the four possible setups from section 4.2.5 are valid. Developers should carefully evaluate the circumstances in which the Corona Heatmap is used and then decide which setup fits best.

5.2.2.3 Dataset

For the showcase, we used a dataset from Gowalla [26], a location-based social networking website where users can check-in at locations shared by the community. For our purpose, we trimmed the dataset to lower the computational effort since this is not the focus of the showcase. The full dataset is available at the Stanford Network Analysis Project[9]. The dataset shows how many times users checked in at different locations. We use the locations as the Mobile Carrier cell towers and the number of check-ins as the time unit to measure the individual's duration at some particular cell tower. The trimmed dataset consists of $N = 10000$ individuals and $k = 3755$ cell towers and consists of two separate parts:

Data file: This file contains the matrix $Z \in \mathbb{Z}^{N \times k}$. It is in Comma-Separated Values (CSV) format without a header and consists only of numbers. There are exactly N rows in the file, each carrying k values. Of course, every row corresponds to one individual and shows how often this individual was at any of the k locations. We decided to use CSV as the file format in this application since it is a widely

[8] https://webrtc.org/

[9] https://snap.stanford.edu/data/loc-gowalla.html

known and well-established format for data. The number of possible formats for the application could easily be extended if wanted.

Mapping file: The mapping file, again in CSV format, holds the mapping from location index to longitude-latitude. It carries N rows and three values per row. The first one is the location index in the matrix, the second one is the longitude, and the third one holds the latitude of this location. This file is mainly to generate the heatmap at the Client end.

The showcase is not limited to this particular dataset and can be used with data of arbitrary size. We recommend using our files since we think this parametrization fits the application in the best way in terms of runtime and scope.

5.2.2.4 Inputs

For our showcase, we use four input files. As already mentioned, the Client vector's indices need to match the Server's matrix indices for PSA. In our showcase, this is not the case. At the beginning of the communication, the Mobile Carrier sends all of his N numbers in the exact order used in his matrix Z to the Client, who then constructs a new encrypted binary vector $\boldsymbol{c}$ that holds N values which are only set to 1 if this number belongs to a positively tested person. The Client has to provide a file with only one column holding all of the infected people's phone numbers. The Server has to provide a similar file holding all the phone numbers of his clients. Additionally, the Client has to provide the mapping file for the generation of the heatmap and the Server has to provide his data file (the matrix Z).

5.2.2.5 Drawing of the Heatmap

After the protocol is executed and the resulting vector is available to the Client, the heatmap can then be constructed. Since drawing the hotspots involves a map of the real world and some mechanisms to generate a heatmap layer on top, we used the popular web mapping library Leaflet[10]. Under the hood, Leaflet uses the open-source project OpenStreetMap[11] as its underlying map. Since we use React for development, the usage of a wrapper for the heatmap layer[12] was required. This way, the heatmap is encapsulated and easy to extend and adjust. Figure 5.2 shows an example of the heatmap, generated by our Corona Heatmap application, positioned over Vienna. One can observe how Mariahilfer street and Stephansplatz are the *hottest* places. We used the dataset that is described above.

[10] https://leafletjs.com/
[11] https://www.openstreetmap.org
[12] https://www.npmjs.com/package/react-leaflet-heatmap-layer

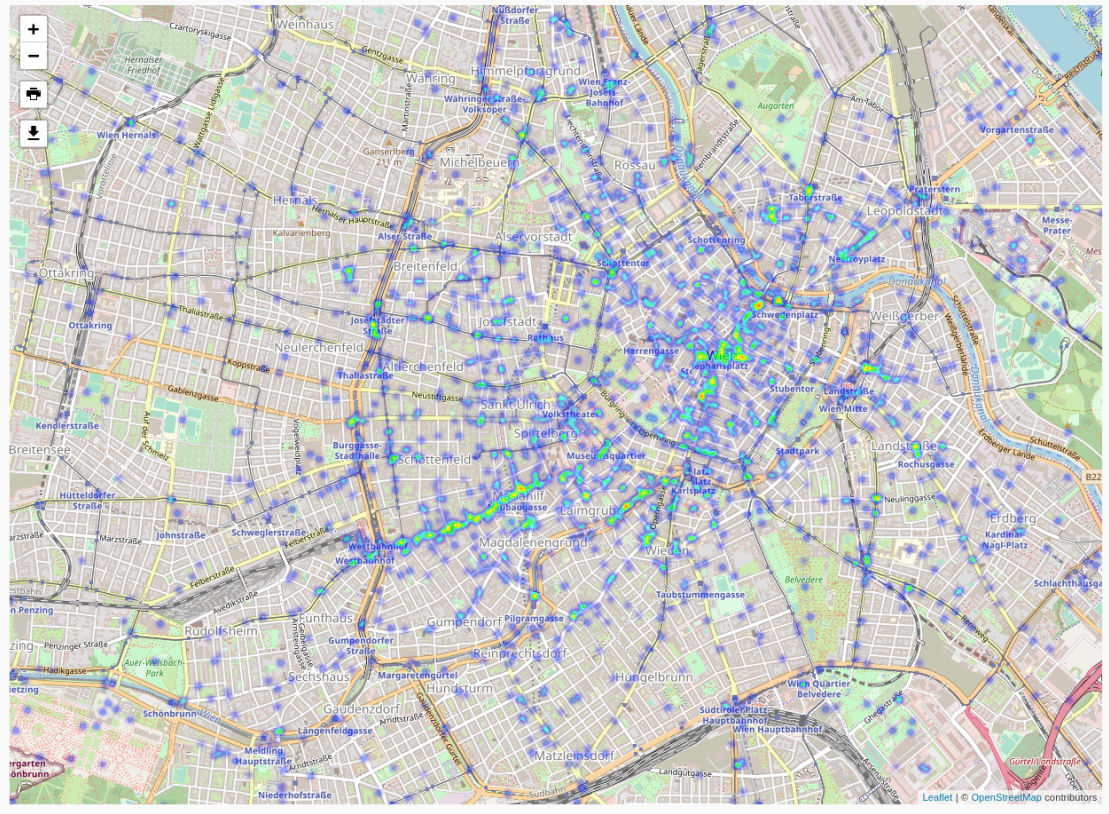


Figure 5.2: Heatmap of Vienna using sample data from Gowalla [26].

5.2.2.6 Choosing Epsilon

We experimented with various settings of ϵ in the range of 0.01 to 7 and found a value of 1.5 to be most appropriate since existing hotspots grow negligibly and no new ones are generated, even though there is generated a high amount of noise. However, we want to stress that our findings are based on our specific heuristic and must be re-evaluated in future applications. The original heatmap without noise is shown in Figure 5.3. Assuming that the heatmap is generated hourly, the sensitivity Δq of the system is 1 since a person can only stay at a maximum of one hour at a specific cell tower. We used the dataset that is described in section 5.2.2.3. As already mentioned, the dataset contains $N = 10000$ individuals and $k = 3755$ cell towers. The optimal value for ϵ is 1.5 since there is clearly more noise but no new hotspot. The resulting heatmap is shown in Figure 5.4. Setting ϵ as low as 0.5 (Figure 5.5) results in too high levels of noise in the output. This value of ϵ leads to additional hotspots at places which are not present in the original picture. Setting ϵ to 5 yielded a result (Figure 5.6) that is nearly indistinguishable from the original picture. Changing a single data entry with such an ϵ could allow an attacker

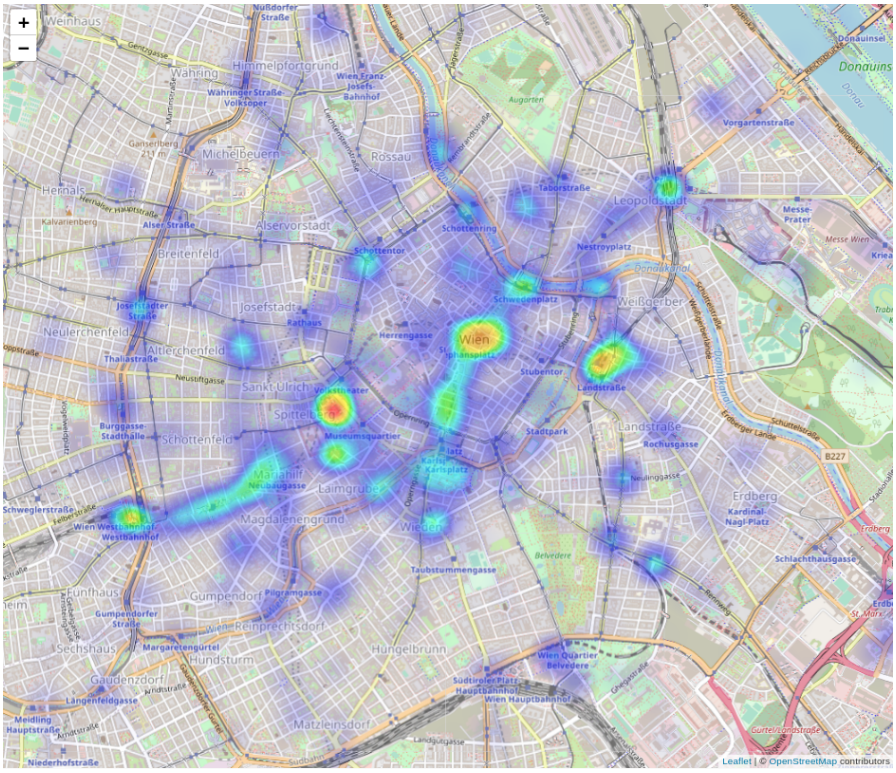


Figure 5.3: Corona Heatmap without Differential Privacy.

to gain knowledge about the corresponding individual. Therefore, the value is too high. Privacy of individuals is in danger when using an ϵ as high as this.

5.2.3 Usage of the Application

The usage of the Corona Heatmap application is straightforward, provided that the user knows the concept of PSA and details of the general idea of the Corona Heatmap. Since this application is deployed in the web, various people from different sectors are privileged to use it. To make the application accessible to a wide crowd, we describe PSA's concept as simple as possible.

5.2.3.1 Back-end

The website for the Corona Heatmap must be hosted on a server. This server can be at any of the two parties' ends or at a third party. It is very straightforward to start the web and socket server since start-up scripts are provided with the code files. We followed the standard rules to start a Node application which will not lead to problems for users who are familiar with Node. The default port for the application is 3000. The port must be provided to the users if no reverse proxy is used.

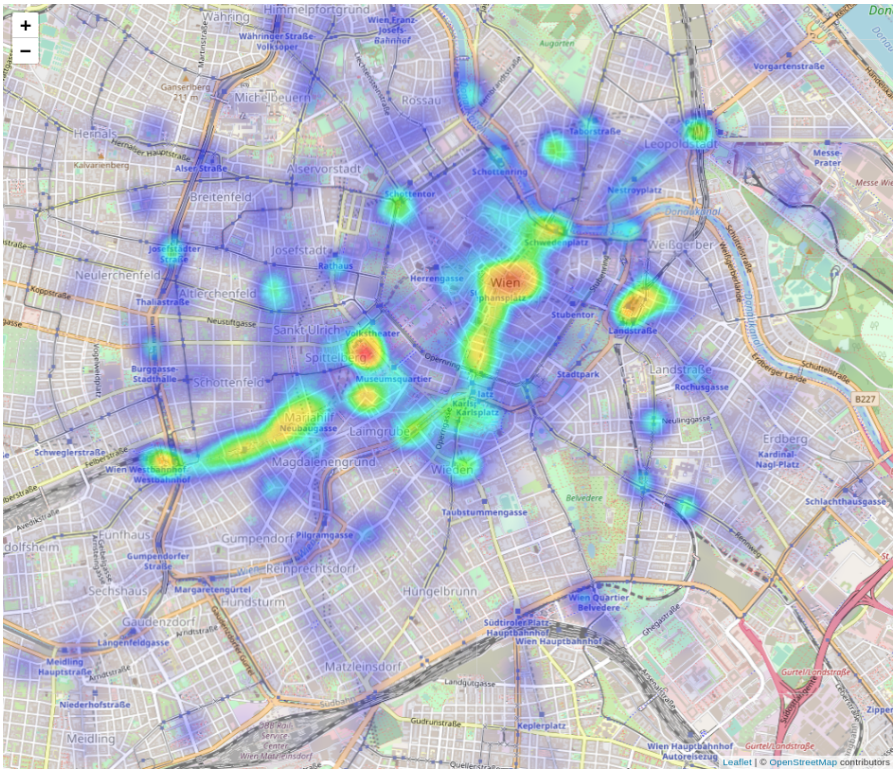


Figure 5.4: Corona Heatmap with Differential Privacy parameter ϵ set to 1.5.

5.2.3.2 Front-end

The front-end of the application is the part that is directly visible to the user in the browser and, therefore, the interface between the user and the application. It should be designed as intuitive as possible. In our application, a user visits the link given by the website's hoster. This link will point to the main entry point of the Corona Heatmap web app. On the start page, there are only two options from which the Client and Server must choose their role and follow the protocol. The Ministry of Health uploads the two files holding the phone numbers and the mapping file as well as the Mobile Carrier's phone numbers and the corresponding matrix. Both parties start the protocol by clicking the start button. After execution, the heatmap is automatically generated at the Client's view and ready to be browsed and even downloaded.

5.2.4 Caveats

Corona Heatmap servers as a showcase and has some limitations we want to discuss below.

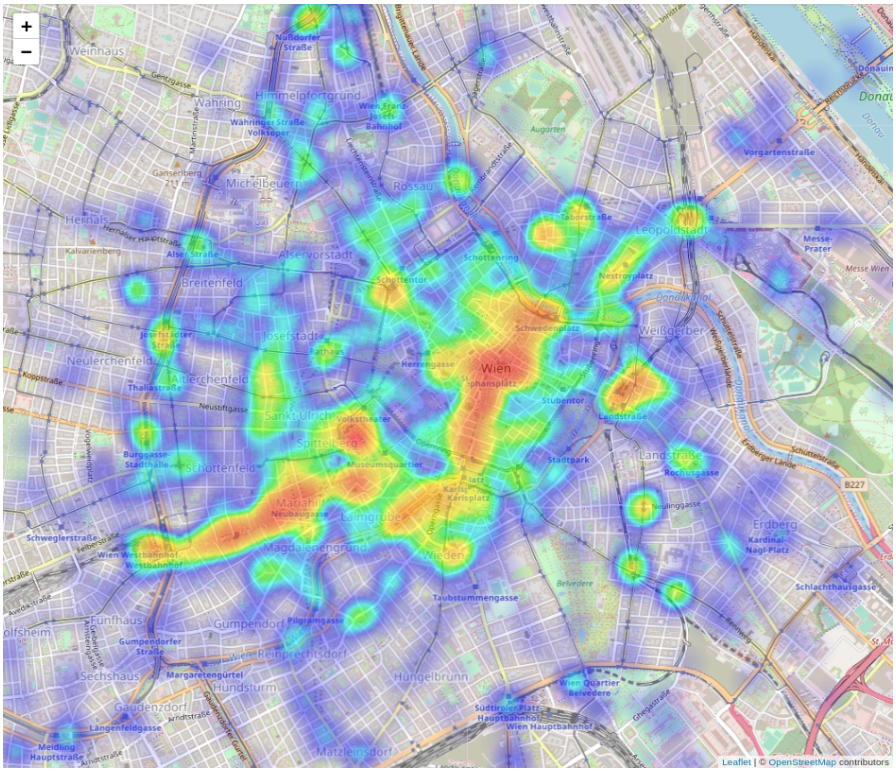


Figure 5.5: Corona Heatmap with Differential Privacy parameter ϵ set to 0.5.

Sessions: We designed Corona Heatmap in a way that only one Client and one Server can participate in the protocol. It is impossible to run multiple Client-Server sessions simultaneously since this would require a more sophisticated configuration of the WebSocket server and some kind of identifier attached to each session and would exceed the scope of this work. Nevertheless, due to highly modular code, the modification required to achieve this would be neglectable and can be done in future works.

File loading: Since PSA usually works with vast amounts of data, the files that are loaded into each environment, especially the Server one, can be quite large. Therefore, users should be patient and wait until the File API finished handling the loading process.

Freezing: During the execution of the protocol, the server view freezes completely. However, this does not indicate an error during processing. In JavaScript, there is only one thread that handles the main routine. In this case, it handles the PSA protocol and can therefore not perform the necessary operations. Nevertheless, we included console logs in the code to give the user the possibility to check the thread's activity.

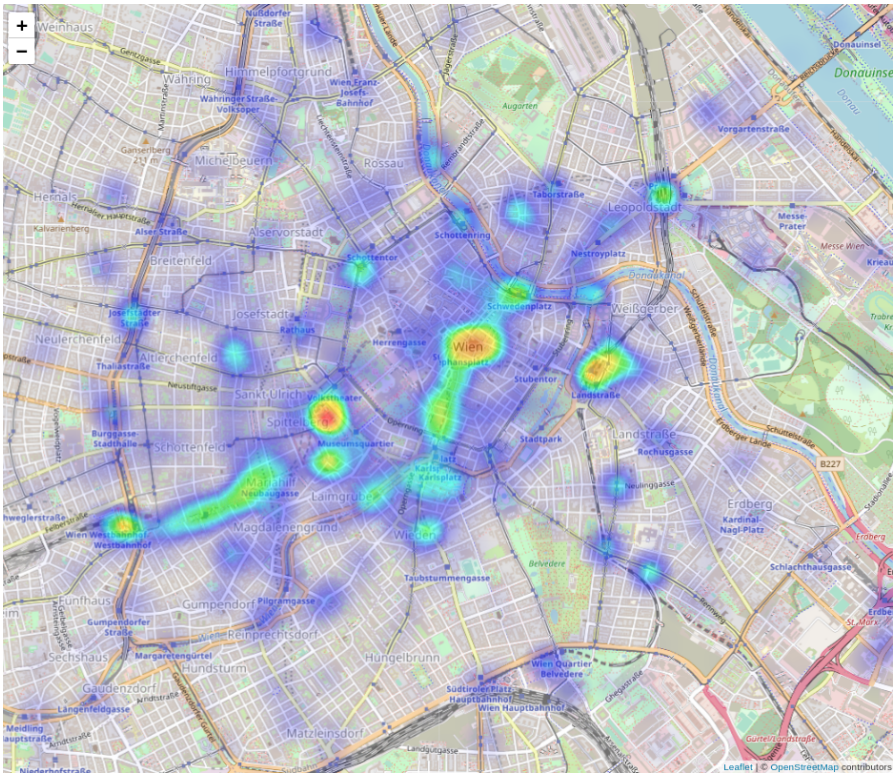


Figure 5.6: Corona Heatmap with Differential Privacy parameter ϵ set to 5.

Direct paths: The web app is started by heading to the Uniform Resource Locator (URL) provided by the web server's hoster. It will not work to append `/client` or `/server` directly since configurations essential to the protocol must be done before the specific view can be entered.

5.3 Risk Assessments

An interesting use case for Private Selective Aggregation is in the field of risk assessments. PSA enables an investor to estimate an investment risk. This would allow companies or investors to gain knowledge about investment subjects and mitigate possible failures along the way. A Financial Institution (FI) that is aware of a large customer pool's credit standing and financial stability is required. The FI creates a set of assessments or scores based on chosen or predefined variables for each customer and constructs the matrix Z. The Investor can then query this matrix with the query vector $\boldsymbol{x}$ to gain insights into the financial stability of a ceratin set of interest and lower the investment risk. This process is shown in Figure 5.7. The Investor might be troubled about giving away information about his customers. However, since this process is based on PSA, the institution does not learn anything about them or the specific investment subject, and the Investor will

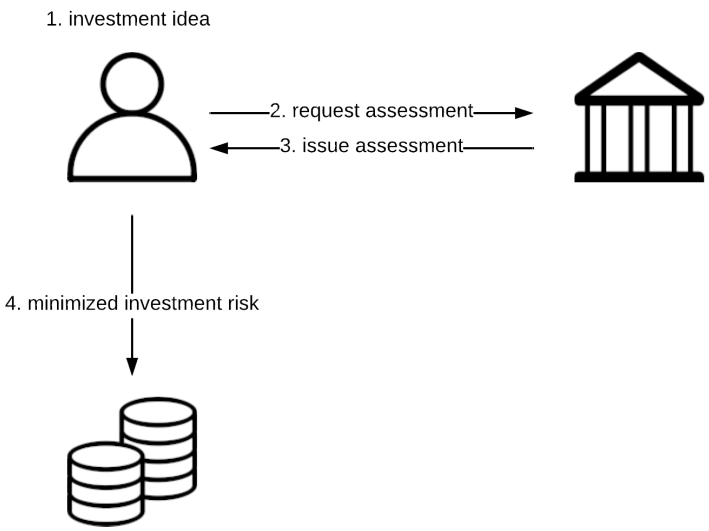


Figure 5.7: Private Selective Aggregation applied to the area of investments. An investor can lower the risk of a bad investment by considering a rating issued by a credit institution.

not gain insights into individual data, provided that all privacy recommendations are followed.

5.3.1 Rating

For every application, the individual scores of the rating must be chosen. The FI can determine them on its own. However, it is also an option that the Investor issues a request for certain scores. This is free to choose and the protocol remains dynamic in this choice. In many cases, the FI will already have an internal rating scheme. In the area of credit scoring, there are many well-established methods based on Classification and Regression Trees, Neural Networks, and k-nearest Neighbor Classifiers [68]. Depending on the exact implementation, the FI could try to convert it to be used in this application. It does not matter how the rating is done, the only requirement is the possibility to convert the rating into numerical form, such that it can be summarized and later averaged. After the protocol is finished, the Investor will hold a vector of aggregated values and might not be able to make immediate sense of it. Therefore, in a few cases, it might be necessary that the institution provides a mapping to the Investor, which clearly shows how to convert the rating in numerical form back to the original form.

Scheme. Certain parameters or scores must be defined to issue an assessment or rating that measures the subject's stability. Indicators of financial stability or creditworthiness are debt, liquidity, profitability, activity, as well as other indicators [3]. Some of them are of particular interest to us and fit most of the applications:

Payment History: A financially stable subject makes payments on time and has a history of only minimal delays. It is not a good sign if a subject was bankrupt or had his accounts sent to collections.

Amount Owed If massive portions of available assets are borrowed or in the form of credits, this can lead to financial instability. However, the institution must inspect the subject's responsibility thoroughly since borrowed assets can help to thrive and are not bad by default.

Liquidity: Subjects with steady, available financial resources to meet obligations should get a higher score. Solvent subjects are more likely to stay active in the market and are ready to pay for sudden, unforeseen events.

Activity: Financial activity of a subject indicates consistent management and commitment to assets. A subject might open new accounts, add new subscriptions or spend money regularly.

Responsibility: Stable subjects spend money responsibly. It is not a good sign if the subject holds more assets than appropriate or spends money in unreasonable ways. Responsible customers may hold different accounts for different concerns like loans, mortgages, or credit cards.

This list can be extended as desired. However, the calculation of the rating is not in the scope of this work and remains the task of the institution that is performing the rating in the first place.

5.3.2 Financial Stability Report

As the first application for risk assessments, we introduce the issuing of Financial Stability Reports via Private Selective Aggregation. This report could help a company to lower internal investment risks. In this scenario, the objective to be measured is not the investment subject itself but the steady income flow.

5.3.2.1 Scenario

Company A is selling a product in the Software as a Service (SaaS) sector. They offer the product online and provide the necessary infrastructure to their customers. This product is a modern Enterprise Ressource Planning (ERP) system with many additional modules and plugins. Company A's customers, mostly small to medium-sized companies, are subscribed to the software and pay monthly fees. Company A depends on its customers' credit standing since its business model builds upon subscriptions with monthly payments.

The company's shareholders consider investing a significant amount of money into a new module with technology entirely new to the market. They are not sure if this module will fail and cost them money to compensate for the failure. Since it is tough to analyze the market in advance when a product is completely new, this turns out to be a risky project. If some customers quit their services during the time of the expenditure, the company would run into financial problems. The management would like to have more certainty before starting this operation. The customer's financial stability is an indicator whether the customers will keep paying for company A's services.

5.3.2.2 The Company as the Client

In this scenario, the Client is Company A. The input vector $\boldsymbol{x}$ is a vector holding the customers' identifiers x_i of interest. A possible company's identifier could be its name or the International Bank Account Number (IBAN) of a bank account. The term "of interest" describes that some customers may be irrelevant and therefore not essential and used for the assessment. Moreover, a customer could be a company or even a governmental institution. The input vector is defined as

5.3.2.3 Financial Institution as the Server

The Server is a bank or some institution aware of the customers' respective financial ratings. A bank knows about the assets of their customers. How liquid they are, where they invest, and how much money they have. The bank needs to assign ratings to the companies. It is essential for the protocol that, in the end, the ratings are numbers from 0 up to an arbitrary value. A common way of classification in the credit rating industry is using letters like AAA (triple-A rating), the FISCO Score Factors[13], or VantageScore Factors[14]. However, it is not mandatory for an institution to use these systems. For the protocol it is only required that the ratings are mapped to a range of numbers ultimately. The matrix Z is constructed with one row for every of the N customer, holding one column for every of the k scores. An entry $z_{i,j}$ represents the score at index j for the customer at index i, for i in $1, ..., N$ and j in $1, ..., k$.

5.3.2.4 Generating the Measurement

The resulting vector $\boldsymbol{h}$ holds the summarized rating of each score for each customer. It can be transferred to the Client again, who decrypts it and divides every entry by the number of all customers N to get the Financial Stability Report. The company now has a measurement of its customers' average financial standing, which will give a rough estimation of the liquidity and other factors of their customer pool. For investments of higher risk, a higher score is recommended in the areas of interest.

[13] https://www.myfico.com/credit-education/whats-in-your-credit-score
[14] https://vantagescore.com/

5.3.3 Estimating Financial Standing of Investment Subjects

Investing in stocks is an example for a very common financial strategy not only for companies but also for private persons. Since these investments always come with some risk, it is in the Investor's interest to minimize that risk. A financial institution can help the Investor by generating scores for each of the investment subjects and issue a report privately by using Private Selective Aggregation. In the following, we describe a real-world scenario.

5.3.3.1 Scenario

A, who is either a person or a company, plans to expand his investment portfolio. To achieve this in a way that maximizes profit and minimizes risk, Person A needs to invest in companies with a high rate of success and growth, healthy history, a solid customer base, and, most importantly, a safe financial standing. Unfortunately, A does not gain reliable knowledge about a particular company's financial situation by just looking at the stocks since its value is merely determined by supply and demand. Organizations like the Austrian association for the protection of creditors (Kreditschutzverband, KSV)[15] offer a rough estimation. However, since Person A invests a high amount of money, it is essential to use every bit of information.

5.3.3.2 Investor as the Client

In this scenario, the Client is Person A since he owns the list of companies he wants to invest in. The input vector is a vector $\boldsymbol{x}$ holding the identifiers of the companies of interest. The identifiers could be the company names. There might be the need to combine the names with the sector since some countries allow the same name for different sectors which would result in ambiguities.

5.3.3.3 Credit Institution as Server

The Server is, same as in Example 1, a bank or some financial institution that is aware of the companies' respective financial circumstances. Such an institution knows about the companies' assets, how liquid they are, where they invest and how much money and debt they have. Based on this knowledge, the institution can generate a report by assigning ratings to the companies. It is essential for the protocol that, in the end, the ratings are numbers from 0 up to an arbitrary value. The requirements on the Server's matrix are the same as in the first example described in section 5.3.3.1. The resulting ratings are required to be mapped to numbers, with 0 being the lowest and counting up from there to the highest rating. The institution constructs the matrix Z with k scores for each of the N companies:

[15] https://www.ksv.at/en

5.3.3.4 Aggregated ratings

The resulting vector holds the summarized ratings of all the companies, which can be transferred to the Client again, who then calculates an average over N for every entry. Person A now has a measurement of its company pool's overall credit standing, which will give a rough estimation of how safe the investment is.

5.3.4 Implementation

For the purpose of demonstration, we implemented a showcase for Private Selective Aggregation applied to the generation of a Financial Stability Report in JavaScript, utilizing the PSA library. We use the browser-to-browser setup since it offered us the possibility to implement a GUI and is accessible by every digital system in possession of a web browser. We tried to keep the application as simple as possible. Client as well as Server, in this case, Company and Credit Institution, are required to visit a certain webpage and follow the links to their individual interfaces. Each party uploads their respective data file and starts the protocol. After some time when the computation is finished, the Financial Stability Report is displayed in the company's view in the form of a bar chart. Figure 5.8 depicts a sample report generated in the app. The chart displays the Financial Stability Report over 10000 customers. The chart can be saved to the local disk for later inspection.

5.3.4.1 Frameworks

In our implementation, we tried to keep the amount of frameworks as low as possible, but still tried to guarantee some convenience for future developers. The reasoning about the decisions we made along the way is equal to the one in the Corona Heatmap in section 5.2.2 since those two applications are very similar in their nature.

Front-end libraries. We used the popular library React for the front-end since it offered us the possibility to separate the concerns in the application and write highly modular code. We used some React components from the Material-UI library since this enabled us to write an appealing GUI without unnecessary effort.

Back-end libraries. For the back-end that runs the webserver, we used Express.js in a Node environment. Express is easy to use which enabled us to do the set up fast. Since we chose the browser-to-browser setup from section 4.2.5, we were required to host a WebSocket server. For the WebSocket part we again used the *websocket* package from NPM.

5.3.4.2 Application infrastructure

For the communication between Client and Server, we used a transport layer based on web sockets. Since two browsers cannot communicate directly via web sockets, we introduced a Node server running Express and the WebSocket server. It can either

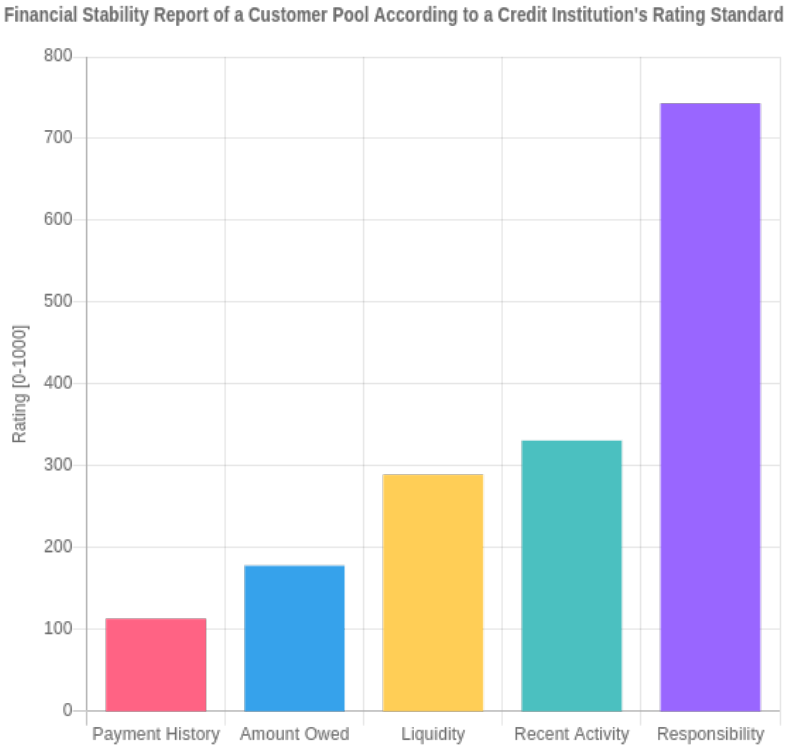


Figure 5.8: Bar chart showing various financial stability ratings of a set of 10000 customers. In this example, the ratings reach from 0-1000 for five different scores for randomly generated data.

be executed by a third party, the Client, or the Server. We discussed each scenario's implications in section 4.2.5, whereby the browser-to-browser case is the most important one since it is the one used here.

5.3.4.3 Dataset

For our dataset, we generated a list of $N = 10000$ random IBANs, which serve as the input file for the Client-side. For the Server side, we used $k = 5$ different scores, namely the ones we described in section 5.3.1. We generated the scores arbitrarily and assigned them to each of the N IBANs. Both files are in CSV file format.

5.3.4.4 Inputs

For the execution of the protocol only two inputs are needed:

IBAN file: The CSV file holding all the IBANs of the company's customers. Each row only contains one IBAN. The header in the CSV file is omitted.

Rating file: A file in CSV format without a header. It contains five scores in the range 0...1000 for every IBAN. Every row holds the IBAN as its first value. The five scores are saved in subsequent columns, separated by commas.

Both inputs are provided to the application via the respective interfaces.

5.3.4.5 Generating a Bar Chart

For drawing the bar chart that finally shows the Financial Stability Report after the protocol's execution, we used the Chart.js[16] library. It allowed us to draw the chart by only setting a few parameters and the library then generates the plot. The plot is two dimensional and based on the HTML canvas[17], which allows simple downloading of an image. The x-axis shows a bar with different colors indicating a different score. The y-axis depicts the extend of the score.

5.3.4.6 Choosing Epsilon

The result of the protocol is a vector $\boldsymbol{h}$ where every entry $h_i \in \mathbb{N}$ is the sum of N subscores, and i is in the range $1, ..., k$. Therefore, the Investor needs to divide every h_i by N to obtain the averaged result. A malicious Investor could form two queries $\boldsymbol{x}$ and $\boldsymbol{x'}$ that differ in only a single entry. That is, $\boldsymbol{x} = (x_1, x_2, ..., x_N)$ and $\boldsymbol{x'} = (x_1, x_2, ..., x_{N-1})$. After running the protocol, the query $\boldsymbol{x}$ will result in the output $\boldsymbol{h}$ and $\boldsymbol{x'}$ in $\boldsymbol{h'}$. By simply computing $\boldsymbol{h} - \boldsymbol{h'}$, the malicious Investor learns the exact contribution of the single entry that was not included in $\boldsymbol{x'}$. That is, the Investor learns a specific row $\boldsymbol{z}$ of the Financial Institution's matrix Z and therefore massively intrudes privacy. We can prevent this behavior by introducing Differential Privacy to the application. However, setting ϵ is not trivial since the noise that needs to be added is highly dependent on the size of the dataset and the desired amount of privacy. The goal is to maximize privacy and still retain a sufficient level of utility. We measure the rising level of privacy as ϵ increases by two indicators:

Noise difference on matrix entry scale: Provided that we do not add any noise to $\boldsymbol{h}$ and $\boldsymbol{h'}$, we learn the matrix row $\boldsymbol{z}$ directly by computing $\boldsymbol{z} \leftarrow \boldsymbol{h} - \boldsymbol{h'}$. Assume another protocol run where we add noise to both vectors. We obtain $\boldsymbol{h_{ns}} \leftarrow \boldsymbol{h} + \boldsymbol{n}$ and $\boldsymbol{h'_{ns}} \leftarrow \boldsymbol{h'} + \boldsymbol{n'}$, where $\boldsymbol{n}$ and $\boldsymbol{n'}$ are vectors holding the noise. By computing $\boldsymbol{h_{ns}} - \boldsymbol{h'_{ns}}$ we get $\boldsymbol{z_{ns}}$, which is the noisy version of the critical matrix row $\boldsymbol{z}$. Let

$$d_{vec} = \|\boldsymbol{z} - \boldsymbol{z_{ns}}\|$$

be the difference between the orginal version $\boldsymbol{z}$ of the critical matrix row and the noisy version $\boldsymbol{z_{ns}}$. Intuitively, d_{vec} is the *distance from the critical matrix row* $\boldsymbol{z}$ *to its noisy variant*. We can assume, that the higher this value is, the more noise was introduced and therefore more privacy is obtained. As a side effect, the implication

[16] https://www.chartjs.org/
[17] https://www.w3schools.com/html/html5_canvas.asp

on the final averaged output increases as d_{vec} grows. In other words, the final score gets more noisy and the utility decreases as d_{vec} increases. Therefore, we also need to measure the impact on the noise in the final averaged output.

Noise difference on overall score scale: As we introduce more noise to the output $\boldsymbol{h}$ of the protocol, the overall score is distorted. Luckily, we average all scores h_i, which allows us to add a substantial amount of noise without suffering from too much distortion in the averaged result. We compute the difference between the original averaged scores to the noisy averaged scores:

$$d_{out} = \|\boldsymbol{h} \cdot \frac{\mathbf{1}}{\mathbf{N}} - \boldsymbol{h_{ns}} \cdot \frac{\mathbf{1}}{\mathbf{N}}\|$$

Intuitively, d_{out} is the *distance from the noisy score to the original averaged score.* We want to keep d_{out} as low as possible.

The vectors $\boldsymbol{n}$ and $\boldsymbol{n'}$ hold the noise which is solely influenced by the privacy parameter ϵ. We conducted a series of experiments to evaluate which setting of ϵ fits best. However, one should keep in mind that our findings represent directives that must be re-evaluated in future applications since the results are the product of our specific heuristic approach. We used the two measures d_{vec} and d_{out} to evaluate each setting. We assumed a scoring scheme with $k = 5$ scores lying within the range $0, ..., 1000$. Since the optimal setting of ϵ is heavily dependent on N, we tested against datasets of N in $[10, 100, 1000]$. We chose ϵ from the range 0.1 to 7, depending on the dataset. Our resulting measures d_{vec} and d_{out} are computed by using the mean noise over 1000 iterations of the same dataset. The sensitivity Δq is set to 1000 in all scenarios. Figure 5.9 shows the measurement for the dataset with $N = 10$. In the case $\epsilon = 1$, we can see that, although the distance to the critical matrix row is distorted by 4120 on average, which makes it practically impossible to learn it, the distance to the original averaged scores is only 283. This is a great result since the distance is split on all $k = 5$ scores. However, we cannot neglect the fact that those are the distances computed with the mean noise. In reality, the protocol is executed only once, and chances are that the distance is much higher and unevenly distributed. To investigate this issue and to get a feeling about the impact of a particular distance, we randomly picked some $\boldsymbol{h_{ns}}$ and compared it to the original $\boldsymbol{h}$. That is, we compared the original score with randomly picked noisy variants of it to see how much deviation we have in practice. We listed this comparison in Table 5.1. The results are reasonable since $\boldsymbol{h_{ns}}$ approximates $\boldsymbol{h}$ quite well, but the utility could be increased by lowering the privacy, accomplished by choosing a higher ϵ. This is only an option if there is enough noise in $\boldsymbol{z_{ns}}$ so that it masks $\boldsymbol{z}$ properly. Table 5.2 shows arbitrarily picked $\boldsymbol{z_{ns}}$ for a specific $\boldsymbol{z}$. There is enough noise since it seems hard to infer $\boldsymbol{z}$ by looking at $\boldsymbol{z_{ns}}$. We conclude the optimal setting to be at $\epsilon = 3$, where d_{vec} is still 1389 (averaged), and d_{out} is only 95 (averaged). For the datasets with $N = 100$ and $N = 1000$, we can take even lower values for ϵ without suffering too much from a decreased utility. This stems from the fact that at $N = 100$ and $N = 1000$, there is a high enough number of entries in the set so that even a heavily distorted $\boldsymbol{z}$ does not contribute enough to

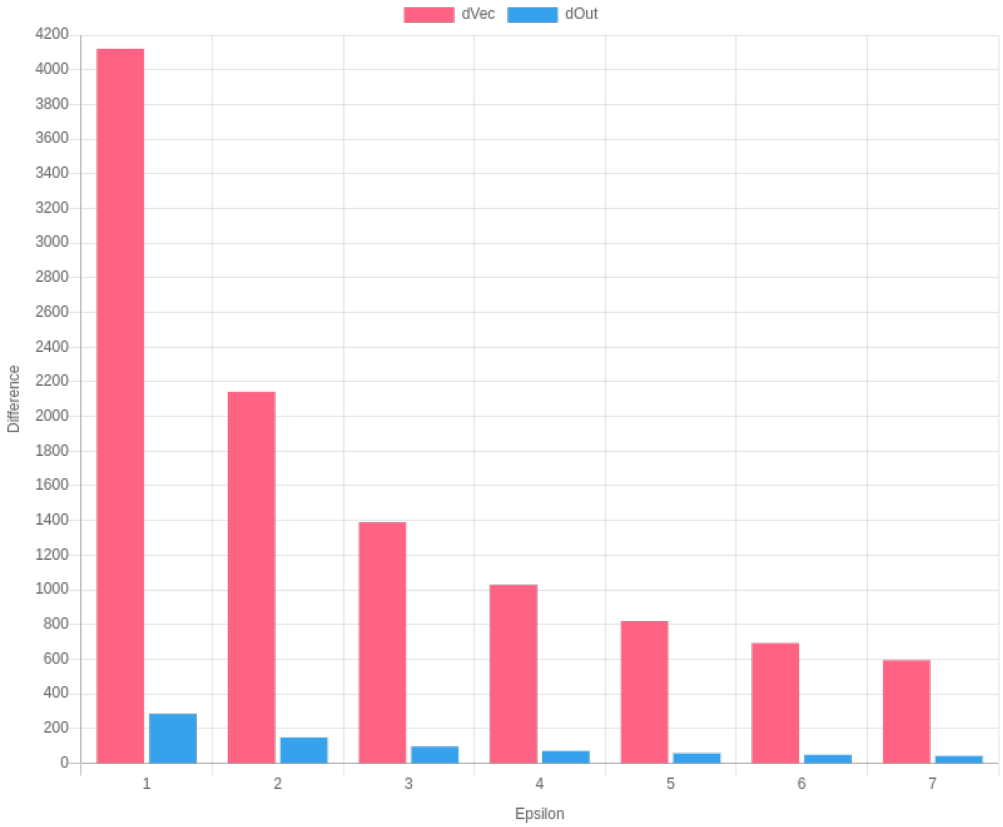


Figure 5.9: Bar chart showing d_{vec} and d_{out} for different settings of epsilon. The difference is the result of the mean noise over 1000 iterations of a dataset with 10 entries.

the final result to make a significant difference. In the case $N = 100$ and $\epsilon = 0.8$ we have a d_{vec} of 5240 (averaged) and d_{out} of only 35. That is, we have a higher d_{vec} then in a dataset with $N = 10$ with $\epsilon = 1$, but with a considerably lower d_{out}. We can see that even though the distortion is considerabely high, there is only a small loss in utility. Therefore, we recommend setting $\epsilon = 0.8$ in the dataset were $N = 100$. The utility gain gets clearer at $N = 1000$. Setting $\epsilon = 0.1$, gives us a d_{vec} of 40488 (averaged) and a d_{out} of 28 (averaged). There is no need to further decrease ϵ as N raises since further impacts on privacy are negligible at this point.

5.3.4.7 Caveats

The Risk Assessment implementation reuses almost all visual components and the logic behind them from Corona Heatmap. Therefore, the limitations and restriction are exactly the same. We want to to refer to section 5.2.4 for more informations about them.

$\boldsymbol{h}$	$\boldsymbol{h_{ns}}$
	$[216, 567, 538, 443, 263]$
	$[276, 761, 503, 581, 345]$
$[216, 586, 435, 389, 382]$	$[102, 630, 300, 386, 304]$
	$[247, 528, 372, 542, 405]$
	$[177, 470, 487, 103, 386]$

Table 5.1: The original scores versus some randomly picked noisy variants in a dataset with $N = 10$ entries. Noise was generated with $\epsilon = 1$.

$\boldsymbol{z}$	$\boldsymbol{z_{ns}}$
	$[988, 4176, -3358, 1642, 372]$
	$[-14, 2115, 2132, 3749, 2530]$
$[44, 750, 554, 991, 503]$	$[3132, 3963, 2969, 1791, 2085]$
	$[674, -1827, 3413, 943, 1326]$
	$[127, 1740, 898, -1404, 2014]$

Table 5.2: The original critical matrix row $\boldsymbol{z}$ versus arbitrarily picked noisy variants $\boldsymbol{z_{ns}}$ of it in a dataset with $N = 10$ entries. Noise was generated with $\epsilon = 1$.

5.4 Privacy-Preserving Questionnaires

Conducting questionnaires can potentially pose a privacy risk for attendees. In most cases, it is not clear how the data will be treated after conduction. The results might only be needed once to then be discarded. However, an attendant cannot be sure that his data will be deleted or treated with discretion. The optimal solution is to encrypt the answers and hand them over to the conductor, who, in return, will evaluate them in the encrypted domain. PSA is a system that allows us to do exactly that. However, it is not possible in every scenario since the Server is required to construct an evaluation procedure based on a matrix. If the questionnaire is needed as input to deliver an output according to some linear transformation of the input, then Private Selective Aggregation can help to secure the process in terms of data privacy.

5.4.1 Eurecat

During the PSA library's further development, we collaborate with Eurecat[18] to implement the library as a privacy-preserving measure in their data rating process. Eurecat offers a quality rating service where a customer can have a dataset rated to gain rough knowledge about its value. The process is split into two components:

- Qualitative Information Extracting and Data Scoring Sub-Component (QDSC)
- Automatic Data Analysis and Scoring Sub-Component

Our solution targets the QDSC and performs that part in a privacy-preserving way. The customer has to fill out a questionnaire about the data on a website, and Eurecat will subsequently use it to generate a rating. The questions are coming from three categories:

- Business Intelligence
- Domain-Specific Questions
- Data Science

Each category targets a different aspect of the dataset. Chances are very high that a customer wants to avoid giving information about the data out of hands unencrypted. It is the customer's interest to keep all answers secret, even from Eurecat, who are rating the customer's data in the first place. This can be achieved with Private Selective Aggregation.

5.4.2 Rating Process

Subject A, who could be an institution or an individual, wants to acquire a quality rating for a dataset. Eurecat does offer a *private rating process* and A would like to make use of it. A visits Eurecat's website and fills out the questionnaire that holds a couple of questions about the data. After checking some boxes and assigning values, A clicks the go-button and waits for a result. Meanwhile, the questionnaire is converted into numerical form by applying a function that maps the answers to an array of fixed numbers. After that, the array is locally encrypted in A's browser and sent to the Server running at Eurecat's end. The Server holds a scoring-table that maps every answer to a score. All scores are summed up and aggregated in a final score. Note that the scoring-table is matched with the answers homomorphically, that is, in the encrypted domain. The Server does not learn anything about the answers. The scoring-table is the matrix used to compute a linear transformation (assigning a score for each answer) on A's encrypted array. The result is still encrypted and sent back to A, where it gets decrypted by A and is ready to be further inspected. Eurecat does not learn anything, not even the final rating, although they are the ones who provide the rating in the first place. This procedure is depicted in Figure 5.10. It seems controversial to rate data without even

[18] https://eurecat.org/en/

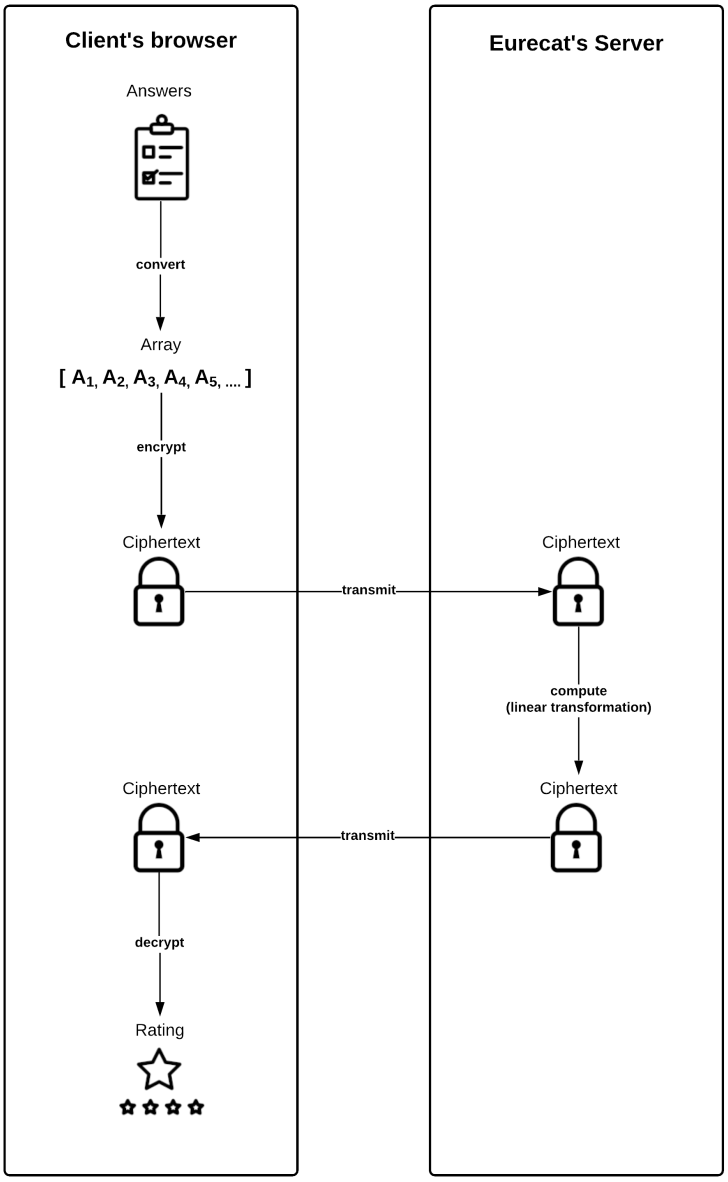


Figure 5.10: A high-level visualization of the private rating obtaining process at Eurecat.

knowing the rating in the end. This is the power of a system based on Homomorphic Encryption. Eurecat has a fixed matrix that is used to apply a linear transformation on A's answer vector. Not all of A's answers have the same impact on the quality of the data, therefore some answers add more value to the overall quality. Thus, the transformation could be seen as a way of applying weights to every answer and summing them up in the

end. This transformation entirely happens in the encrypted domain. Eurecat knows how the answers at A's end are formated and encrypted before transmission and relies on this form when processing the data at the server-end. The most crucial part of this whole system is to define the matrix such that it will produce a reasonable rating for a set of answers.

5.4.3 Environment Details

Since the customer has to fill a questionnaire and needs visual support for usability reasons, the PSA library on the Client's end runs in a browser. This way, the questionnaire can easily be presented to the customer and filled via the GUI. By the click of a button, the questionnaire is evaluated automatically without any interaction from Eurecat directly. Running the PSA server in a Node environment is, therefore, the best option. We refer to the browser-to-node model from section 4.2.5. In this model, data is directly transmitted between both parties with no need for a mediator.

5.5 Statistics

One exciting field in which the PSA protocol can be applied efficiently is in the area of statistics. On some occasions, the party which conducts a survey is obliged to keep the participant's participation private. This could, for example, be the case with statistics related to drugs, child abortion, or intercourse. New statistical data is retrieved out of two independent datasets without leaking information about one party's dataset to the other. This section serves as a basis for future work and was not implemented.

5.5.1 Generating Disease Profiles

Assuming that Statistik Austria[19] wants to conduct a survey about diseases correlated to smoking. For this process, Statistik Austria needs to know who is a heavy smoker and which diseases the smoker was facing in the past. Statistik Austria cannot conduct this survey without the help of a smoking cessation facility and the Austrian Healthcare Insurance (Österreichische Gesundheitskasse, ÖGK)[20] since those two parties own the required data. However, it would massively intrude privacy if they generated the statistic in the naive way. The solution is to let the Smoking Cessation Facility and ÖGK run the protocol and use the output for transmission to Statistik Austria.

5.5.2 Smoking Cessation Facility

A smoking cessation facility owns information about who is a heavy smoker. It acts as the Client in this application. Input to the protocol is the vector $\boldsymbol{x}$ of length N containing the smokers' identities. Since, we are assuming a preceding run of a PSI protocol, the

[19] http://www.statistik.at/web_de/statistiken/index.html
[20] https://www.gesundheitskasse.at/cdscontent/?contentid=10007.813892&portal=oegkportal

data indices match already. The vector $\boldsymbol{x}$ contains a one at indices corresponding to a smoker, and zero in the non-smoker case.

5.5.3 ÖGK

For this application, information about which diseases an individual faced in the past is required. The required information is saved in the electronic health records (Elektronische Gesundheitsakte, ELGA)[21], to which the ÖGK has access. The ÖGK acts as the Server in this scenario and provides a matrix in which the columns map a broad spectrum of diseases to a maximum of k. Entry $z_{i,j}$ holds a 1 if the person at index i suffered from disease j, else 0. The matrix will only consist of zeros and ones.

5.5.4 Disease Profile

Performing the protocol yields the vector *diseases* that holds the summed up amount a_i of how often a particular disease occurred. The Client now divides every a_i by the number of smokers, which is the same as the length of the vector $\boldsymbol{x}$. The result is the general disease profile of a smoker. It can then be sold to statistics institutions like Statistik Austria. High numbers indicate a causality between the specific disease and smoking.

[21] https://www.elga.gv.at/

Chapter 6

Conclusion

Our work showed that Private Selective Aggregation can be used to tackle various privacy-related problems in the industry. We pointed out some of these problems and studied how PSA can be applied as a generic framework to solve them. Along with our experiments, we argued about limitations and provided reasoning about proper parameter settings. By conducting a series of performance tests, we evaluated the viability of PSA in the web and concluded that it is indeed practical depending on the application's size. We developed a publicly accessible library to ease the integration of PSA and provided recommendations regarding its proper usage. In this section, we want to summarize and reflect on our key findings.

6.1 PSA as Generic Framework

We saw that PSA indeed solves some present issues in the industry. The focus in this work was on using PSA to link mobility with infectious diseases and issue risk assessments, but we also discussed other scenarios in less detail. From our experiments, it becomes clear that nationwide rollouts of Corona Heatmap's web version are possible, although it is better to use an implementation in a CPU-optimized language due to the performance boost. The web version gets more into focus as the application is restricted to smaller geographical areas. That is, the size of the underlying data is shrunk down. The same limitations occur when using PSA as a basis for private risk assessments. We can privately generate a Financial Standing Report of an investment subject to estimate the risk of investment. Within the report, we can pick an arbitrary amount of scores to refine the estimation. In practice, such an application's size will be smaller than in Corona Heatmap since the risk assessments do not target whole populations. The same is true for secure questionnaires where we are cooperating with Eurecat, who are integrating PSA already into their process.

6.2 PSA Library

To ease the integration of PSA in future applications, we built a library in JavaScript and provided the results of our performance benchmarks along with it. Our results show that an PSA application based on JavaScript performs roughly 6x slower than an equivalent in C++. This number is not tragic after all since it solely hints us towards the fact that

we have to separate our applications into CPU-intense and CPU-light applications. We recommend using a native implementation for heavy workloads, while for applications with a great need for easy adoption and smaller workloads, we recommend the web implementation. From a usability standpoint, PSA in the web is a great advancement since the setup is easy and usage straightforward. Using PSA in a browser allows easy adoption of the technology. This is only possible with the web version since a native implementation always comes with some kind of setup process. When it comes to increasing the application's size, the runtime scales linearly in the dimensions N and k of the Server's matrix. Additionally, we discussed various kinds of environment setups, showed how to set up a transport layer between Client and Server, and studied corresponding security implications.

6.3 Proper Parameter Settings

We thoroughly studied the right choice of parameters in our work. It is not trivial to find the right ϵ for an application based on Differential Privacy. We cannot give a general statement about the right setting in every case. However, for our applications, the right choice turned out to be somewhere between 0.1 and 3. For Corona Heatmap, we used a dataset with 10000 entries, which led to an ϵ of 1.5. We want to stress that our findings are solely based on the visual impact induced by adjusting ϵ. Therefore, if we would use a different way of dawing the heatmap, the results would probably change. In the end, we conclude that an adversary would probably learn more by using computational operations on the output vector rather than by just looking at the heatmap. Moreover, we evaluated the generation of risk assessments with different settings of ϵ and multiple datasets of different sizes (10, 100, and 1000 subjects). We defined two functions d_{vec} and d_{out} that measure the noise introduced into the assessments. Our results show that ϵ is highly dependent on the applications and the size of the dataset used. The optimal setting in our case was $\epsilon = 3$ for the data set with $N = 10$, $\epsilon = 0.8$ for $N = 100$, and $\epsilon = 0.1$ for $N = 1000$. However, our findings represent guidelines and not hard facts since they are based on our specific heuristic approach. Future applications must re-evaluate them carefully. The results showed that due to the fact that the risk assessments are generated in terms of the average, ϵ can be decreased as the size of the dataset increases. The contribution of the noise in the final result slowly diminishes in large datasets, such that we keep enough utility at a high privacy level.

6.4 Future Work

There is still a long way to maximize the potential of PSA. From a theoretical standpoint, the system would benefit the most if there were a trivial way to determine the privacy parameter ϵ. Currently, the parameter has to be found by trying different settings and evaluate the resulting output. This process is very time-consuming and would benefit considerably from an improvement. From a practical standpoint, we are very optimistic that there are enough tools out there to start building PSA applications in the web after

this work. Nevertheless, the protocol would benefit from an implementation of a generic library in a native language that can be used for arbitrary PSA application construction, similar to our library. As our results show, the JavaScript implementation becomes less viable when the underlying data's size increases. There is no limit per se, but at some point, it is better to power the protocol with a native implementation for the sake of performance. The is still much room for improvements in our library. Future work could address the transport layer problem. It would be a nice feature if the library already provided a way to manage transmissions across environments. In our opinion, the best option would be to integrate a peer-to-peer connection layer.

Acronyms

API	Application Programming Interface	34, 45
BFV	Brakerski/Fan-Vercauteren	9, 28
CSV	Comma-Separated Values	40, 41, 52, 53
DP	Differential Privacy	62
ERP	Enterprise Ressource Planning	48
FHE	Fully Homomorphic Encryption	8, 9
FI	Financial Institution	46, 47
GDPR	General Data Protection Regulation	2, 37
GUI	Graphical User Interface	33, 40, 51, 59
HE	Homomorphic Encryption	2, 6, 19, 22
IBAN	International Bank Account Number	49, 52, 53
IDC	International Data Corporation	2
IND-CPA	Indistinguishability Under Chosen-Plaintext Attack	6, 17, 18
JSON	JavaScript Object Notation	28
MC	Mobile Carrier	37, 40, 41, 44
MH	Ministry of Health	37, 44
NPM	Node Package Manager	21, 25, 39, 51
OT	Oblivious Transfer	iv, 1, 17

PHE	Partially Homomorphic Encryption	8
PIR	Private Information Retrieval	iv, 1
PoC	Proof of Concept	iv, 3
PSA	Private Selective Aggregation	iv, ix, x, 2, 3, 4, 11, 12, 13, 15, 16, 17, 18, 19, 21, 22, 25, 26, 28, 31, 35, 36, 37, 38, 39, 41, 43, 45, 46, 47, 56, 57, 61, 62, 63
PSI	Private Set Intersection	11, 36, 37, 59
SaaS	Software as a Service	2, 48
SWHE	Somewhat Homomorphic Encryption	8, 9
URL	Uniform Resource Locator	46
WHO	World Health Organization	36

Bibliography

[1] Carlos Aguilar-Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. "XPIR : Private Information Retrieval for Everyone". In: *Proceedings on Privacy Enhancing Technologies* 2016 (Aug. 2015). DOI: 10.1515/popets-2016-0010.

[2] Alexandros Bampoulidis, Alessandro Bruni, Lukas Helminger, Daniel Kales, Christian Rechberger, and Roman Walch. "Privately Connecting Mobility to Infectious Diseases via Applied Cryptography". In: *CoRR* abs/2005.02061 (2020).

[3] Dušan Baran, Andrej Pastýr, and Daniela Baranová. "Financial Analysis of a Selected Company". In: *Research Papers Faculty of Materials Science and Technology Slovak University of Technology* 24 (June 2016). DOI: 10.1515/rput-2016-0008.

[4] Alex Berke, M. Bakker, Praneeth Vepakomma, Kent Larson, and Alex 'Sandy' Pentland. "Assessing Disease Exposure Risk with Location Data: A Proposal for Cryptographic Preservation of Privacy". In: *arXiv: Cryptography and Security* (2020).

[5] Raghav Bhaskar, Srivatsan Laxman, Adam Smith, and Abhradeep Thakurta. "Discovering Frequent Patterns in Sensitive Data". In: *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '10. Washington, DC, USA: Association for Computing Machinery, 2010, pp. 503–512. ISBN: 9781450300551. DOI: 10.1145/1835804.1835869. URL: https://doi.org/10.1145/1835804.1835869.

[6] G. R. Blakley and C. Meadows. "A Database Encryption Scheme Which Allows the Computation of Statistics Using Encrypted Data". In: *1985 IEEE Symposium on Security and Privacy* (1985), pp. 116–116.

[7] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. "Evaluating 2-DNF Formulas on Ciphertexts". In: *Theory of Cryptography*. Ed. by Joe Kilian. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 325–341. ISBN: 978-3-540-30576-7.

[8] Joppe W. Bos, Kristin Lauter, Jake Loftus, and Michael Naehrig. *Improved Security for a Ring-Based Fully Homomorphic Encryption Scheme*. Cryptology ePrint Archive, Report 2013/075. https://eprint.iacr.org/2013/075. 2013.

[9] Samuel Brack, Leonie Reichert, and Björn Scheuermann. *Decentralized Contact Tracing Using a DHT and Blind Signatures*. Cryptology ePrint Archive, Report 2020/398. https://eprint.iacr.org/2020/398. 2020.

[10] Z. Brakerski and V. Vaikuntanathan. "Efficient Fully Homomorphic Encryption from (Standard) LWE". In: *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*. 2011, pp. 97–106. DOI: 10.1109/FOCS.2011.12.

[11] Zvika Brakerski. *Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP*. Cryptology ePrint Archive, Report 2012/078. `https://eprint.iacr.org/2012/078`. 2012.

[12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. *Fully Homomorphic Encryption without Bootstrapping*. Cryptology ePrint Archive, Report 2011/277. `https://eprint.iacr.org/2011/277`. 2011.

[13] Zvika Brakerski and Vinod Vaikuntanathan. "Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages". In: *Proceedings of the 31st Annual Conference on Advances in Cryptology*. CRYPTO'11. Santa Barbara, CA: Springer-Verlag, 2011, pp. 505–524. ISBN: 9783642227912.

[14] Zvika Brakerski and Vinod Vaikuntanathan. "Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages". In: vol. 6841. Aug. 2011, pp. 505–524. DOI: `10.1007/978-3-642-22792-9_29`.

[15] Gilles Brassard, Claude Crepeau, and Jean-Marc Robert. "All-or-Nothing Disclosure of Secrets". In: *Advances in Cryptology — CRYPTO' 86*. Ed. by Andrew M. Odlyzko. Berlin, Heidelberg: Springer Berlin Heidelberg, 1987, pp. 234–238. ISBN: 978-3-540-47721-1.

[16] Alessandro Bruni, Lukas Helminger, Daniel Kales, Christian Rechberger, and Roman Walch. *Privately Connecting Mobility to Infectious Diseases via Applied Cryptography*. Cryptology ePrint Archive, Report 2020/522. `https://eprint.iacr.org/2020/522`. 2020.

[17] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. *Bulletproofs: Short Proofs for Confidential Transactions and More*. Cryptology ePrint Archive, Report 2017/1066. `https://eprint.iacr.org/2017/1066`. 2017.

[18] Christian Cachin, Silvio Micali, and Markus Stadler. "Computationally Private Information Retrieval with Polylogarithmic Communication". In: *Advances in Cryptology — EUROCRYPT '99*. Ed. by Jacques Stern. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 402–414. ISBN: 978-3-540-48910-8.

[19] Ran Canetti. *Security and Composition of Multi-party Cryptographic Protocols*. Cryptology ePrint Archive, Report 1998/018. `https://eprint.iacr.org/1998/018`. 1998.

[20] Ran Canetti, Ari Trachtenberg, and Mayank Varia. *Anonymous Collocation Discovery: Harnessing Privacy to Tame the Coronavirus*. 2020. arXiv: `2003.13670 [cs.CY]`.

[21] Justin Chan, D. Foster, Shyam Gollakota, E. Horvitz, J. Jaeger, Sham M. Kakade, T. Kohno, J. Langford, J. Larson, S. Singanamalla, J. Sunshine, and S. Tessaro. "PACT: Privacy-Sensitive Protocols And Mechanisms for Mobile Contact Tracing". In: *IEEE Data Eng. Bull.* 43 (2020), pp. 15–35.

[22] Kamalika Chaudhuri and Claire Monteleoni. "Privacy-Preserving Logistic Regression". In: *Proceedings of the 21st International Conference on Neural Information Processing Systems*. NIPS'08. Vancouver, British Columbia, Canada: Curran Associates Inc., 2008, pp. 289–296. ISBN: 9781605609492.

[23] Hao Chen, Kim Laine, and Peter Rindal. *Fast Private Set Intersection from Homomorphic Encryption*. Cryptology ePrint Archive, Report 2017/299. https://eprint.iacr.org/2017/299. 2017.

[24] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. *Homomorphic Encryption for Arithmetic of Approximate Numbers*. Cryptology ePrint Archive, Report 2016/421. https://eprint.iacr.org/2016/421. 2016.

[25] Nimesh Chhetri. "A Comparative Analysis of Node. js (Server-Side JavaScript)". In: (2016).

[26] Eunjoon Cho, Seth A. Myers, and Jure Leskovec. "Friendship and Mobility: User Movement in Location-Based Social Networks". In: *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '11. San Diego, California, USA: Association for Computing Machinery, 2011, pp. 1082–1090. ISBN: 9781450308137. DOI: 10.1145/2020408.2020579. URL: https://doi.org/10.1145/2020408.2020579.

[27] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. "Private Information Retrieval". In: *J. ACM* 45.6 (Nov. 1998), pp. 965–981. ISSN: 0004-5411. DOI: 10.1145/293347.293350.

[28] Aaqib Bashir Dar, Auqib Hamid Lone, Saniya Zahoor, Afshan Amin Khan, and Roohie Naaz. *Applicability of Mobile Contact Tracing in Fighting Pandemic (COVID-19): Issues, Challenges and Solutions*. Cryptology ePrint Archive, Report 2020/484. https://eprint.iacr.org/2020/484. 2020.

[29] Reinsel David, Gantz John, and Rydning John. *Data Age 2025: The Digitization of the World*. IDC White Paper. https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf. Nov. 2018. URL: https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf%20110.5%20126.7%20143.7https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf.

[30] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. *Fully Homomorphic Encryption over the Integers*. Cryptology ePrint Archive, Report 2009/616. https://eprint.iacr.org/2009/616. 2009.

[31] Changyu Dong and Liqun Chen. "A Fast Single Server Private Information Retrieval Protocol with Low Communication Cost". In: *Computer Security - ESORICS 2014*. Ed. by Mirosław Kutyłowski and Jaideep Vaidya. Cham: Springer International Publishing, 2014, pp. 380–399. ISBN: 978-3-319-11203-9.

[32] Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. "CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy". In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. ICML'16. New York, NY, USA: JMLR.org, 2016, pp. 201–210.

[33] Cynthia Dwork. "Differential Privacy". In: *Automata, Languages and Programming*. Ed. by Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 1–12. ISBN: 978-3-540-35908-1.

[34] Cynthia Dwork and Aaron Roth. "The Algorithmic Foundations of Differential Privacy". In: *Foundations and Trends® in Theoretical Computer Science* 9.3–4 (2014), pp. 211–407. ISSN: 1551-305X. DOI: 10.1561/0400000042. URL: http://dx.doi.org/10.1561/0400000042.

[35] Taher ElGamal. "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms". In: *Advances in Cryptology*. Ed. by George Robert Blakley and David Chaum. Berlin, Heidelberg: Springer Berlin Heidelberg, 1985, pp. 10–18. ISBN: 978-3-540-39568-3.

[36] Shimon Even, Oded Goldreich, and Abraham Lempel. "A Randomized Protocol for Signing Contracts". In: *Commun. ACM* 28.6 (June 1985), pp. 637–647. ISSN: 0001-0782. DOI: 10.1145/3812.3818. URL: https://doi.org/10.1145/3812.3818.

[37] Junfeng Fan and Frederik Vercauteren. *Somewhat Practical Fully Homomorphic Encryption*. Cryptology ePrint Archive, Report 2012/144. https://eprint.iacr.org/2012/144. 2012.

[38] J. Feigenbaum. "Encrypting Problem Instances: Or ..., Can You Take Advantage of Someone Without Having to Trust Him?" In: *CRYPTO*. 1985.

[39] Gartner Inc. *Gartner Forecasts Worldwide Public Cloud Revenue to Grow 17.5 Percent in 2019*. https://www.gartner.com/en/newsroom/press-releases/2019-04-02-gartner-forecasts-worldwide-public-cloud-revenue-to-g. Apr. 2019. URL: http://www.telespazio.it/docs/brodoc/GCC_eng.pdf.

[40] Craig Gentry. "A fully homomorphic encryption scheme". crypto.stanford.edu/craig. PhD thesis. Stanford University, 2009.

[41] Craig Gentry. "Fully Homomorphic Encryption Using Ideal Lattices". In: *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*. STOC '09. Bethesda, MD, USA: Association for Computing Machinery, 2009, pp. 169–178. ISBN: 9781605585062. DOI: 10.1145/1536414.1536440. URL: https://doi.org/10.1145/1536414.1536440.

[42] Craig Gentry, Amit Sahai, and Brent Waters. *Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based*. Cryptology ePrint Archive, Report 2013/340. https://eprint.iacr.org/2013/340. 2013.

[43] O. Goldreich, S. Micali, and A. Wigderson. "How to Play ANY Mental Game". In: *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*. STOC '87. New York, New York, USA: Association for Computing Machinery, 1987, pp. 218–229. ISBN: 0897912217. DOI: 10.1145/28395.28420. URL: https://doi.org/10.1145/28395.28420.

[44] Shai Halevi and Victor Shoup. "Algorithms in HElib". In: *Advances in Cryptology – CRYPTO 2014*. Ed. by Juan A. Garay and Rosario Gennaro. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 554–571. ISBN: 978-3-662-44371-2.

[45] Shai Halevi and Victor Shoup. "Bootstrapping for HElib". In: *Advances in Cryptology – EUROCRYPT 2015*. Ed. by Elisabeth Oswald and Marc Fischlin. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 641–670. ISBN: 978-3-662-46800-5.

[46] Shai Halevi and Victor Shoup. *Faster Homomorphic Linear Transformations in HElib*. Cryptology ePrint Archive, Report 2018/244. https://eprint.iacr.org/2018/244. 2018.

[47] Carmit Hazay and Yehuda Lindell. *Efficient Protocols for Set Intersection and Pattern Matching with Security Against Malicious and Covert Adversaries*. Cryptology ePrint Archive, Report 2009/045. https://eprint.iacr.org/2009/045. 2009.

[48] *How Target Figured Out A Teen Girl Was Pregnant Before Her Father Did*. https://www.forbes.com/sites/kashmirhill/2012/02/16/how-target-figured-out-a-teen-girl-was-pregnant-before-her-father-did/?sh=7b8d67ad6668. Accessed: 2020-12-11.

[49] J. Hsu, M. Gaboardi, A. Haeberlen, S. Khanna, A. Narayan, B. C. Pierce, and A. Roth. "Differential Privacy: An Economic Method for Choosing Epsilon". In: *2014 IEEE 27th Computer Security Foundations Symposium*. 2014, pp. 398–410. DOI: 10.1109/CSF.2014.35.

[50] A. Kiayias, N. Leonardos, H. Lipmaa, K. Pavlyk, and Q. Tang. "Optimal Rate Private Information Retrieval from Homomorphic Encryption". In: *Proceedings on Privacy Enhancing Technologies* 2015 (2015), pp. 222–243.

[51] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. *Efficient Batched Oblivious PRF with Applications to Private Set Intersection*. Cryptology ePrint Archive, Report 2016/799. https://eprint.iacr.org/2016/799. 2016.

[52] A. Korolova, K. Kenthapadi, N. Mishra, and A. Ntoulas. "Releasing search queries and clicks privately". In: *WWW '09*. 2009.

[53] Adriana Lopez-Alt, Eran Tromer, and Vinod Vaikuntanathan. *On-the-Fly Multiparty Computation on the Cloud via Multikey Fully Homomorphic Encryption*. Cryptology ePrint Archive, Report 2013/094. https://eprint.iacr.org/2013/094. 2013.

[54] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. "On Ideal Lattices and Learning with Errors over Rings". In: *Advances in Cryptology – EUROCRYPT 2010*. Ed. by Henri Gilbert. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 1–23. ISBN: 978-3-642-13190-5.

[55] Ashwin Machanavajjhala, Aleksandra Korolova, and Atish Das Sarma. *Personalized Social Recommendations - Accurate or Private?* 2011. arXiv: 1105.4254 [cs.DB].

[56] Frank McSherry and Ratul Mahajan. "Differentially-Private Network Trace Analysis". In: *SIGCOMM Comput. Commun. Rev.* 40.4 (Aug. 2010), pp. 123–134. ISSN: 0146-4833. DOI: 10.1145/1851275.1851199. URL: https://doi.org/10.1145/1851275.1851199.

[57] Moni Naor and Benny Pinkas. "Efficient Oblivious Transfer Protocols". In: *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '01. Washington, D.C., USA: Society for Industrial and Applied Mathematics, 2001, pp. 448–457. ISBN: 0898714907.

[58] Pascal Paillier. "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes". In: *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques*. Vol. 1592. Lecture Notes in Computer Science. Springer, 1999, pp. 223–238. DOI: 10.1007/3-540-48910-X_16.

[59] Benny Pinkas, Thomas Schneider, and Michael Zohner. *Scalable Private Set Intersection Based on OT Extension*. Cryptology ePrint Archive, Report 2016/930. https://eprint.iacr.org/2016/930. 2016.

[60] Michael O. Rabin. *How To Exchange Secrets with Oblivious Transfer*. Harvard University Technical Report 81 talr@watson.ibm.com 12955 received 21 Jun 2005. 2005. URL: http://eprint.iacr.org/2005/187.

[61] Oded Regev. "On Lattices, Learning with Errors, Random Linear Codes, and Cryptography". In: *J. ACM* 56.6 (Sept. 2009). ISSN: 0004-5411. DOI: 10.1145/1568318.1568324. URL: https://doi.org/10.1145/1568318.1568324.

[62] R L Rivest, L Adleman, and M L Dertouzos. "On Data Banks and Privacy Homomorphisms". In: *Foundations of Secure Computation, Academia Press* (1978), pp. 169–179.

[63] R. L. Rivest, A. Shamir, and L. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". In: *Commun. ACM* 21.2 (Feb. 1978), pp. 120–126. ISSN: 0001-0782. DOI: 10.1145/359340.359342. URL: https://doi.org/10.1145/359340.359342.

[64] Ronald L. Rivest, Len Adleman, and Michael L. Dertouzos. *On data banks and privacy homomorphisms*. 1978.

[65] *Microsoft SEAL (release 3.6)*. https://github.com/Microsoft/SEAL. Microsoft Research, Redmond, WA. Nov. 2020.

[66] Hezbullah Shah and Tariq Soomro. "Node.js Challenges in Implementation". In: *Global Journal of Computer Science and Technology* 17 (May 2017), pp. 72–83.

[67] N.P. Smart and F. Vercauteren. *Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes*. Cryptology ePrint Archive, Report 2009/571. `https://eprint.iacr.org/2009/571`. 2009.

[68] Martin Vojtek and Evzen Kocenda. "Credit scoring methods". In: *Finance a Uver - Czech Journal of Economics and Finance* 56 (Jan. 2006), pp. 152–167.

[69] Emelie Widegren. "Fully Homomorphic Encryption: A Case Study". In: 2018.

[70] Fengli Xu, Zhen Tu, Yong Li, Pengyu Zhang, Xiaoming Fu, and Depeng Jin. "Trajectory Recovery From Ash". In: *Proceedings of the 26th International Conference on World Wide Web* (Apr. 2017). DOI: `10.1145/3038912.3052620`. URL: `http://dx.doi.org/10.1145/3038912.3052620`.