Master Thesis
Computer Science

Radboud University SAP Netherlands

Secure Computation on the SAP Cloud Platform

Author:
Wouter de Boer

Internal supervisors:
Prof. dr. ir. Joan Daemen
Marloes Venema MSc.

External supervisors:
Ronald Schippers
Marcel Smits

Second reader:
dr. Asli Bay

November 11, 2020

Abstract

In the shared responsibility model of the cloud, all parties should be equipped with enough tools to protect data actively. While users might be partially responsible, they often do not have full control over their data. Data breaches in the cloud are amongst the highest ranked threats and arguably can do the most damage. Encryption techniques can ensure data confidentiality, but only for data at rest and data in transit. If we want to protect the data during processing, as a user, we need more advanced techniques. In this thesis, we will look into how secure computation can help protect data in the cloud and more specifically what homomorphic encryption can achieve and how it impacts performance and security.

Contents

Chapter 1

Introduction

In the last decade the transition into the cloud has been fast forwarded after the tech giants Amazon, Google and Microsoft emerged as cloud giants. Millions of users, companies and individuals alike, make use of the computing power and storage provided through the internet. Massive data centres around the globe provide services ranging from web-based email to entire IT infrastructures. In order to offer such a variety of services at a competitive price, a massive scale of operation is required.

Not only the massive scale but also the optimal usage of the cloud infrastructure helps in reducing cost. It follows the philosophy that a large group of users would more efficiently use the computing power than a single user can, thus increasing throughput and reducing idle-time. This also means that users can scale their resources effectively, and pay only for what they use.

Companies such as Netflix can benefit from this so called *unlimited* scaling through a modular and flexible architecture. They can upscale to several thousands of servers on a rainy day and downscale when it is nice and sunny outside. While Netflix might have a lot of users during the evenings, Microsoft Teams sees a surge in the amount of meeting minutes during the day. At the end of March, an increase from 900 million minutes to almost 3 billion meeting minutes was measured. Since the processing and hosting of a meeting happens in the cloud, they were able to scale and handle a record of 4.1 billion meeting minutes in a single day [1].

The core business in the case of Netflix is to provide the customers with series and movies. This requires storage of movies and video encoding before sending it to customers. For Microsoft Teams, the core business is to create a virtual meeting space. This requires storage and hosting of potentially confidential chat messages or voice calls. Although these are just two examples of micro-service applications operating in the cloud, we can already see their alternating usage, the value of scalability and more importantly a difference in the level of trust.

In the near future, cloud-dependent applications and platforms will, maybe unwillingly, become more and more important in our personal and professional lives. The cloud giants, also called hyperscalers, will thus have to gain more and more trust in their security infrastructure. Even though a lot of money and expertise is invested into the security of the cloud, incidents and mistakes still happen on both cloud and user side. These incidents are not allowed to happen once highly regulated industries such as telecom, electric power or healthcare join the cloud landscape. Data breaches will be more severe since these industries are often associated with highly sensitive data, e.g., financial and health records. This will also make them a more valuable target for attackers.

Therefore, it is important to reflect on the amount of trust we put in the cloud providers. Not only for highly regulated industries, but also in general. We should also look into technical advancements, such as secure computation, which might enable and support developers and users of cloud applications to process data securely without losing the advantages of the cloud.

Secure computation is a subfield of cryptography that focuses on privacy-preserving computation. This implies that the input data, output data and function used in the computation remain private. This would in theory allow cloud providers to compute on data without knowing its content. To make this concept less abstract, we can again take Netflix as an example. In addition to providing movies, they also want to provide you with the best selection of movies. In order to achieve this they can use a recommendation system, e.g., through a technique called collaborative filtering or content-based filtering [2]. It uses movie ratings from users as an input and provides recommendations as output. Implementing this with secure computation will aim to protect user preferences and everything that can be derived from it, e.g., an interest in movies from a certain franchise can indicate that this person is also interested in buying its merchandise. This information can be valuable to marketeers and advertisers.

The SAP Cloud Platform supports developers in the creation and deployment of secure full-stack applications through integrated security features and services, e.g., authentication for web applications. In the future, this could perhaps be extended with a secure computation service. The service could allow developers to integrate secure computation easily into their applications and actively secure user data from the design phase.

In this thesis, we will look into secure computation with a focus on its feasibility in cloud applications and infrastructures. To correctly assess the current cloud landscape, we discuss the concept of the cloud, the benefits and the obstacles in terms of architecture and security and the main threats to

the cloud. We explain the main three techniques used in secure computation of which homomorphic encryption is highlighted for further research. We describe the recent advancements in the field of homomorphic encryption and their performance and security guarantees. In the last chapters we describe a general framework and scenario, derived from several promising usecases. To demonstrate what an implementation of such a general usecase could look like, we created a proof of concept on the SAP Cloud Platform using the SAP full-stack Web IDE and the Microsoft SEAL library.

Chapter 2

Preliminaries

This chapter will describe some prior knowledge necessary to clarify this thesis.

2.1 Terminology

- **Encode**, an operation that transforms human readable data into data that is ready for encryption.
- **Decode**, an operation that transforms encoded data back into human readable data.
- **Encrypt**, an operation that transforms encoded data into encrypted data.
- **Decrypt**, an operation that transforms encrypted data back into encoded data.
- **Message**, contains human readable data.
- **Plaintext**, contains encoded data.
- **Ciphertext**, contains encrypted data.
- **Packing/Batching**, an operation that packs multiple messages into a single plaintext.
- **Encryption scheme**, refers to the specific plaintext, ciphertext and key space, and the encryption and decryption algorithms. Homomorphic schemes also include an evaluation algorithm.

Encoded data does not provide any confidentiality and can be decoded by everybody, encrypted data can not be decrypted by everybody.

2.2 Mathematical structures

Definition 2.2.1. (Group) A group is a set with the following properties:

- It includes an operation on its elements, e.g., + or ×;
- The group is closed under this operation;
- Each element has an inverse;
- The group should include an identity element;
- The associative property should hold.

An abelian or commutative group also includes the commutative property.

Definition 2.2.2. (Rings) A ring is a set with two operations + and ×. It should be closed under both operations. The elements should form an abelian group under addition. For multiplication the ring only has the associativity property.

Definition 2.2.3. (Homomorphism) A homomorphism is a structure-preserving map between two algebraic structures, e.g., between two groups or two rings. For this mapping (f), the following should hold:

$$f(x) \cdot f(y) = f(x * y)$$

Where $\cdot$ is the operation from the first structure and $*$ of the second.

For this thesis we are focused on the ring structure of the plaintext and preserving its structure in the ciphertext, e.g., the addition of ciphertexts should result in the addition of plaintexts.

2.3 Encryption

When encrypting data, you need a key. The two ways of using this key is either asymmetric or symmetric. In symmetric cryptography we use the same key for encryption and decryption. In asymmetric cryptography we use two different keys. A secret (private) key and a public key. The main reason for asymmetric cryptography is to solve the issues regarding key management and establishing shared keys. For every pair of people communicating we need a shared symmetric key. However, in asymmetric crypto we only need a public and secret key per user. To establish a shared symmetric key, a hybrid approach is often used. This uses asymmetric cryptography to establish a symmetric shared key. This symmetric key can then be used to encrypt large amounts of data efficiently. One of the most famous symmetric encryption schemes is Rijndael, which became the Advanced Encryption

Standard. This standard is now used in almost all cloud setups to protect data at rest.

In asymmetric cryptography the public key should be authenticated and publicly available, this allows you to encrypt a plaintext for a specific user. The secret key belongs only to the user and can be used to decrypt the ciphertext. All encryption schemes are based on hard computational problems and trapdoor functions that are only possible to revert with the secret key.

A computational hardness assumption refers to the hypothesis that there is no known algorithm that can solve a particular problem efficiently. This comes from the computational complexity theory but is particularly useful in cryptography to indicate whether or not a scheme is secure.

The most famous public encryption scheme is RSA, introduced by Rivest, Shamir and Adleman. The assumption used in RSA is based on the factoring problem. It describes the challenge to factor a large number into its prime components.

Definition 2.3.1. (Textbook RSA) [3]

- $KeyGen(1^\lambda)$, choose e such that $gcd(e, \phi(N)) = 1$ and find d such that $e \cdot d \equiv 1 \bmod \phi(N)$. Where $N = p \cdot q$ and $\phi(N) = (p-1) \cdot (q-1)$. Output $pk = (N, e)$, $sk = d$.
- $Encrypt_{pk}(m)$, Output $c = m^e \mod N$.
- $Decrypt_{sk}(c)$, Output $m = c^d \mod N$.

The security of ElGamal is based on the decisional Diffie–Hellman assumption with respect to the underlying group. It describes the challenge of finding the discrete logarithm of a certain element in a cyclic group, e.g., Z_p^* with p a large prime. In a cyclic group all elements are generated from a single element called the generator.

Definition 2.3.2. (ElGamal) [4]

- $KeyGen(1^\lambda)$, Generate a cyclic group G with order q and generator g.
  Pick a random value x, between 1 and $q-1$.
  Compute $h = g^x$.
  Output $pk = (G, q, g, h)$, $sk = x$.
- $Encrypt_{pk}(m)$, Choose a random value r. Compute h^r and g^r. Output $c = (g^r, m \cdot h^r)$.
- $Decrypt_{sk}(c)$, compute $s = (g^r)^x$ and its inverse s^{-1}. Output $m = (m \cdot h^r) \cdot s^{-1}$.

2.4 Lattice-based cryptography

With the introduction of Shor's algorithm [5], the previous hard problems are not hard anymore for quantum computers. Since then several new public key encryption schemes were proposed that are quantum resistant. This sparked a new area of cryptography called Post-Quantum cryptography. In the search for a quantum secure encryption scheme, lattice-based cryptography has been one of the contenders from the start. As the name suggests it is based on lattices.

A lattice , as seen in Figures 2.1 and 2.2, is an abstract structure of points in a space. In this case we have a two dimensional lattice. It can be viewed as a regular tiling using squares, rectangles or even parallelograms. A lattice is defined by its basis. A basis is an ordered set of vectors that generate a lattice by scaling these vectors. A short basis such as $b_1 b_2$ consists of short vectors. A long basis $b_3 b_4$ consists of relatively long vectors. It is easier to see that $b_1 b_2$ generates the lattice than $b_3 b_4$. Some problems are only hard given that we provide a long basis, e.g., the shortest vector problem (SVP). These problems become even harder if we increase the dimension from two to three. Three dimensions can be viewed as a cube with lattice points inside it. The basis now consists of three vectors. If we scale this to n-dimensions we can really comprehend the complexity of certain hard problems on lattices even though we can not properly visualise it.

2.4.1 Hard lattice problems

Lattices are well studied and can provide us with several hard problems. In general, there are two variants of these problems, a search and decision problem. The search problems asks to search and give a specific solution. The decision problem asks to decide between different solutions or decide whether something is true or not.

Shortest vector problem

The shortest vector problem challenges us with finding the shortest non-zero vector in a lattice, with respect to the origin point [6].

- Search SVP, given a lattice basis B, find $v \in L(B)$ such that $||v|| = \lambda_1(L(B))$ (minimum distance).
- Decisional SVP, given a lattice basis B and a rational $r \in \mathbb{Q}$, determine whether $||v|| = \lambda_1(L(B)) \leq r$ or not.

For a two dimensional lattice, we can visualize this as seen in Figure 2.3.

Closest vector problem

The CVP problem [7] is closely related to the SVP problem. Given a lattice and a target point. Find the lattice-point closest to the target. This differs from the SVP problem, which has the origin as target point. For a two dimensional lattice, we can visualize this as seen in Figure 2.4.

Bounded distance decoding

Bounded Distance Decoding [7] is a variant of the CVP problem in which the target is guaranteed to be close to the lattice, relative to the minimum distance $\lambda_1(L)$ of the lattice.

Learning with errors

The main hard problem we are interested in for this paper is the Learning with Errors problem. It describes the hard problem of determining a secret given a sequence of random linear equations [8]. An LWE instance looks as follows:

- Choose a size parameter n, a modulus q and an error probability distribution $\mathcal{X}$ on $\mathbb{Z}_q$;
- Let $\mathcal{A}_{s,\mathcal{X}}$ be the probability distribution with samples $(a, < a, s > +e)$, where a is a vector $\in \mathbb{Z}_q^n$, e $\in \mathbb{Z}_q$ from $\mathcal{X}$ and fixed $s \in \mathbb{Z}_q^n$;
- An algorithm solves LWE if for an arbitrary amount of samples from $\mathcal{A}_{s,\mathcal{X}}$ it can output s with high probability.

In this thesis we will give a high level interpretation of the basic BFV scheme, it is based on the ring-LWE hard problem. This means the previous samples will thus be from a ring, e.g., $\mathbb{Z}[x]/(X^n + 1)$.

2.5 Miscellaneous

Next we will describe some additional concepts [9].

Negligible functions

Since an attacker can always randomly guess correctly (or gain another small advantage) there is a small chance that a scheme will be broken. This can be modeled by allowing a small probability of adversarial success, also called a negligible probability. In other words, the advantage that an adversary has is negligible.

Indistinguishability

In the area of computational complexity and cryptography, two distributions are computationally indistinguishable if no efficient algorithm can tell the difference between them except with negligible probability. When two encryptions are indistinguishable we speak of semantic security.

Chinese remainder theorem

The chinese remainder theorem is used to solve a linear system of congruences. This can be used to speedup calculations by performing the calculation in steps and then combine them later through this theorem to find a unique solution.

Application programming interfaces

Facilitates the interaction between, e.g., the frontend and backend of a system. It is used to provide a unified way to make requests and calls between several software components.

Privilege escalation

A type of threat that allows an adversary that has access to a user account, to increase his privileges. This means that the adversary can elevate the amount control it has over certain systems or files.

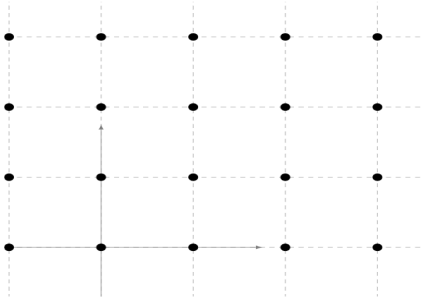


Figure 2.1: A Lattice

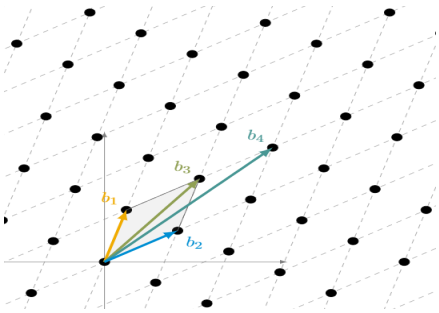


Figure 2.2: Good basis vs Bad basis

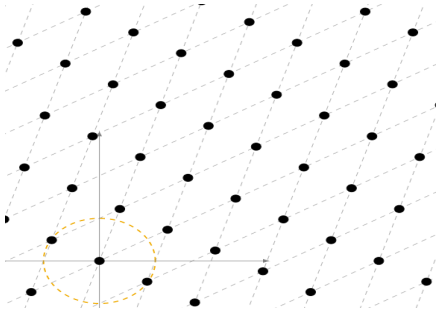


Figure 2.3: Shortest Vector Problem

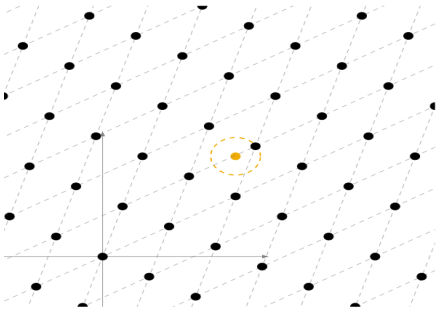


Figure 2.4: Closest Vector Problem

Chapter 3

Introduction to the cloud

In this chapter we will look in to the concept of the cloud and explain all different types of models in which the technology can be used. Towards the end we will explain one way in which SAP uses the cloud.

3.1 Origin

The cloud is, as boring as it might sound, nothing more than a blanket term. *The cloud* is easy to say and remember but gives little detail about what it is and how it can be used. In essence, the cloud describes the shift from static infrastructures to scalable and flexible infrastructures. In order to achieve flexibility of the infrastructure, most of it will be accessible from everywhere, location and device independent. To increase accessibility some parts of the infrastructure will be available and controllable over the internet, in other words it appears to be "in the clouds". In fact, the internet is just a means of transporting data. The actual data is located in ordinary servers just as in an on-premise system. The technique and type of infrastructure is thus not described by this term. This is where the more specific terms, cloud computing and cloud storage come into play.

Cloud storage refers to data accessible through the internet which is stored on a server. Cloud computing describes the processing of this data. Processing can be seen as a computation on certain numbers, changing data or removing data entirely. These changes require processing power, which can be provided by servers. The results can again be retrieved over the internet. In essence the server can do everything for the user, however it can also have partial functionality. That is why there are several terms to describe the infrastructure specific details. These are called the delivery and deployment models.

It appears that every paper has a different explanation, for a standardization on the terms we can take a look at the NIST definition [10]. We

will explain the exact definition of the standardized terms during the following sections. First we will describe the several roles that exist in the cloud landscape:

- The cloud provider, is the owner of the cloud computing and storage resources;
- The service provider, is the owner of the application that uses these resources;
- The user, consumes the cloud service;
- The customer, can refer to both the customer of the cloud resources and the cloud service.

Several companies can have multiple roles, e.g., Microsoft Azure owns the cloud resources that their own service, Microsoft Teams, operates on. This makes them both the cloud and service provider.

3.1.1 Cloud deployment models

Since the cloud offers a wide variety of services, we require a distinction between each service. The following models and services are designed in order to best describe the limit of each service. The deployment models describe how the services are provided, where they are physically located and who you share the infrastructure with.

Public cloud

Most people interact with the public cloud on a daily basis, sometimes without even noticing it. Some websites providing online video content, such as Youtube, Netflix or Twitch are hosted in the public cloud. In this deployment model the infrastructure is located at an external server owned by the cloud providers, e.g., AWS or Microsoft Azure, and the resources are provided over the internet. The model is characterized by its on-demand nature, resource pooling and rapid elasticity which makes it approachable and affordable.
For Netflix this would mean that they are the customer of the cloud resources, provided by AWS. The customer of Netflix would thus be the user of the cloud service.

Private cloud

The second deployment model is called the private cloud. The specific cloud storage and computing resources are allocated to a specific service provider. Since this is a more tailored approach to providing resources, the amount of control for the customer is greater than in the public cloud. Both in terms of

infrastructure and security architecture. The resources can be managed by the service or the cloud provider. The resources can be located on premise or off premise.
For Netflix this could mean that, e.g., their billing processes can be located in a private cloud.

Community cloud

The community cloud is an extension of the private cloud that allows access by multiple organizations or subdivisions of an organization often with a similar goal. This combines the benefit of sharing an infrastructure, without the massive scale in which this happens on the public cloud. This can both be on premise or off premise. This could be useful when multiple companies have to access the same application or service and want to share the application server, e.g., when having a time difference in the peak-usage.

Hybrid cloud

Any combination of a private, public or community cloud, sometimes also combined with existing on-premise infrastructure. An example of an application of a hybrid cloud is to separate business critical services from publicly accessible services.

3.1.2 Cloud delivery models

The delivery models specify to what extent the tasks are outsourced. These models range from just providing the software that has to run to full configuration of the underlying servers and networking.

Software as a service

This is the most extensive service. It will include everything that is required to run the application and also the application itself, e.g., Office 365. It is one of the easiest to use services and abstracts all the backend information away from the user. The user will not notice anything of what happens behind the scenes and can access the application and data from all devices.

Platform as a service

This service provides the platform to host and manage applications. The platform might also support several building blocks to create applications or a web based interactive development environment. The cloud often encourages micro-service architectures that can be structured and scaled using containers. These can be viewed as light-weight and often more efficient virtual machines.

Infrastructure as a service

In the previous service the low-level details about the OS, storage configuration and network components is taken care of by the cloud provider. In this model it is possible to specify exactly how you want the infrastructure to behave. Options can include server and storage configuration, network components and sometimes also the physical data center location. While the service providers are able to control these details, the cloud provider will still provide the resources. In this deployment model the customer has the most amount of control but also requires them to do a lot of the setup themselves.

3.2 Benefits and obstacles

As described in the introduction, for certain companies the cloud infrastructure has improved their ability to scale. This is not entirely due to the usage of the cloud, but also to their micro-service architecture. We think a large part of the added benefit can be attributed to the more efficient way in which resources are used in micro-service architectures. Companies with infrastructures like Netflix can benefit both from the scalability and cheap resources.

The service Netflix provides is especially a good fit with the micro-service architecture and the scaling. However, not all companies can instantly use this to cut costs and thus not every company can benefit equally from a transition into the cloud. Especially when the services include more sensitive data, a well tailored private cloud, in combination with an on-premise infrastructure are more likely to succeed and do not provide as much (cost) benefit as a public cloud.

From a high-level security perspective the cloud can introduce a lot of uncertainty due to its shared nature. However, the online nature of the cloud also forces the cloud providers to spend a lot of expertise and resources on the security of their services from the start. For some companies this might even mean a better security architecture after transitioning into the cloud.

That is why we think the cloud, as with all new technologies, should be used only in certain scenarios in which it truly has added benefit and security can be guaranteed.

3.3 SAP Cloud Platform

The SAP Cloud Platform is a cloud development and deployment platform [11]. The goal is to help customers in creating, hosting and linking applications in the cloud, also called the enterprise PaaS. It offers services and capabilities that can help build business applications for the cloud and link

them to both a cloud or on-premise infrastructure. It supports the developer through application development services and capabilities with features such as big data infrastructures or machine learning.

The applications that are being created can be deployed on either the Neo or Cloud Foundry environment. The Neo environment is fully operated by SAP on several servers across the world. These are specifically created for customers and run exclusively on a SAP infrastructure, it can be categorized as a private cloud.

3.3.1 Cloud Foundry

Cloud foundry is the open-source platform as a service that runs atop of the existing infrastructure [12]. The environment can be hosted on the infrastructure of different cloud providers such as Azure, AWS or GCP.

Cockpit

The cockpit is the launchpad that controls the PaaS, including Global Accounts and high level services [13]. Each global account can have several subaccounts, linked to a specific provider, region and environment. Within each subaccount you can create development spaces in which you deploy your application and launch service instances. These can be custom applications or pre-configured SaaS. The deployment often happens in a container infrastructure. Each container has access to a portion of the storage, memory and processing power. This micro-service architecture or containerized way of working allows for easy deployment of additional services when necessary.

SAP service marketplace

The SAP marketplace offers solutions supported and maintained by SAP. Services can range from SAP Hana Cloud to an entire Web IDE. The SAP Hana Cloud can be used as a building block in your application to store and process data whereas the Web IDE comes closer to a SaaS which can be used in the development. The Web IDE has features to develop, test, build and deploy directly on the PaaS. Due to the guidelines and templates you can quickly create or extend frontend applications and link them to a cloud HANA database. It also supports plenty of building packs for languages such as Java, Ruby and Python.

We will use The SAP Cloud Platform for the proof of concept described in Section 9. It provides us with a Web IDE, used to develop the application and the service marketplace, used to quickly add building blocks such as a database.

Chapter 4

Secure computation

In this chapter we will look into the concept of secure computation and which methods exist to achieve this. Towards the end we make a choice on which technique we will further research.

Secure computation is a research area focused on creating techniques and protocols that allow parties to jointly compute functions over private input, sometimes also referred to as secure function evaluation. The main goal is to provide the parties with the correct evaluation of the function without learning any information they are not supposed to. In a computation we have the function, the input and the output. All three aspects can be private and therefore we introduce three properties.

- privacy of input, refers to the input of a function, e.g., user preferences;
- privacy of output, refers to the output of the function, e.g., the result of a recommendation system;
- privacy of function, refers to the function itself, e.g., a machine learning algorithm.

There are several reasons why you can not share the input, output and function, e.g., competitive advantage, regulations or privacy concerns. However, sometimes we still want to get meaningful results from an outsourced or joint computation.

An example is the right to a secret ballot in, e.g., electronic voting [14]. In this case we want to guarantee privacy of the input since releasing who voted for or against in a referendum directly violates the right to a secret ballot. The function is not private and can be the summation of all the votes or a direct output of the winner. Since everybody can know the output, i.e., the winner, we also do not require the output to be private. If we can come up with a protocol or techniques with which we only reveal the winner of the referendum and not the individual votes, we can guarantee the privacy

of the input. This can be seen as one example that secure computation can solve.

In a cloud environment we want to outsource a computation and use the strength of the clouds computing resources. However, it is not always guaranteed that the cloud provider can keep the input, output or function private. This is why we look into secure computation techniques and reflect on whether they can be a useful extension to the cloud. There are several subfields that focus on the different properties previously described and also use different methods and techniques to achieve them:

- secure multi-party computation [15], focuses on multi-party protocols. In these protocols multiple participants compute a part of the output and share their results in such a way that no one participant has access to all the data;

- secure hardware, defines a secure environment through hardware-level isolation. This means the data does not leave the secure hardware in an unencrypted form even if, e.g., the operating system has been compromised;

- homomorphic encryption [16], is a cryptographic solution that makes it possible to perform calculations directly on ciphertexts.

To describe how successful these techniques are in terms of security, we often compare it to an ideal world scenario [17]. In an ideal world this problem is trivial, we can use a trusted third party that gathers all the input from the participants. The function can now be computed on all the input data. The result will be sent back to each participant and as a final step the input data will be deleted.

In the real world this problem is far from trivial, however we want to create a protocol, hardware or cryptography that can imitate the ideal world scenario without using an actual trusted third party. In the next sections we will describe the three main techniques in more depth and discuss which technique will suit the concept and goals of cloud computing.

4.1 Secure multi-party computation

Imagine a distributed environment in which multiple parties wish to perform a joint computation of some function. The goal of secure multi-party computation is to enable such a computation in a secure way. The computation should only reveal the result of the function and the parties can learn nothing more than what is absolutely necessary. Hence, secure multi-party computation considers how multiple parties can securely compute a function on private data.

In his paper called "Protocols for secure computations" [18], Andrew Yao introduced the concept of secure computation, later known as secure multiparty computation. His introduction proposes the problem in the form of a conversation between two millionaires who want to find out who is the richest. An obvious solution would be to tell how wealthy they both are and compare it, however, they do not want to reveal exactly how wealthy they are. This is called the Yao's Millionaires' Problem.

The protocol described, has to satisfy specific security and privacy constraints, such as input privacy, security against an eavesdropper or a dishonest participant. The protocol describes an algorithm that the participants have to follow, which if followed correctly can guarantee security and privacy.

The solution he proposed in his paper involves an active protocol in which both participants partially compute the result and combine their results using a technique called oblivious transfer [19]. Oblivious Transfer is a specific primitive often used in these protocols. It allows a participant to choose between two values without the other participants knowing what value has been chosen. Hence we choose between values while the other participant remains oblivious to which value has been chosen. He later generalized this problem from a secure comparison to the secure evaluation of an arbitrary function using Garbled Circuits [20].

The millionaire's problem has 2 participants, thus we speak of a secure two-party computation. Later this solution has been extended by Goldreich, Micali and Wigderson [21] to n-parties and thus becoming a secure multiparty computation.

The most famous example of an actual application that uses multi-party computation is a fully automated secure auction[22]. It was the first application of MPC which was seen as efficient. It allowed the auction to happen automatically without a single trusted party that has full responsibility.
To conclude this section we give a short recap of the general characteristics of secure multi-party computation:

- interactive protocol;
- requires communication between parties;
- supports input from multiple sources;
- often requires an honest majority;
- the result is revealed to all participants.

4.2 Secure hardware

The second technique is focused on a lower level and allows users to define secure enclaves in hardware. These secure regions remain confidential when the platform is under attack by malicious software. The applications are put into the enclaves with a special instruction set, an example is the Intel Software Guard Extensions [23]. Since the computation inside these secure regions still take place on plaintext, the performance of the computation itself should be just as fast. However, loading into secure memory and context switching still introduces performance overhead [24].

Although all hardware should be secure, not all hardware is called secure hardware. The addition of the word secure often means that it will protect against software level attacks. An obvious requirement for secure hardware is also that the hardware itself should be secure. The Intel SGX has an unclear trust model with regards to side-channel [25] and fault-injection attacks [26], as seen in several attacks [27]. Side-channel attacks try to attack the hardware by abusing the implementations instead of the algorithms. The implementations can leak, e.g., timing, power consumption or other side-channels that can be abused. Fault-injection attacks try to purposefully induce faults to force execution or output errors that can be abused to break the hardware.

An example of secure hardware that is both protecting against software level attacks and has good hardware security are secure cryptoprocessor, also called hardware security modules [28]. It often manages keys and performs encryption and decryption functions. Although it is not able to perform arbitrary computation, it can be useful in combination with either secure multiparty computation or homomorphic encryption. To conclude this section we give a short recap of the general characteristics of secure hardware:

- used to define secure areas in hardware;
- protects against software-level attacks;
- unclear trust model with regards to hardware security;
- computation on plaintext can have performance benefits.

4.3 Homomorphic encryption

Homomorphic encryption is a type of encryption that contains homomorphic properties and can be used to achieve secure computation. The term HE is an umbrella term to describe all encryption schemes that contain a homomorphism. This homomorphism allows the evaluation of an operation on the ciphertext, and through the properties of the scheme, directly change the plaintext. One can imagine that if we can calculate directly on, e.g., encrypted numbers, we do not have to reveal them during calculation. Since the result is once again a valid ciphertext, we also do not reveal the output.

This means that we satisfy both the privacy of input and output property at once while also correctly evaluating a computation. The challenging part is that we have to build this computation from the basic operations, e.g., addition and multiplication. To visualize this, we can use the following example.

Example 4.3.1. The function that we want to evaluate has three input values and can be described as follows.

$$f(x, y, z) = (x + y) \cdot z.$$

To ensure input privacy we encrypt (E) all values separately and use the homomorphic property to calculate the result.

$$f(E(x), E(y), E(z)) = (E(x) + E(y)) \cdot E(z) = E((x + y) \cdot z)$$

After decryption (D) we get the same result as if we used the function on the "normal" inputs.

$$D(E((x + y) \cdot z)) = (x + y) \cdot z$$

Some calculations, e.g., a summation of votes, are easy to perform with the addition operation, while other more advanced calculations are harder to express efficiently. However, in essence this gives us a "simple" technique to perform secure computation. To conclude this section we give a short recap of the general characteristics of homomorphic encryption:

- inherent performance overhead;
- allows for outsourcing of the computation;
- advanced functions are harder to represent.

4.4 Combination with the cloud

When we try to implement secure computation in real-life, there are many choices that can be made to optimize performance for different architectures and different application scenarios. Sometimes a combination of these techniques should be used. The architecture we analyse in this thesis is the cloud architecture and more specifically the addition of secure computation techniques to the SAP Cloud Platform. Therefore, we will look at the key characteristics of the techniques once again and see if they align with the goals of the cloud.

Secure multi-party computation (SMPC) is highly focused on the protocol and on sharing the computation. While the cloud focuses more on replacing computational power at the edge and shifting it towards more efficient servers. To benefit from the cloud we want our solution to be computationally heavy on the cloud side instead of sharing the computational load. However, HE can also be used inside the protocols described in SMPC. This creates a hybrid approach as mentioned earlier. Therefore, SMPC can also benefit from the cloud and SMPC can perhaps also increase the security of the cloud. However, for this thesis we will focus on HE.

Although secure hardware might also be valuable in certain scenario's, the lack of transparency of the manufacturer and the lack of proper side-channel protection lead to an unclear trust model. HSMs are a part of many secure environments and work well with regards to key storage, encryption and decryption. This can also be useful in combination with HE.

Since HE is a cryptographic primitive it can be provided as a software library. The evaluation algorithms can be implemented on the cloud providers side, where all the processing takes place. The encryption and decryption algorithms can be implemented at the user side, where all the data is generated. This should in theory be easy to add to an existing cloud platform as an additional service. The main concern is the amount of performance overhead this will bring to the applications.

In general, we can see that HE can provide the extra privacy for users in a cloud environment in which the cloud only has to evaluate a function on encrypted data. It also allows the cloud to use its cloud computing resources efficiently. Therefore, the rest of the thesis will be structured around HE as the secure computation technique that will be further researched and discussed.

Chapter 5

Cloud security

In this chapter we look into the main threats to the cloud. This helps us identify in which parts of the cloud landscape, secure computation can have added benefit.

In order to assess whether or not HE might be useful, we will look into the cloud security issues. Although we cannot exactly determine the security issues, we can look at recent threats and trends. The Cloud Security Alliance (CSA) reports on their identification of the top threats to the cloud [29], called the egregious eleven. The goal of their report is to raise awareness of threats, risks and vulnerabilities. We will discuss every threat, reflect on its impact on confidentiality and whether or not secure computation can protect against it. Every threat in the report also includes a STRIDE analysis. This model has been developed by Microsoft and is used to classify the threats [30]. It is an acronym for:

- **S**poofing identity;
- **T**ampering with data;
- **R**epudiation;
- **I**nformation disclosure;
- **D**enial of service;
- **E**levation of privilege.

While the CSA prefers to use the STRIDE framework, we will use the terms confidentiality, integrity and availability. We will mainly reflect on information disclosure, which relates to confidentiality. Integrity is related to tampering with data and availability is related to denial of service. We will shortly describe the egregious eleven threats:

1. **Data Breaches** are the main threat to confidentiality of data in the cloud. Data disclosure can be the consequence of many of the threats we will mention later and is often the most valuable for an adversary. Personal health information, financial information, personally identifiable information, trade secrets and intellectual property are all very valuable, e.g., on black markets, to governments or competitors. It also impacts the victims brand value and that of the cloud provider.

2. **Misconfiguration and Inadequate Change Control** can also lead to data disclosure. Most commonly caused by a wrong setup of a (public) system. While the system itself may be perfectly secure, default credentials or disabled security controls can still lead to unwanted access to data. Either due to negligence or lack of knowledge.

3. **Lack of Cloud Security Architecture and Strategy** is often related to the differences in the way public clouds and on-premise infrastructures need to be protected. This can lead to incidents that were improbable in an on-premise infrastructure.

4. **Insufficient Identity, Credential, Access and Key Management** can result in direct access to plaintext data. In a normal setup we often have key management located at the cloud provider. Insufficient control can thus lead to decryption at the cloud provider. In this scenario encryption (at rest) will lose its value. An obvious solution is to shift the key management towards on-premise HSMs or other key management techniques. In essence, this gives control back to the user, in a way that on-premise infrastructures can. This also means that the users now have to protect their keys themselves, which could be a good or bad thing. A downside to this solution is that the cloud cannot perform any calculations on the data.

5. **Account Hijacking** can be the consequence of the previous threat. It can also be caused by phishing, exploits or stolen credentials. The account with the highest value is often the account with the highest privileges. These privileges can lead to sensitive data or control over the applications. While the first one affects confidentiality and/or integrity, the last one can impact availability if the attacker wants to disrupt the applications.

6. **Insider Threat**, is always a hard threat to solve since the insider has legitimate access. When there is an insider threat at the cloud provider side, depending on the amount of privilege, this attacker can access confidential data located in customer applications or storage. This data can be encrypted at rest, however, insider threats often have legitimate access to the decrypted data. If there is a separation between on-premise and the cloud, through own key management, this

can be partially avoided. When there is an insider threat at the on-premise location, this is nearly impossible to avoid.

7. **Insecure Interfaces and APIs** are the most public part of an application, this means that it will be attacked a lot and should be secure. Insecure APIs can give access to data that is used in applications. This often is plaintext data since it has to be used in applications.

8. **Weak Control Plane** refers to the fact that the service providers and users do not have enough control over the security architecture. For example, they want to include their own key management and use their own encryption solution but the cloud provider does not support this. This limits the control of the service provider and prevents them from actively securing their applications. This also makes them more dependent on the cloud provider for security.

9. **Metastructure and Applistructure Failures** describe threats that are caused by the protocols and communication between the different layers of the cloud. This is quite an obscure threat and we can not conclude whether or not HE can help in these situations.

10. **Limited Cloud Usage Visibility** occurs when a service provider does not have enough tools to visualize the usage of cloud services. Detecting malicious or abnormal behaviour of applications will thus be hard. Abnormal behaviour can occur when cloud services are being exploited, e.g., by botnet malware or cryptocurrency mining malware.

11. **Abuse and Nefarious Use of Cloud Services** refers to the fact that the adversary can also use the cloud. This does not impact confidentiality of data directly, however, it can be beneficial for an adversary to be located in the same physical infrastructure. Information extraction or privilege escalation attacks can be the consequence of bad virtualization at the cloud provider [31].

5.1 HE in the cloud

All threats described can result in data disclosure, HE can only help in a limited amount of cases. Data breaches (1) are the main threat to confidentiality and shows us that we should prevent and minimize access to the data in the cloud.

In the case of misconfiguration (2), we should look into how we can reduce the amount of data that can be disclosed. For example, through encryption at rest. The data that will be disclosed is now encrypted instead of plaintext. However, this all falls or stands by key management. Insufficient Identiy, Credential, Access and Key Management (4) is not acceptable and

will discard all the value encryption at rest will provide us. Shifting the key management from the cloud provider to the user can thus be seen as a good development. This would separate the keys and the data. A downside to this solution is that we cannot compute on encrypted data directly in the cloud. HE can solve a part of this problem by allowing direct computation on encrypted data.

Another downside of storing the keys in the cloud is that cloud accounts have legitimate access to the data. This removes encryption at rest for those accounts that have the correct privileges and allows them to access the plaintext data and applications. Therefore, encryption at rest, without separating keys and data, does not protect against Account Hijacking (5) and Insider Threat(6).

Threats can also occur inside cloud services themselves. Insecure Interfaces and APIs (7) allow an adversary to extract plaintext data that resides inside the services. Inside these services the encryption at rest is removed in order to use the data. With HE we can create secure cloud services that operate on ciphertext data. This means that insecure APIs only disclose encrypted data instead of plaintext data.

Most of the threats reflect on outsider threats, however, insider threats also occur when an adversary resides in the same cloud as the target. Abuse and nefarious use of cloud services (11) can thus also be a threat to the data of cloud users. While it is good to highlight the good points of HE, we should not forget the fact that, just like all cryptography, it can also be used by the adversary. The fact that the cloud provider does not know what data is stored on their servers, or the fact that they do not know exactly which computation they are running, might cause a problem for certain scenarios.

Although most of these threats are not directly caused by insecure applications or the application layer, we can see that implementing HE at this layer can indirectly protect the other layers. We also see that ten out of eleven threats can result in data tampering, this is another crucial part of applications in the cloud and something HE can not provide any guarantees about. The report also reflects on the recent trend that sees the threats shift from the cloud providers responsibility to threats related to the user responsibility. This somewhat indicates that the security of the cloud infrastructure is improving at the cloud providers side.

To conclude this section we want to discuss a new threat in the list. For this thesis one of the more interesting threats is the Weak Control Plane (8). Perhaps the name is not that descriptive but it reflects perfectly on the need for additional means to protect data as a user. For example, in the form of secure computation in the cloud. Secure computation is a technique that a user might want to implement in their applications. This means cloud/service providers should support the option to use secure computation

since it empowers users with techniques to actively secure their data from the design. It gives users the extra security they might want for sensitive applications and helps them to securely transition into the cloud.

Although the report does not give us an exhaustive list with all the security threats and issues, it does give a good perception of the key issues. This gives us some more tangible points with which we show why HE can have added benefit for the cloud and thus we hope it can improve the cloud security.

Chapter 6

Homomorphic encryption

In this chapter we dive deeper in to the topic of homomorphic encryption and specify several classifications of the schemes. We will look into the techniques used to increase performance and shortly reflect on the performance. At the end we will show the mathematical background of the BFV scheme.

In their 1978 paper, *On data banks and privacy homomorphisms*, Rivest, Adleman and Dertouzas first spoke of a so called privacy homomorphism [16]. The observations described in this paper perfectly reflect the contradiction that currently exists in the cloud. Although encryption can preserve the confidentiality and privacy of the data, the only useful operations that can be performed are storage and retrieval. If we want to compute on this data, we need to decrypt it. In their paper they describe this in the following way:

> *"One might prefer a solution which did not require decryption of the user's data (except of course at the user's terminal). That is, the hardware configuration will be that of Figure 1, but the encryption function used will permit the computer system to operate on the data without decrypting it."*

When looking at Figure 1 (See appendix A) we can exactly see the analogy with the cloud. The paper also states the concept of time-sharing, which we would nowadays call cloud computing. Even the need for sharing the resources stayed the same. In 1970 it was too expensive to inefficiently use a computer, while in 2020 it is still expensive to inefficiently use computers. In both time frames the concept of privacy during computation is something we want to ensure, whether we call it a data bank or the cloud, it should not matter.

When talking about homomorphic encryption, we often refer to the entire set of algorithms and properties. This means that every homomorphic

scheme should be a secure encryption scheme, but not every secure encryption scheme is homomorphic. A traditional encryption scheme specifies the key generation, encryption and decryption algorithm. Homomorphic encryption refers to a normal encryption scheme, which has been extended by the evaluate algorithm. This evaluate algorithm allows for manipulation on ciphertext while directly impacting the plaintext.

Since 1978 several schemes have been published which are used in everyday applications, e.g., RSA and ElGamal. However, these schemes are not particularly famous because of their homomorphic properties. This was often seen as a weakness rather than a strength since this introduces malleability. In Timeline 1 we can see a summary of the evolution of this research field. There are several ways these schemes are classified in the literature based on their "level" of homomorphic properties. In general we can divide them in three categories:

- Partial Homomorphic, supports the evaluation of an addition or multiplication an unlimited amount of times;
- Fully Homomorphic, supports the evaluation of an addition and a multiplication an unlimited amount of times.
- Somewhat Homomorphic, supports the evaluation of an addition and a multiplication a limited amount of times.. Improving this building block is the focus of many subsequent research.

In the next sections we will explain each classification in depth.

TIMELINE 1: *History of the Privacy Homomorphism*

1978	RSA [32]
1978	Rivest et al. introduce the privacy homomorphism [16]
1984	Goldwasser & Micali [33]
1985	ElGamal [4]
1999	Paillier [34]
2005	Boneh, Goh & Nissim [35]
2009	**Gentry** [36]
2011	Brakerski & Vaikuntanathan [37]
2012	Brakerski, Gentry & Vaikuntanathan [38]
2012	Brakerski, Fan & Vercauteren [39][40]
2013	Gentry, Sahai & Waters [41]
2016	Cheon, Kim, Kim & Song [42]
202?	Practical homomorphic encryption [**?**]

6.1 Partial homomorphic encryption

Just before their paper on data banks, Rivest and Adleman together with Shamir released their RSA encryption scheme [32]. It is no coincidence that the scheme they describe has a partial privacy homomorphism. We can describe this with the following example.

Example 6.1.1. (Multiplicatively Homomorphic RSA)
See Definition 2.1.1 for the textbook RSA encryption scheme. If we take two message, m_1 and m_2, we get the following two ciphertexts: $c_1 = {m_1}^e \mod N$, $c_2 = {m_2}^e \mod N$.
If we multiply the two ciphertexts, we get: $c_3 = {m_1}^e \cdot {m_2}^e \mod N$
Which results in: $c_3 = (m_1 \cdot m_2)^e \mod N$.
Decryption will give us the new message: $m_3 = {c_3}^d = (m_1 \cdot m_2)^e = m_1 \cdot m_2 \mod N$.

In this example we can see that by multiplying the ciphertext c_1 with c_2, we implicitly multiply the two messages m_1 and m_2.

Definition 6.1.1. (Multiplicative property) An encryption scheme has the multiplicative property if we can combine two ciphertexts with the group operator and on decryption we will receive the multiplication of the messages.

The ElGamal cryptosystem was introduced in 1985[4]. This crypto system also has the multiplicative property.

Example 6.1.2. (Multiplicatively Homomorphic ElGamal)
See Definition 2.2.2 for the ElGamal encryption scheme.
$c_1 = (g^{r_1}, m_1 \cdot h^{r_1})$
$c_2 = (g^{r_2}, m_2 \cdot h^{r_2})$
$c_3 = c_1 \cdot c_2 = (g^{r_1} \cdot g^{r_2}, m_1 \cdot h^{r_1} \cdot m_2 \cdot h^{r_2})$
$c_3 = (g^{r_1+r_2}, (m_1 \cdot m_2)h^{r_1+r_2})$
$m_3 = m_1 \cdot m_2$

We can use the product rule of exponentiation to transform the multiplicative property into an additive property.

Example 6.1.3. (Additively Homomorphic ElGamal)
See Definition 2.2.2 for the ElGamal encryption scheme.
$c_1 = (g^{r_1}, g^{m_1} \cdot h^{r_1})$
$c_2 = (g^{r_2}, g^{m_2} \cdot h^{r_2})$
$c_3 = c_1 \cdot c_2 = (g^{r_1} \cdot g^{r_2}, g^{m_1} \cdot h^{r_1} \cdot g^{m_2} \cdot h^{r_2})$
$c_3 = (g^{r_1+r_2}, (g^{m_1+m_2}) \cdot h^{r_1+r_2})$
$t_1 = g^{m_1+m_2}$
Solve discrete log to get $m_3 = m_1 + m_2$.
Note that this only works for a short message m_3, otherwise it will be hard to compute the discrete log, as it should be.

Definition 6.1.2. (Additive property) An encryption scheme has the additive property if we can combine two ciphertexts with the group operator and on decryption we will receive the addition of the messages.

In 2005 Boneh et al. [35] released the first scheme that could do both. The main contribution in this paper is the first homomorphic encryption scheme which allows both addition and multiplication. However, the amount of multiplications is limited to one. Before and after this multiplication, we can perform any number of additions.

This scheme is close to a privacy homomorphism but the true definition of fully homomorphic encryption refers to an unlimited amount of computations of both addition and multiplication on encrypted data. RSA, ElGamal and many others that only contain a partial property are thus classified as partial homomorphic encryption.

6.2 Fully homomorphic encryption

As mentioned shortly in the previous section, fully homomorphic encryption refers to both the operations supported but also to the unlimited nature. In the literature this is also called an evaluation of a function. When we talk about functions we often refer to the process of transforming some input, in this case two ciphertext. Functions can be expressed in many ways, but for homomorphic evaluation we often prefer to represent it as a circuit. A circuit describes a calculation or function with low-level components. In this case the components are addition and multiplication. We can express these operators on different levels, a boolean and arithmetic circuit [43].

Boolean circuits can express a function in binary gates. With the addition and the multiplication we can create a NAND gate. The NAND gate is functional complete, which means we can create every circuit by using NAND gates. Therefore we can create every function or computation possible, making it fully homomorphic.

An example of a function that might be useful for secure computation is a secure comparison. With two input values, we want the result to tell us whether they are equal or not. We can write the wanted output of a circuit in the form of a truth table, see Table 6.1.

Once we have a circuit representing the function that we want to compute on plaintext, we can translate this to an evaluation on the ciphertexts using the homomorphic operators. One could imagine that the amount of gates and thus the depth and complexity of the circuit grows once we have multiple input bits. More complex functions, circuits or computations in general will result in more gates used. Hence more gates equals more multiplication and additions.
Opposed to the low-level circuits of a boolean circuit, an arithmetic circuit

X	Y	Output
0	0	1
1	0	0
0	1	0
1	1	1

Table 6.1: Truth table

can operate on a higher level. The gates translate to the arithmetic operations of elements in a field. Since we work on a higher level this is easier to comprehend and implement in applications directly. To shortly recap, boolean and arithmetic circuits are used to represent a computation. This is a good way to represent functions on encrypted data because we can only use specific operators or gates. However, the main question remains. How can we achieve addition and multiplication on encrypted data?

Breakthrough

In 2009, Gentry published a paper in which he explained how we can achieve addition and multiplication on encrypted data through lattices [36], this provides the building blocks for the previously mentioned circuits. However, due to the way the plaintexts are encrypted, a noise term is introduced in the ciphertext. When performing calculations on the ciphertexts this noise increases. During decryption this noise is removed and the plaintext can be recovered. If this noise grows too large, the decryption will fail. This means that the encryption scheme introduced by Gentry does support both addition and multiplication but not an unlimited amount of times. The intuition behind his solution to this problem is to define a circuit that represents the decryption algorithm. The homomorphic version of the decryption algorithm requires an encrypted version of the private key and encrypted input, the output is also encrypted. This allows us to refresh the encrypted input and thus remove the noise, without having to decrypt it to plaintext first. Another way of looking at this is refreshing the ciphertext to reduce noise. This solution is called bootstrapping.

If a scheme is strong enough to support bootstrapping and one additional NAND gate, it is a fully homomorphic encryption scheme. Even though we can theoretically and technically achieve fully homomorphic encryption now, it is far from practical due to its performance overhead. With performance in mind, there are two ways to proceed. Improve the speed of bootstrapping or increase the power of the somewhat homomorphic scheme such that it can evaluate more complex functions before decryption fails.

6.3 Somewhat homomorphic encryption

Since the bootstrapping procedure brings a lot of performance overhead we often do not want to use it or we need to increase its efficiency. Another downside is the requirement of the encryption of a private key, which could have unwanted implications, also refered to as circular security [44].

Not using bootstrapping limits the expressivity of the computations, i.e., the depth of the circuits but provides us with faster evaluation. To increase the expressivity again we want to reduce the noise growth. The more computations we can perform before the noise becomes too large the better.

Somewhat homomorphic encryption and more specific leveled homomorphic encryption are schemes that can handle functions of a limited amount of depth, thus not fully homomorphic. The recent improvements in the field of SHE focus on reducing the amount of noise growth, reducing the size of the parameters and increasing the speed of the evaluation. Other improvements can be made with regards to simplicity of the security reductions, i.e., base them on well studied hardness assumptions and remove additional assumptions.

In 2011 an improved variant of Gentry's SHE scheme was proposed by Brakerski [45]. This scheme is based on the Learning with Error hard problem introduced by Regev [8], which can be reduced to a variant of the shortest vector problem. It also makes any additional assumptions obsolete. They introduce two new techniques:

- Relinearization, allows us to perform a multiplication and reduce the resulting ciphertext size back to its initial size;
- Dimension-modulus reduction, is used to reduce the size of the parameters of the ciphertext.

Both techniques will result in more practical systems because of the reduced sizes of parameters and ciphertext.
Subsequent improvements are made by extending the LWE problem to a ring variant[40]. The Brakerski, Gentry and Vaikuntanathan scheme (BGV) and the Brakerski, Fan and Vercauteren scheme (BFV) are considered the most promising schemes for practical performance. The security of both schemes is based on the hardness of the RLWE problem. This ring variant allows for a lot of different optimizations:

- Ciphertext packing, combine several data points into a single ciphertext;
- Single Instruction Multiple Data (SIMD) [46], perform the same instruction on all these data points;

- Residue number system variants [47], used to split big integers in smaller parts for calculation and combine them later through, e.g., the Chinese Remainder Theorem.

6.4 Performance

As seen in the previous section, the performance is impacted by a lot of different factors. From a high level perspective we can see two things, firstly that the encryption and decryption of the data are pure performance overhead. Since a plaintext computation does not need this additional step. Secondly, the homomorphic evaluation should be compared to the plaintext equivalent.

For example, instead of multiplying two integers directly, we now multiply polynomials in which the integers are encoded and encrypted. However, this ciphertext polynomial can include several integers using the batching technique. Therefore, it would not be efficient to perform just one integer multiplication. Thus comparing an integer calculation directly to its homomorphic equivalent is hard to do. This is even more complicated for more advanced functions and especially if they contain a lot of multiplications. More multiplications directly impact the parameters and thus also the speed of the evaluation.

In order to achieve maximum performance we also need to represent the function as an efficient circuit, optimize the parameters for this specific circuit, optimize the security parameters and also know all the optimizations of the underlying scheme. We conclude it is not possible to compare a plaintext computation to a HE computation without knowing all these aspects. Optimizing in all these areas is also something that has to be further researched and perhaps automated to a certain extend by encryption libraries. Although we cannot directly compare the functions, and several schemes and papers define their speed differently, we can roughly describe the improvements by looking at their measurements. Since the evaluation speed is dominated by the multiplication speed and bootstrapping speed, we will now show several measurements on those operations and show the improvement.
In the first implementations of Gentry's scheme the time of a bootstrapping operation ranged from 30 seconds to 30 minutes depending on the parameters. The parameters are used to ensure the hardness of the underlying problem and thus the security level and to allow several different sizes of data to be encrypted. A small setting can be seen as a dimension of 2048, this makes public-key sizes around 70 megabytes and the time to run one bootstrapping operation around 30 seconds. For larger dimensions such as 32768 this could increase to public-key sizes of several gigabytes and a bootstrapping operation in around 30 minutes [48].

In a paper from 2016, bootstrapping speeds were achieved sub 0.1 seconds. This is not directly comparable to the Gentry setting since it uses a different scheme and parameters. However, this does indicate that a lot of progress has been made on reducing the bootstrapping time [49].

In recent papers focused on optimizing multiplication and relinearization, speeds were achieved between 3ms and 174ms for multiplication and between 0.41ms and 78 ms for relinearization, depending on the depth of the circuit [47]. This is also a significant improvement compared to the several seconds per multiplication in previous schemes.

This short section shows that although it is hard to compare the different calculations directly, we still see some great improvements in terms of speed of the homomorphic components. However, this comparison will always be in the favor of plaintext computation since performance overhead is unavoidable, yet the gap is closing.

While the improvements to the algorithms are very promising, we can also improve on hardware implementations, e.g., GPU, ASICS and FPGA implementations and multi-threading. In addition to this the the amount of computational power will increase with time, making this difference less noticeable.

When looking at the applications from this performance perspective we have to consider whether or not the performance overhead is noticeable in a particular calculation and whether we want to increase the privacy/confidentiality of the data or that we need fast computations.

6.5 BFV scheme

The BFV scheme is one of the most recent advancements in homomorphic encryption. It is based on the RLWE hardness assumption which allows for previously described optimizations. This scheme has also been implemented in recent homomorphic software libraries such as Microsoft SEAL and PALISADE. To get a greater understanding of what this scheme looks like we will go over the basic version of BFV. We especially explain how the evaluation functions operate and how the previously mentioned noise is introduced [47].

In Ring-LWE we work on a polynomial ring. This means that messages are encoded into a plaintext polynomial and a ciphertext consists of two polynomials, c_0 and c_1.

1. Sample a secret key S, which is a random element.

2. Generate the public key consisting of $[pk_0, pk_1]$, using S, a random element from the ring A and a sampled error rate e. Output $[-(A \cdot$

$S + e), A]$. This public key is an instance of the RLWE problem and thus both parts of the public key should be indistinguishable and this means it is not possible to recover S.

3. To encrypt a message M, first we need to encode it into a plaintext polynomial. We sample a new element U from the ring and two new error samples e_1 and e_2. After which we generate RLWE instances, c_0 and c_1.
   c_0 consists of $[pk_0 \cdot U + e_1]$ and the encoded message M.
   c_1 consists of $[pk_1 \cdot U + e_2]$. Since this is another RLWE instance, the hardness assumption prevents the recovery of M from $c0$.

4. To perform an addition between two ciphertexts, we add both c_0 parts and both c_1 parts together, hence the new c_0 will now include the addition of M_1 and M_2.

5. Decryption is performed by computing $[c_0 + c_1 \cdot S]$, this will use the private key to cancel out the S in the RLWE samples, the last step is to round the result to remove the error and get the message M.

Although this is an over-simplified version of the BFV scheme, we can clearly see that the error terms in the messages originate from the sampled error terms required in an RLWE instance. When combining two ciphertexts through addition, the error terms are also added. This indicates that the error term increases as we perform more and more additions and multiplications. For multiplications this error term is multiplied and thus will increase more than on an addition. The decryption involves a rounding operation in which the error term will be cancelled out. If the error term will grow too large, this rounding will fail and the decryption will fail.

From a high-level we can see that the public key masks the message and that the RLWE assumption prevents us from recovering both M and S from the created RLWE instances. This makes a trapdoor function because we can only remove this masking with the private key and retrieve the message.

Chapter 7

Security of HE

The use of homomorphic encryption can only be justified in certain environments. In this chapter we will look into additional security requirements while reflecting on the weakness inherent to HE.

7.1 Secure architecture

Homomorphic encryption is a cryptographic primitive. Cryptographic primitives are used to create secure protocols and architectures. Encryption schemes are building blocks that can provide confidentiality. Homomorphic encryption is a more advanced building block that allows evaluation of a function on encrypted data. However, this also makes the encryption scheme malleable. This property often conflicts with traditional measures to guarantee integrity and thus introduces weaker notions of security. Therefore this has to be taken into account when developing protocols and architectures with homomorphic encryption.

As an example we take the most basic version of an outsourced computation. In this setup we encrypt data, send it over to the third party for computation and receive the result. This simple architecture already has several problems from a security perspective which we will further describe in the rest of this section.

7.1.1 CIA model

A simple and general way to describe the additional requirements is through the CIA model, confidentiality, integrity and availability. While encryption achieves confidentiality when used properly, integrity and availability aren't provided. Confidentiality makes sure that the third party can not see and decrypt our data, integrity refers to the fact that we want to make sure that the third party does not change our data. One could imagine that it is quite hard to change encrypted data exactly the way we want it to. Attempts to

change encrypted data reliably, often result in malformed data and decryption failure. However, one can not assume that its not possible to change the data reliably. Especially when using homomorphic encryption, since it gives us this exact feature by design. This is why we need to have additional security measures to guarantee integrity in protocols and applications that rely on HE or encryption in general.

The last goal in this model is to guarantee availability. This allows us to access the result and the service reliably. This is something the cloud can provide us through high up time and reliable access to our data. The fact that decryption might fail due to too many computations, can also be seen as a way to disrupt availability, i.e., go over the noise budget on purpose to prevent decryption. This can lead to bad data quality, i.e., we expect our data to be located in a certain ciphertext but it has been changed or even discarded completely. Therefore, we can see that threats to integrity can also lead to a threat to availability. If we want to securely create architectures with HE, we need additional controls to guarantee availability as well.

7.1.2 Verifiability

An extra property that we would like to achieve, which is related to integrity, is called verifiablity. The goal of this property is to give guarantees as to which function has been calculated. We can achieve this through Zero-Knowledge Verifiable Computation [50]. This means that the party that computes the function has to provide the user with some sort of proof of computation. This proof has to be verified by the user, and thus provides the guarantee that the result corresponds to the evaluation of the function.

7.2 Secure schemes

From an architecture perspective, we can see that we need extra controls in terms of integrity, availability and verifiability. When purely looking at homomorphic encryption as a primitive, there are several ways in which homomorphic encryption might be less secure than more classical asymmetric encryption. It is important that the primitives used as building blocks itself are secure, in this section we will look into several requirements and assumptions made to describe the security.

We can use several different notions from literature to describe basic types of attacks against encryption schemes [9].

Definition 7.2.1 (Ciphertext-only Attack)**.** The adversary can observe a ciphertext and tries to determine the plaintext.

Definition 7.2.2 (Known-plaintext Attack)**.** The adversary can observe pairs of corresponding plaintexts and ciphertexts.

Definition 7.2.3 (Chosen-plaintext Attack)**.** The adversary can observe pairs of corresponding plaintexts and ciphertexts and, as the name suggests, choose two plaintexts. The challenger encrypts one of the two plaintexts and challenges the adversary to determine which one has been encrypted. A scheme is CPA secure if there is no adversary that can achieve this with more than negligable advantage over random guessing.

Definition 7.2.4 (Chosen-ciphertext Attack)**.** The adversary can observe pairs of corresponding plaintexts and ciphertexts and, as the name suggests, choose ciphertexts. After the challenge has been send the adversary is not allowed to ask for the decryption of ciphertexts anymore and especially not the decryption of the challenge. A scheme is CCA secure if there is no adversary that can achieve this with more than negligable advantage over random guessing.

Each attack model shows a different power level of the adversary. Depending on the real-life application in which encryption is used, we can determine which model fits best. For the cloud we can make some assumptions to the strength of the adversary.

Ciphertext-only attacks are very likely since the data is located at the cloud provider. This means it has access to encrypted data, which it can observe. It might also be possible that an adversary has a pair of ciphertext and plaintext, e.g., when results are published. These are the two passive models a secure HE scheme has to protect against.

The other two are active models, this means the adversary can choose which plaintext or ciphertext will be revealed. These attacks are far more invasive, but not impossible in real-life. Since it is likely that we will be encrypting the same plaintext multiple times, we require the encryption to be different each time. This property is called semantic security, i.e., two encryptions of the same message should be indistinguishable. Therefore, CPA security should be guaranteed by any secure HE scheme. In almost all HE schemes this is guaranteed through randomized encryption.

CCA attacks are harder to prevent since HE is designed to be malleable [51]. If we want to create CCA secure encryption, the homomorphic properties will often be lost and this defeats the purpose of HE. Although this cannot be resolved strictly by the scheme itself, certain schemes require evaluation keys. This prevents performing calculations by random attackers, and thus providing some form of non-malleability. However, this does not solve the problem fundamentally.

7.2.1 Security reductions

As seen in the previous chapter, the BFV scheme is based on the RLWE hardness assumption. We can tell something about the hardness of certain problems by trying to make a reduction to other well-studied hardness assumptions. A security reduction proves that the new assumption is at least as hard to break as the problem we reduce it to. This means that if we can break RLWE, we can use it to break more classical lattice assumptions, e.g., BDD, SVP or CVP [52]. Since these older lattice assumptions are well-studied and the most efficient attacks against these problems are not fast enough, we can assume that attacks against the new assumption are also not fast enough.

To ensure that these problems are actually hard, the parameters have to be correctly set. Selecting the correct parameters for the encryption schemes should be supported such that we can reliably base our security on the underlying hardness assumption. With wrong parameters this underlying problem is not hard anymore and decryption would be trivial.

Chapter 8

Use cases

In this chapter we will look into a usecase in which the combination of HE and cloud are described. We will generalize this usecase and look into other potential usecases.

In the introduction we shortly mentioned Netflix and how they benefit from the cloud. They can quickly upscale their streaming service to several thousand servers providing cloud services to the user. When Netflix gains a lot of users in quick succession this is very valuable. What is even more valuable is to retain users and keep them on the platform for as long as possible. Their goal is to maximize the total amount of hours streamed. This is often achieved through a recommendation system. In order for them to make accurate predictions, a detailed customer profile has to be created. This will improve prediction accuracy but requires the collection of as much data as possible. This collected data can lead to privacy risks for its users [53]. This creates a conflict between the accuracy of the predictions and the amount of privacy for the users. This calls for a technical solution that should preserve the privacy of the user. Threats to this privacy can not only come from Netflix or the cloud provider but also from vulnerabilities in the cloud, as mentioned in Chapter 5. This solution should also not conflict with the benefits that Netflix receives from the cloud.

8.1 Recommendation systems

Netflix' recommendations are displayed on the main webpage that is dynamically generated. Each row contains a specific theme, e.g., "trending now" or "recently added". Within a theme the movies are ranked from least to most accurate. The movies located on the left side are the best recommendation for a specific user. The way in which they determine the ranking is called, personalised video ranking. Two common ways to realise this are collaborative filtering and content-based filtering. Collaborative fil-

tering uses input from multiple users to predict the preferences of a single user. Content-based filtering uses a description of items, in this case movies and series, and a list of user preferences. While collaborative filtering can predict what you would like to watch based on your history and the history of others, content-based filtering directly compares your preferences to a list with similar items. Since collaborative filtering uses input from multiple sources the use of HE in this specific scenario will be hard, therefore we consider content-based filtering. This requires the creation of features describing the movies and features describing the user. The next step is to find similarities between both features. These similarities can be used to create future recommendations, for Netflix there is also an incentive to keep this secret. Other companies that want to create a recommendation system for their streaming platform can replicate it, which is also a reason why the user is not allowed to compute this locally. The calculations used in this filtering, e.g., weighted sum, can be computed using HE. The weighted sum is used to calculate the weighted mean. This directly translates to a ranking with which the user can see which movies are similar to the ones that have been scored. The calculation uses the following input:

- S is a similarity matrix, e.g., position (1,2) shows the similarity between movie 1 and movie 2, ranging from 1 to 10;
- $\vec{p}$ is the preference vector containing scores on movies, ranging from 0 to 5. Note that in the second calculation, this ranges from 1 to 5 and only includes the rated movies;
- $\vec{p'}$ is the vector containing a 1 if the movie has been rated and 0 otherwise.

It outputs the recommendation vector, $\vec{r}$. There are two ways to compute the recommendation.

1. On encrypted $\vec{p}$ and encrypted $\vec{p'}$:
   (a) $\vec{t} = S \cdot \vec{p}$
   (b) $\vec{u} = S \cdot \vec{p'}$
   (c) $\vec{r} = \frac{\vec{t}}{\vec{u}}$, this will be calculated entrywise.
2. On encrypted $\vec{p}$ and $\vec{p'}$:
   (a) Multiply all preferences in $\vec{p}$ with corresponding similarity scores of the movies in each row in S. Either by searching the scores directly or filtering the matrix in advance using $\vec{p'}$;
   (b) Calculate the pairwise addition of the resulting rows;
   (c) Calculate the sum of the similarity scores which are part of this addition;

(d) Divide previous two results pairwise.

While the first solution requires the vector to contain zero values, the second one only requires an encryption of the ranked items. This would be more efficient considering that the total amount of movies greatly surpasses the amount of movies rated, thus increasing the size of $\vec{p}$ unnecessarily. Therefore, we give an example of the second calculation. Note that in the second case $\vec{p'}$ is not confidential. The service provider learns which movies have been rated but not the actual rating. This would in theory provide less privacy, however, in practice the service provider can potentially also learn the watched movies from side-channels or meta data.

Example 8.1.1. (Recommendation calculation 2) [54]
For this example we will use the following input:

$$S = \begin{pmatrix} 10 & 7 & 9 & 6 \\ 7 & 10 & 10 & 5 \\ 9 & 10 & 10 & 4 \\ 6 & 5 & 4 & 10 \end{pmatrix}, \vec{p} = (2, 3).$$

The calculation requires the following steps:

1. The user will encrypt $\vec{p}$.
2. The service provider computes the weighted sum on encrypted $\vec{p}$: $(2 \cdot 10, 2 \cdot 7, 2 \cdot 9, 2 \cdot 6) + (3 \cdot 7, 3 \cdot 10, 3 \cdot 10, 3 \cdot 5) = (20 + 21, 14 + 30, 18 + 30, 12 + 15) = (41, 44, 48, 27)$.
3. The service provider will send this encrypted result back to the user, along with the sum of the similarity scores, $(10+7, 7+10, 9+10, 6+5) = (17, 17, 19, 11)$.
4. The user can now compute the weighted mean after decryption: $\vec{r} = (41/17, 44/17, 48/19, 27/11) \approx (2.41, 2.59, 2.53, 2.45)$.

After removing the already watched movies from the list, the conclusion is that the user should watch movie 3 since it has a higher score than movie 4.

Note that we want to perform the division at the user side since we are working over the integers in this example. For modular arithmetic, division could be implemented using the modular multiplicative inverse. However, in this example we want the real number as a result.

When looking at this calculation from a performance perspective, we can see that the multiplicative depth is low. It involves one multiplication with the weight and a summation of the results. Using the batching and SIMD techniques, we would estimate that it is possible to perform this calculation in a matter of seconds. The calculation can also be precomputed by the

service provider, making this performance overhead less noticeable for the user.

8.2 General scenario

In general, the previously described example involves three parties, the service provider, the cloud provider and the user. The cloud provider facilitates the infrastructure and/or the platform. The service provider facilitates the service to the user, making use of the cloud providers resources. The user wants to benefit from the service provided. This involves a calculation on data that the user cannot do locally. This can have several reasons, e.g., the service includes intellectual property, the user does not have enough computational resources or just to reduce cost and increase convenience. When the service includes intellectual property, we can include an extra property to provide privacy of the function. Which should protect against replication of the service by either the user or the cloud provider. In figure 8.1 we see an overview of this general scenario.

The client-side of this service should facilitate encryption before sending the data to the cloud and decryption upon receiving the result back from the cloud. With traditional encryption this would make it impossible to process this data by the service provider and the cloud would function as just a backup service. HE allows processing of encrypted data and thus allows the cloud service to operate as intended. However, we still rely on traditional encryption for the communication between the cloud and the secure environment. This can be symmetric encryption or hybrid encryption with the necessary authentication and integrity checks to create a secure channel between the secure environment and the cloud service.

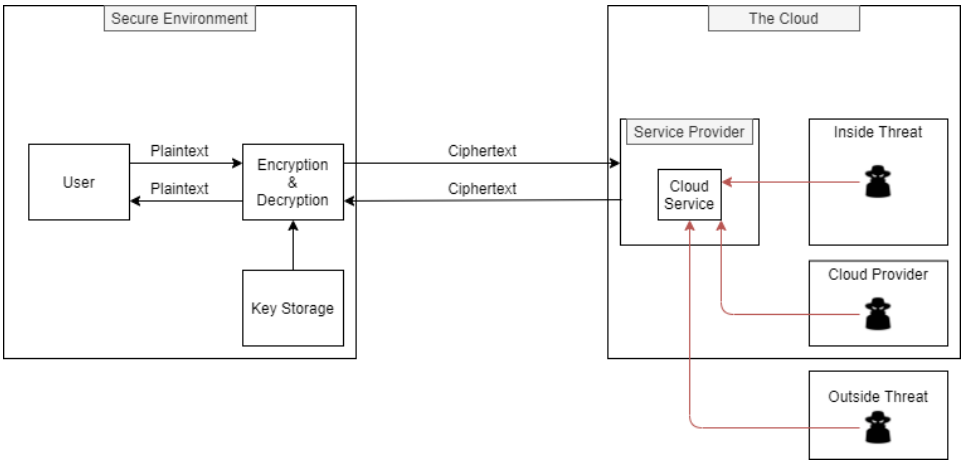


Figure 8.1: The general scenario and threats

8.3 General framework

The described use case, together with the examples from the introduction show some key characteristics. In general we can see the following properties in a use case:

- remote computation;
- partially or fully untrusted infrastructure;
- privacy of input, output and (optionally) function.

The computation involves:

- descriptive statistics, e.g., calculating a summation, an average or a maximum value;
- predictive statistics, e.g., data mining, machine learning or predictive modeling.

While the first type of computation is relatively easy to represent in an homomorphic calculation, the second type is less trivial.

As seen in the general scenario and figure 8.1, we can create more privacy for users by using HE in specific calculations that involve sensitive data. This would still allow service providers to operate and benefit from the cloud without compromising the privacy of users and possibly compromising sensitive data to the service provider, cloud provider or attackers of the cloud. Although all processing of this data that happens in the cloud is legally allowed through service level agreements, this would still help with minimizing the amount of data that the service and cloud providers have to acquire and can potentially compromise.

8.4 Potential usecases

In the literature we have seen a lot of potential use cases that also contain most of the key characteristics described in the general framework [55]. These cases not always reflect on the exact combination of HE and the cloud but we can clearly see some benefit for a form of secure computation that could also work in a cloud environment. We describe two more potential usecases:

- The prediction of the likelihood of certain health issues can be seen as a valuable resource to healthcare. If a research institute managed to create such an algorithm to diagnose patients, they want to provide this to as many hospitals as possible. However, they do not want to reveal this algorithm to the hospital in order to earn money to fund

new research. The research institute can choose to provide this algorithm in the form of a cloud service. Directly using plaintext patient data in their algorithm in the cloud can lead to privacy risks. When implementing this cloud service using HE, we can run the algorithm on encrypted patient data and return an encrypted result back to the hospital. This also makes it possible to protect the confidentiality of the function. The usage of this algorithm also depends on the amount of patients that have to be diagnosed, which can be seen as a varying demand influx.

- Descriptive statistics on manufacturing processes, e.g., statistical process control, are very valuable to companies. It allows them to control and improve their processes. It often involves control limits that specify within which boundaries a specific test should be. These control limits can be valuable to a competitor that runs a similar manufacturing process. This comparison requires the encryption of the upper and lower control limit. We then require two secure integer comparisons. One comparing the test value to the upper limit and one to the lower limit. This result can be combined with an AND gate. The result will be 1 if it is between both limits and 0 if it is not. In this case we protect the confidentiality of the upper and lower control limit while comparing them to test data that resides in the cloud.

This chapter shows that HE can work as a privacy-enhancing technology by providing confidentiality throughout the entire data journey, from within the secure environment of the user, through the service and cloud provider, throughout the actual computation and eventually back into the secure environment.

Chapter 9

Proof of concept

This chapter will explain the reasoning behind the design, architecture and technical choices. It will include several technical details that are needed to understand the proof of concept. Several suggestions are mentioned that can be used to further improve upon the concept.

9.1 Design

When developing a proof of concept it is important to keep in mind that we want to show that a concept is feasible. For HE it is not possible to show the input and output plaintext of a computation on the cloud platform, since the entire concept is based on local storage of the private keys and user-control over these keys. The cloud is not allowed to see the plaintext. Therefore we decided to show the ciphertexts and their manipulations and explain this as best as possible. In the design we have three main segments. The data, the calculation and the result. The data tab allows the user to upload a ciphertext and see the description and context of this ciphertext. The calculation tab allows the user to create a calculation, in which you select the ciphertexts and the operator. The calculations that are created can be selected and executed. The results tab allows the user to see the description and context of the calculation they have performed and supports downloading the result.

The decryption of the ciphertext can not be shown in the cloud due to the nature of the application. However, this is part of the total demo and also is necessary to show the correctness of the computation performed in the cloud. The only way in which the homomorphic evaluation can be shown in the cloud demo is to disable the encryption and only show the homomorphic addition and multiplication functionality on plaintext. However, this does not represent the concept as well as local encryption and decryption.

9.1.1 Minimum viable product

In essence every application that is based on HE follows the same structure. To start we have to encode and encrypt our data in such a way that the evaluating party can process the data accordingly. Therefore, the first step is always: encrypt the data. The second step is to retrieve the data for evaluation. After evaluation we get a new ciphertext which is an encryption of the result. This ciphertext has to be stored. The final step involves the decryption of the data and hence retrieving the correct result. For the PoC these three steps are structured in the following way.

1. Locally encode and encrypt the data and upload the ciphertexts.
2. Evaluate an addition or multiplication on two ciphertexts and store the result in a database.
3. Download encrypted result and locally decrypt and decode the result.

This is the minimum functionality that the PoC has to support. However, in more advanced applications based on HE, the decode, decrypt and upload functionality should be mainly automated to enable processing of larger datasets through cloud connectors. The evaluation functionality should be extended to combine multiple ciphertexts and allow for more advanced computation. The third step can also be combined with other techniques such that only the people that are entitled to it get the correct result without manual downloads and decryption.

9.1.2 Open-source libraries

In order to create a PoC on the cloud platform, we need a library that supports HE. There are plenty of options, with each their strengths and weaknesses depending on the applications. We will highlight the main three libraries that were considered for this applications.

- Microsoft SEAL, which provides a set of encryption schemes fit for homomorphic evaluation of data. The supported algorithms include several variants of BFV and CKSS. These schemes already include many optimizations [56].
- PALISADE, which has been under development by engineers and scientists from defense contractors, start-ups and universities, supported by DARPA, IARPA, Sloan Foundation and others. It provides a set of lattice-based encryption schemes, not limited to homomorphic encryption. They also support traditional encryption and hashing based on lattices. An advantage of using this library is that it supports multiple cryptographic building blocks that can be used to secure the application [57].

- HElib, which has been under development with help from IBM since 2009. The library supports the BGV and the CKKS scheme. In both schemes a number of optimizations have been implemented. HElib also supports "assembly language for HE" which means that it is possible to manipulate data on the lowest level [58].

These libraries are available as C++ codebases. The SAP Cloud Platform does support C++, however for Node.js there exists a convenient buildpack. This makes it easy to deploy a Node.js backend on the cloud platform. Therefore we chose the Node.js implementation of the Microsoft SEAL library. This library has good documentation and is fit for the purpose of the proof of concept. Node-seal npm package brings SEAL to Javascript through Web Assembly called by wrappers that invoke the C++ code. It can also be easily installed with the Node package manager (npm). According to benchmarks from the node-seal library, the web implementation will be 6 times slower for addition and 4 times slower for multiplication[59]. Although this implementation will be slower than the C++ version, the PoC is not purely focused on showing the speed. A modular architecture will allow us to switch out the backend for the C++ equivalent at any time.

Standardization

In addition the development of several open-source libraries, there is also a standardization effort by several developers from the academic world in combination with major companies such as Google, Microsoft and IBM[60]. In these standards the security, API and applications of homomorphic encryption are defined and discussed. The goal is to create a knowledge base for researchers and companies alike. This can be compared to the Cryptographic Standards and Guidelines provided by NIST. We think this is an important part in the secure and efficient usage of homomorphic encryption in applications.

Microsoft SEAL

The choice for SEAL has been made based on several key aspects. The first aspect is the clear documentation and the ease of use. This is something that they are actively working on to improve and this keeps making it easier for programmers without much of a cryptographic background. In short, the library supports two types of schemes, the Brakerski Fan Vercauteren(BFV) and the Cheon Kim Kim Song (CKKS) scheme. The BFV scheme is mainly used when the data can be represented by integers, the CKKS scheme supports real numbers (floating points). In addition to the schemes itself there are also several batching and encoding techniques that will be relevant for this proof of concept. The final consideration is the speed of the algorithm. SEAL has implemented most of the optimizations that were introduced in

the last couple of years which will result in higher performance and speed in the cloud.

9.1.3 SAP Cloud Platform

As discussed in Chapter 3 the SAP Cloud Platform supports application development through a set of services, tools and standard applications. Since the goal is to create an application for the cloud, it might be convenient to also develop it in the cloud. In the design we concluded that we need a frontend to show the ciphertexts and a backend that can support the chosen library and store the results. Therefore we chose to use all the functionality of the full-stack Web Integrated Development Environment. The IDE includes the option to use templates for Multi-target applications. In this MTA we can add a lot of different services such as a SAP Hana Cloud database, a Node.js backend server and a frontend server/Web UI.

SAP Fiori

There are several ways to create and structure the frontend. SAP Fiori Cloud gives developers the tools and guidelines to create a consistent user experience. This methodology allows the creation of uniform applications across the cloud. The client UI technology with which this is made possible is called SAPUI5, an HTML-5 based development toolkit. To help clarify the PoC we want a consistent and clear design, we think this is best achieved with SAPUI5.

MTA model

As discussed earlier we want a modular architecture such that we can swap several modules e.g. Node.js backend for a C++ backend. The cloud platform provides us with a multi-target application model. In this model we can link several modules together, sharing services such as the Hana Cloud database or the UAA (user, account and authentication service). This means the frontend and backend can make use of the same authentication provider and we can easily swap modules.

9.2 Implementation

In Appendix B.1, we show the file structure and in Appendix B.2 we show the user interface. The application code can be split into two parts, the frontend and the backend. We used the pre-generated model, view controller structure in the frontend [61]. For the backend we used a Node.js webserver, combined with a SAP HANA Cloud database [62].

9.2.1 Frontend

The model contains all the models for the data and handles the application data. This is separated from the view, which only shows the data through the UI. The UI has been specified by a home view and three fragments. In the fragments we specify all the elements such as the predefined table module or the wizard screen. The buttons in each fragment contain an event. The controller modifies the view and model when called through these events. The controller also contains functions that are used to communicate with the webserver. We used Ajax to send data and call the services of the backend [63].

There are also some general settings that are used for the routing within the webapp, these can be found in the approuter. This is the single point of entry which uses the xsuaa authentication service. This means you need to have a user account on the cloud platform to reach the application. From this point we can also redirect to the service backend.

9.2.2 Backend

The backend contains a database and several different services, both linked to the frontend using the multi-target application structure and configuration files. All the services require an authenticated user and a csrf-token. To upload the files, xsjs is used. The xsOdata service provides the tables in the frontend with the correct values to display. The final services the backend provides is the HE computation. They are provided through several functions supported by the Node.js server and have been split in two parts:

- createCalculation, creates the calculation as specified in the wizard;
- executeCalculation, retrieves the specifications of the calculation using the CalculationID. Gathers the input to the function by looking at both the CipherTextID's. It then executes the computation using the correct operation and input data. The result will be stored in a new entry in the database.

To communicate with the database, we use calls to the stored procedures. Stored procedures are pre-defined queries that can be called by the backend. This means it prevents wrong insert statements and custom SQL queries, which in the worst case, can lead to SQL injection. There are three procedures:

- dataUploader, used to insert ciphertext data after uploading;
- calculationUploader, used to insert a calculation specified by the user;
- resultUploader, used to insert the result of the computation.

Database

The database used is the SAP HANA Cloud database, it is mainly used as a database service. It provides real-time data access and analysis. However, we only use it to store and retrieve our ciphertexts, calculations and results. To specify the database tables there is a clear distinction between design-time and run-time. In the design-time we specified all the tables, procedures and synonyms. After deployment we can use the real-time container to store and retrieve our data. To generate the three unique ID's we use sequences, these will automatically increment the ID for each new table entry. In the database we have three tables:

- Project-H.database.data::Data;
- Project-H.database.data::Calculations;
- Project-H.database.data::Results.

To link design and run-time we used the following synonyms:

- Project-H.database.data::localData;
- Project-H.database.data::localCalculations;
- Project-H.database.data::localResults.

Client-side code

On the client-side we also use a Node.js server with the Microsoft SEAL library. It is used to encrypt and decrypt the data on the client-side. It can also provide us with some additional tests before we move the code into the cloud. The private key used to encrypt remains offline at the client and should be stored securely. The context of the encryption, together with the ciphertext, can be written to a file. This file can be uploaded in the application. The result can be downloaded and contains the same context and the new ciphertext with the result. This can be used to decrypt and retrieve the result.

Chapter 10

Related work

This chapter will take a look at other existing implementations and recent events in the area of secure (cloud) computation.

With the recent developments in algorithms and implementations, the performance overhead becomes less of a problem for several interesting applications. In the last couple of years, both in the literature as well as in the industry, applications that use HE are starting to arise. In this chapter we will highlight some of the applications that were not mentioned previously.

- CryptoNets [64], in this publication the researchers present a method to convert learned neural networks that operate on plaintext data, into neural networks that operate on ciphertext. They call it a CryptoNet. This can be used on, e.g., financial or medical data sets, to make privacy preserving predictions.
- Privacy-preserving electronic toll pricing [65], this paper shows how to create a protocol for electronic toll pricing in which a minimal amount of location data will be revealed. HE is used in a part of the protocol to compute the total tax that has to be payed. Since each individual component directly relates to the amount of distance traveled, and thus also the location, we want them to remain confidential.
- Privacy-preserving yet accountable ride-hailing service [66], this paper proposes ORide (Oblivious Ride). This system solves the location privacy concerns that arise in services such as Uber or Lyft. It can match customers and drivers without knowing their identities or location. It uses a version of the BFV scheme to compute which driver is the closest.

Next we will describe two platforms that provide a secure computation service.

- ShareMind, a database and application server that can process encrypted data. One of the techniques they use to accomplish this is HE. It can provide end-to-end data protection and accountability. A more concrete example is an implementation made using the ShareMind framework to prevent satellite collisions [67]. They use encrypted trajectory data to securely compare and calculate whether or not a collision will happen. This will improve orbital safety without sharing confidential satellite information, e.g., in military satellites.

- Duality Technologies, purely focuses on HE as their main technique to facilitate secure data science for regulated industries. Their platform is called SecurePlus which protects data and the models used for the analyses. An example of an application of their platform is the sharing and analysis of data by financial institutions [68]. This allows them to leverage their collective data set to detect financial crime.

Chapter 11

Conclusions

In this thesis, we researched the use of secure computation techniques in a cloud environment. At the start of this work we described the most important features, benefits and obstacles of the cloud and of secure computation.

While the public cloud is indeed an online and public space, the cloud also offers more private solutions in the form of a community or private cloud. This provides more separation between services which can prove to be useful in decreasing the chance of certain attacks. However, a private cloud is not the only protection we need against attacks.

Since the user has no direct control over the security measures at the cloud and service provider, this often revolves around trust. Although the user or service provider might not be legally responsible for the data anymore, one can imagine they want to secure the data as best they can. Secure computation can provide this functionality without removing the benefits that the cloud provides.

The main techniques used in secure computation are secure multi-party computation, secure hardware and homomorphic encryption. While each have their strengths and weaknesses, homomorphic encryption has been highlighted as the most interesting in a cloud environment due to the way it can outsource a computation to an untrusted environment, while preserving confidentiality.

When looking at the top 11 threats, we can see that they all threaten the confidentiality of data and that HE can achieve better confidentiality guarantees in certain situations. By shifting the encryption and decryption process from the cloud to the secure environment, we can see that a lot of threats can now only compromise ciphertext rather than plaintext, especially when data is used within cloud services. Besides the threats to the cloud itself, the service and cloud providers themselves also have access to user data and can compromise confidentiality. This is where we see a usecase for secure computation and we hope it can improve the cloud security.

The usecase we described not only illustrates why Netflix wants to use the cloud, but also that they perform computations on sensitive data inside the cloud. As an example we showed that their recommendation system can be made privacy-preserving by replacing the specific calculations on sensitive data with calculations on encrypted data, i.e., the calculation of the weighted sum. This would protect the user preference against the service provider, cloud provider and some of the described threats to the cloud itself. This example also allowed us to derive some key characteristics. We concluded that it would make most sense to use HE when outsourcing a computation on highly sensitive data to a partially or fully untrusted environment, replacing plaintext computation rather than the traditional cryptography. This computation often involves mathematical and statistical functions but can also be more complex as seen in, e.g., machine learning.

This means that the use cases are very specific and also depend on the performance of the different HE schemes. The performance of HE varies between each different type. We will give a short summary of the different HE classifications:

- Partial homomorphic encryption supports addition or multiplication an unlimited amount of times.
- Fully homomorphic encryption supports addition and multiplication an unlimited amount of times, through bootstrapping a somewhat homomorphic encryption scheme.
- Somewhat homomorphic encryption supports addition and multiplication a limited amount of times.
- Leveled homomorphic encryption supports additions and multiplication but limits the multiplicative depth.

While somewhat homomorphic encryption is limited by the amount of noise, leveled homomorphic encryption is limited on purpose. This means the parameters have been chosen to support only a specific circuit depth to optimize performance.

Additional ways to increase performance come from the improvements made to the two aspects that make a scheme fully homomorphic, the bootstrapping procedure and the SHE scheme. Improvements to the SHE scheme include reducing ciphertext and parameter size (with the same level of security), ciphertext packing and speedups from classical computing. One of the best performing schemes is the BFV scheme, which has been used in the proof of concept.

The proof of concept shows an implementation of a simple application in the cloud that can receive ciphertext data and a specific calculation. When executing this calculation the specific operator (addition or multiplication)

is applied to a pair of ciphertexts. All the numbers packed inside the ciphertexts are combined into one new value, by evaluating them pairwise. The new resulting ciphertext includes the encrypted results of this pairwise evaluation, without revealing either the input or the output.

Although the proof of concept can perform calculations, it also shows some of the weaknesses in HE. We conclude that the confidentiality of the data can indeed be guaranteed by secure HE schemes. However, other wanted properties, such as integrity, availability and verifiability have to be guaranteed through additional measures. Verifiability is especially interesting for HE since it can give us guarantees about the function that has been evaluated. When looking at HE as a primitive we can also see that CCA security conflicts with its malleability. This changes our assumption on implementing "just" an encryption into the conclusion that HE can only work in good security architectures including additional controls, key management and when taking malleability into account that is inherent to HE. The security requirements and limitations in combination with the performance overhead show us why many people see homomorphic encryption as a step backwards. However, recent developments show us that if secure computation can be implemented correctly and efficiently, we can unlock value from data that was previously inaccessible, i.e., in cases in which we need to achieve confidentiality of input, output or function. This makes us conclude that it is a good privacy-enhancing technology and that it certainly has a future in many applications and protocols that want to include privacy by design and truly ensure privacy end-to-end in a cloud environment.

11.1 Future work

In the last couple of years a lot of progress has been made both on creating faster algorithms and also in developing secure computation platforms. We will present some future work for the thesis, and some improvements that can be made on the proof of concept.

- In this work we often considered the case in which one party has the encryption/decryption key. In a decentralized and real-life setting this may not always be the case. Multi-Key Homomorphic Encryption schemes allow multiple participants to decrypt. Other options are Treshold FHE schemes or proxy re-encryption techniques.

- The privacy of function is not something we can always guarantee, however, this property can be wanted in, e.g., the protection of machine learning algorithms. It would be interesting to research how one could steal or copy models and how you can protect against this and if HE can protect against it.

- To optimize the performance of implementations with HE, we need to carefully choose the parameters and also have great knowledge about the underlying scheme. It also means we have to translate our higher level functions into HE circuits (binary or arithmetic). How can we best achieve this?
- Serverless computing or functions as a service, as seen in AWS Lambda, is a relatively new execution model for the cloud. Is it possible to use HE in this setting?
- In the proof of concept, one could have a closer look at the application security, look for ways to include integrity and verifiability and increase stability and performance (by swapping to a C++ backend).

Appendix A

A.1 On data banks

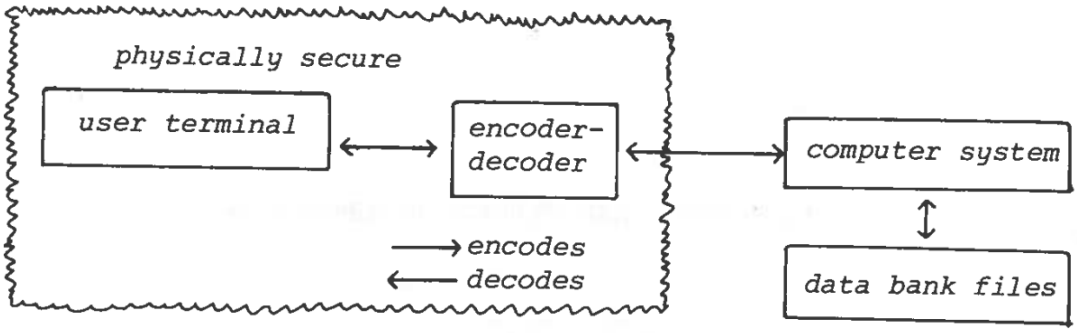


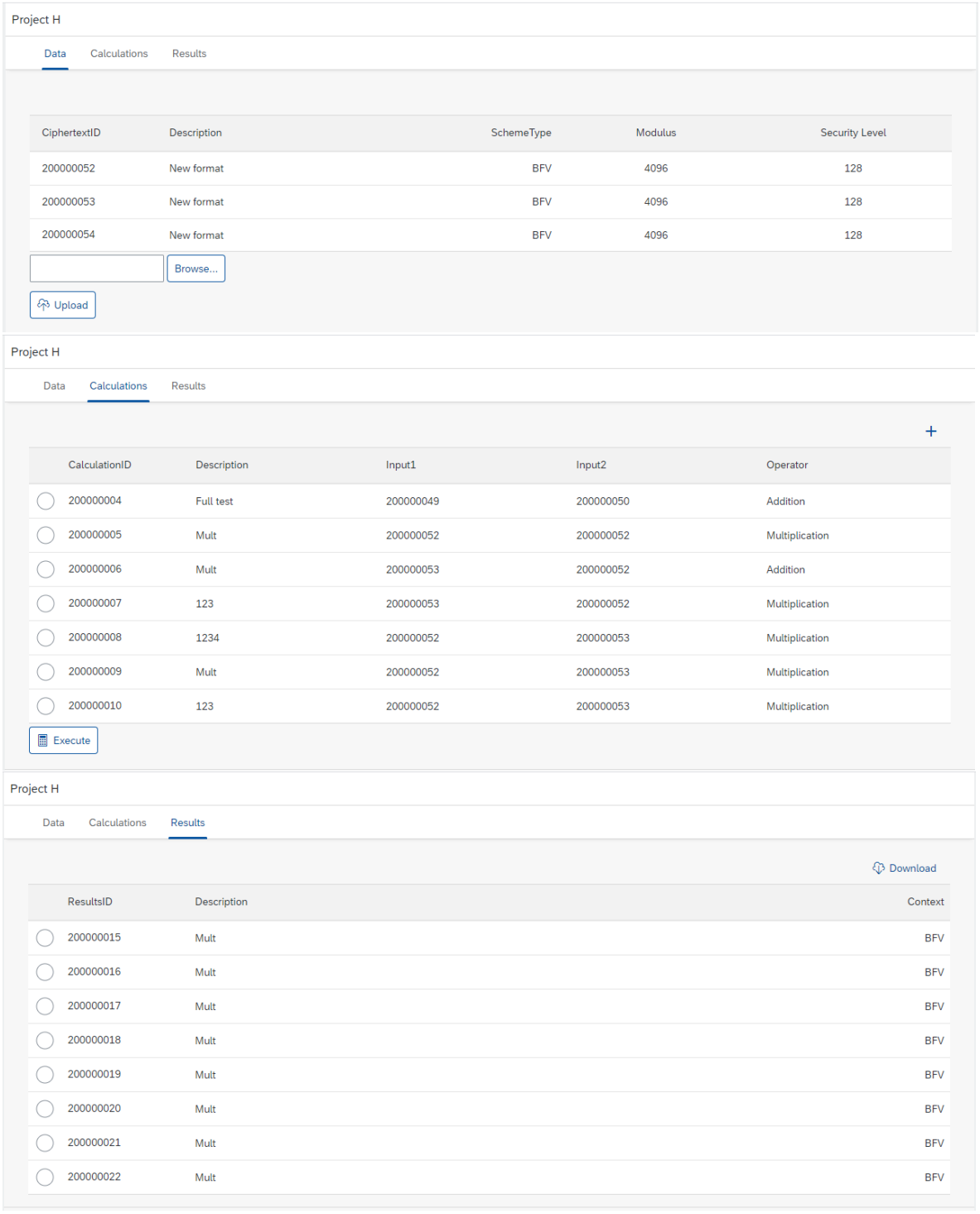
Figure 1

Appendix B

B.1 File structure

```
Workspace
|- Project-H
|    |- database
|    |    |- src
|    |    |    |- data          -- Contains design-time objects
|    |    |- package.json
|    |- frontend
|    |    |- webapp
|    |    |    |- controller    -- Controls model and view
|    |    |    |- model         -- Retrieves and contains data
|    |    |    |- view          -- Defining and rendering the UI
|    |    |    |- index.html
|    |    |- neo-app.json       -- Environment specifications
|    |    |- package.json       -- Build specifications
|    |    |- ui5.yaml           -- Frontend specifications
|    |    |- xs-app.json        -- Defines the routing
|    |- service
|    |    |- HE_computation     -- Homomorphic calculation logic
|    |    |- lib                -- Xsjs/xsodata services
|    |    |- node_modules       -- Imports and packages
|    |    |- test               -- Tests
|    |    |- server.js          -- Start-up script
|    |    |- testrun.js         -- Test start-up script
|    |- mta.yaml                -- Multi-target application configuration
|    |- xs-security.json        -- Security configuration
```

B.2 Proof of concept



Bibliography

[1] M. Russinovich, "Advancing microsoft teams on azure—operating at pandemic scale." `https://azure.microsoft.com/en-us/blog/advancing-microsoft-teams-on-azure-operating-at-pandemic-scale/`, June 2020.

[2] S. Zhang, J. Ford, and F. Makedon, "A privacy-preserving collaborative filtering scheme with two-way communication," vol. 2006, pp. 316–323, 01 2006.

[3] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, p. 120–126, Feb. 1978.

[4] T. El Gamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in *Proceedings of CRYPTO 84 on Advances in Cryptology*, (New York, NY, USA), pp. 10–18, Springer-Verlag New York, Inc., 1985.

[5] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pp. 124–134, 1994.

[6] M. Ajtai, "Generating hard instances of lattice problems (extended abstract)," in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, (New York, NY, USA), p. 99–108, Association for Computing Machinery, 1996.

[7] Y.-K. Liu, V. Lyubashevsky, and D. Micciancio, "On bounded distance decoding for general lattices," pp. 450–461, 01 2006.

[8] O. Regev, "The learning with errors problem (invited survey)," in *Proceedings of the 2010 IEEE 25th Annual Conference on Computational Complexity*, CCC '10, (USA), p. 191–204, IEEE Computer Society, 2010.

[9] J. Katz and Y. Lindell, *Introduction to Modern Cryptography, Second Edition*. Chapman and Hall/CRC, 2nd ed., 2014.

[10] P. Mell and T. Grance, "The nist definition of cloud computing," September 2011.

[11] SAP Help Portal, "Sap cloud platform." https://help.sap.com/viewer/product/CP/Cloud/en-US?task=discover_task, 2020.

[12] Cloud Foundry Foundation. https://www.cloudfoundry.org. Open Source Cloud Application Platform.

[13] SAP Help Portal, "Sap cloud platform cockpit." https://help.sap.com/viewer/ea72206b834e4ace9cd834feed6c0e09/Cloud/en-US/19d7119265474dd18ec16fad2a0b28c1.html. SAP Help Portal.

[14] R. Cramer, R. Gennaro, and B. Schoenmakers, "A secure and optimally efficient multi-authority election scheme," vol. 8, 10 2000.

[15] R. Cramer, I. Damgård, D. Catalano, G. Crescenzo, I. Darmgård, D. Pointcheval, and T. Takagi, *Multiparty Computation, an Introduction*, pp. 41–87. 03 2006.

[16] R. L. Rivest, L. Adleman, and M. L. Dertouzos, "On data banks and privacy homomorphisms," *Foundations of Secure Computation, Academia Press*, pp. 169–179, 1978.

[17] D. Beaver, "Foundations of secure interactive computing," in *Advances in Cryptology — CRYPTO '91* (J. Feigenbaum, ed.), (Berlin, Heidelberg), pp. 377–391, Springer Berlin Heidelberg, 1992.

[18] A. C. Yao, "Protocols for secure computations," in *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science*, (Washington, DC, USA), pp. 160–164, IEEE Computer Society, 1982.

[19] M. Rabin, "How to exchange secrets with oblivious transfer," *IACR Cryptol. ePrint Arch.*, vol. 2005, p. 187, 2005.

[20] S. Yakoubov, "A gentle introduction to yao ' s garbled circuits," 2017.

[21] 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, NY, USA), p. 218–229, Association for Computing Machinery, 1987.

[22] I. Damgård and T. Toft, "Trading sugar beet quotas - secure multiparty computation in practice.," *ERCIM News*, vol. 2008, 01 2008.

[23] M. Schunter, "Intel software guard extensions: Introduction and open research challenges," in *Proceedings of the 2016 ACM Workshop on*

Software PROtection, SPRO '16, (New York, NY, USA), p. 1, Association for Computing Machinery, 2016.

[24] D. Harnik, "Impressions of intel sgx performance." `https://medium.com/@danny_harnik/impressions-of-intel-sgx-performance-22442093595a`, Dec. 2017.

[25] F.-X. Standaert, *Introduction to Side-Channel Attacks*, pp. 27–42. 12 2010.

[26] A. Barenghi, L. Breveglieri, I. Koren, and D. Naccache, "Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures," *Proceedings of the IEEE*, vol. 100, pp. 3056–3076, Nov 2012.

[27] J. V. Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution," in *27th USENIX Security Symposium (USENIX Security 18)*, (Baltimore, MD), p. 991–1008, USENIX Association, Aug. 2018.

[28] S. Mavrovouniotis and M. Ganley, "Hardware security modules," *Secure Smart Embedded Devices, Platforms and Applications*, pp. 383–405, 06 2013.

[29] Cloud Security Alliance, "Top threats to cloud computing, the egregious 11." `https://cloudsecurityalliance.org/`, June 2019.

[30] A. Shostack, "The threats to our products." `https://www.microsoft.com/security/blog/2009/08/27/the-threats-to-our-products/`.

[31] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds," in *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, (New York, NY, USA), p. 199–212, Association for Computing Machinery, 2009.

[32] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, p. 120–126, Feb. 1978.

[33] S. Goldwasser and S. Micali, "Probabilistic encryption," *Journal of Computer and System Sciences*, vol. 28, no. 2, pp. 270 – 299, 1984.

[34] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'99, (Berlin, Heidelberg), p. 223–238, Springer-Verlag, 1999.

[35] D. Boneh, E. Goh, and K. Nissim, "Evaluating 2- dnf formulas on ciphertexts"," *Proceedings of the 2Nd Conference on Theory of Cryptography*, pp. 325–342, 01 2005.

[36] C. Gentry, *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford University, Stanford, CA, USA, 2009.

[37] Z. Brakerski and V. Vaikuntanathan, "Efficient fully homomorphic encryption from (standard) lwe." Cryptology ePrint Archive, Report 2011/344, 2011. `https://eprint.iacr.org/2011/344`.

[38] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, (New York, NY, USA), p. 309–325, Association for Computing Machinery, 2012.

[39] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical gapsvp," in *Proceedings of the 32nd Annual Cryptology Conference on Advances in Cryptology — CRYPTO 2012 - Volume 7417*, (Berlin, Heidelberg), p. 868–886, Springer-Verlag, 2012.

[40] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption." Cryptology ePrint Archive, Report 2012/144, 2012. `https://eprint.iacr.org/2012/144`.

[41] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," in *Advances in Cryptology – CRYPTO 2013* (R. Canetti and J. A. Garay, eds.), (Berlin, Heidelberg), pp. 75–92, Springer Berlin Heidelberg, 2013.

[42] J. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," pp. 409–437, 11 2017.

[43] J. von zur Gathen and G. Seroussi, "Boolean circuits versus arithmetic circuits," *Information and Computation*, vol. 91, no. 1, pp. 142 – 154, 1991.

[44] D. Cash, M. Green, and S. Hohenberger, "New definitions and separations for circular security," in *Public Key Cryptography – PKC 2012* (M. Fischlin, J. Buchmann, and M. Manulis, eds.), (Berlin, Heidelberg), pp. 540–557, Springer Berlin Heidelberg, 2012.

[45] Z. Brakerski, *Cryptographic Methods for the Clouds*. PhD thesis, the Weizmann Institute of Science, Rehovot, Israel, 2011.

[46] N. Smart and F. Vercauteren, "Fully homomorphic simd operations," *IACR Cryptology ePrint Archive*, vol. 2011, p. 133, 01 2011.

[47] S. Halevi, Y. Polyakov, and V. Shoup, "An improved rns variant of the bfv homomorphic encryption scheme," in *Topics in Cryptology – CT-RSA 2019 - The Cryptographers' Track at the RSA Conference 2019, Proceedings* (M. Matsui, ed.), Jan. 2019.

[48] C. Gentry and S. Halevi, "Implementing gentry's fully-homomorphic encryption scheme," in *Proceedings of the 30th Annual International Conference on Theory and Applications of Cryptographic Techniques: Advances in Cryptology*, EUROCRYPT'11, (Berlin, Heidelberg), p. 129–148, Springer-Verlag, 2011.

[49] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds." Cryptology ePrint Archive, Report 2016/870, 2016. `https://eprint.iacr.org/2016/870`.

[50] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation.," in *IEEE Symposium on Security and Privacy*, pp. 238–252, IEEE Computer Society, 2013.

[51] M. Chenal and Q. Tang, "On key recovery attacks against existing somewhat homomorphic encryption schemes." Cryptology ePrint Archive, Report 2014/535, 2014. `https://eprint.iacr.org/2014/535`.

[52] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings." Cryptology ePrint Archive, Report 2012/230, 2012. `https://eprint.iacr.org/2012/230`.

[53] N. Ramakrishnan, B. Keller, B. Mirza, A. Grama, and G. Karypis, "Privacy risks in recommender systems," *IEEE Internet Computing*, vol. 5, pp. 54–62, Nov. 2001.

[54] E. Zekeriya, M. Beye, T. Veugen, and R. Lagendijk, "Privacy-preserving content-based recommendations through homomorphic encryption," 01 2012.

[55] D. Archer, L. Chen, J. H. Cheon, R. Gilad-Bachrach, R. A. Hallman, Z. Huang, X. Jiang, R. Kumaresan, B. A. Malin, H. Sofia, Y. Song, and S. Wang, "Applications of homomorphic encryption," tech. rep., HomomorphicEncryption.org, Redmond WA, USA, July 2017.

[56] "Microsoft SEAL (release 3.5)." `https://github.com/Microsoft/SEAL`, Apr. 2020. Microsoft Research, Redmond, WA.

[57] "PALISADE Lattice Cryptography Library (release 1.9.2)." https://palisade-crypto.org/, Apr. 2020.

[58] "HElib (release 1.0.1)." https://github.com/homenc/HElib, Apr. 2020.

[59] Morfix IO, "node-seal." https://github.com/morfix-io/node-seal, 2020.

[60] An Open Industry, Government, Academic Consortium, "Homomorphic encryption standardization." https://homomorphicencryption.org/, Nov. 2018.

[61] SAP SE, "Documentation." https://sapui5.hana.ondemand.com/#/topic.

[62] OpenJS Foundation, "Nodejs documentation." https://nodejs.org/api/.

[63] E. Woychowsky, *AJAX: Creating Web Pages with Asynchronous JavaScript and XML (Bruce Perens' Open Source Series)*. USA: Prentice Hall PTR, 2006.

[64] N. Dowlin, R. Gilad-Bachrach, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," Tech. Rep. MSR-TR-2016-3, February 2016.

[65] J. Balasch, A. Rial, C. Troncoso, B. Preneel, I. Verbauwhede, and C. Geuens, "Pretp: Privacy-preserving electronic toll pricing," pp. 63–78, 11 2010.

[66] A. Pham, I. Dacosta, G. Endignoux, J. Troncoso-Pastoriza, K. Huguenin, and J. Hubaux, "Oride: A privacy-preserving yet accountable ride-hailing service," in *USENIX Security Symposium*, 2017.

[67] L. Kamm and J. Willemson, "Secure floating-point arithmetic and private satellite collision analysis." Cryptology ePrint Archive, Report 2013/850, 2013. https://eprint.iacr.org/2013/850.

[68] A. Kaufman, "Privacy-preserving information sharing against financial crime." https://dualitytech.com/privacy-protected-information-sharing-against-financial-crime/.