IT Security Master Program
Department of Economic Informatics and Cybernetics
Faculty of Cybernetics, Statistics and Economic Informatics
Bucharest University of Economic Studies

Dissertation Thesis

Coordinator,
University Lecturer Andrei Toma

Master Program Student,
Andrei-Marius Tecșor

Bucharest
2023

IT Security Master Program
Department of Economic Informatics and Cybernetics
Faculty of Cybernetics, Statistics and Economic Informatics
Bucharest University of Economic Studies

Secure Data Processing in Interbank Synchronisation

Dissertation Thesis

Coordinator,
University Lecturer Andrei Toma

Master Program Student,
Andrei-Marius Tecșor

Bucharest
2023

Statement regarding the originality of the content

I hereby declare that the results presented in this paper are entirely the result of my own creation unless reference is made to the results of other authors. I confirm that any material used from other sources (magazines, books, articles, and Internet sites) is clearly referenced in the paper and is indicated in the bibliographic reference list.

Table of Contents

Introduction

The world of film production has always been a challenging and competitive industry. Financing a movie happens to be a hurdle for many filmmakers. Over the years, there have been some famous stories about movie producers that have taken unusual steps to fund their projects, like Robert Townsend. In 1987, Townsend charged $100.000 on his fifteen personal credit cards to finance the production of “Hollywood Shuffle”. [1]

While successful financing efforts are usually highlighted, the stories of less successful attempts are often overlooked. What happened to the film producers who aspired to finance their movies in the same way, but ended up declaring bankruptcy? What about the damages they had done to the banks or others NBFIs (Non-Banking Financial Institution). What about the affected clients? How could these incidents have been foreseen?

The technical solution described in this thesis is inspired by the questions previously mentioned, which draw attention to the security implications of certain practices. I successfully engineered a peer-to-peer system, specifically designed for financial institutions. The cornerstone of this system is the use of Fully Homomorphic Encryption (FHE), an advanced encryption technique that allows the processing of encrypted data without the need for decryption, thus enhancing data security and privacy.

The solution I’m proposing is not aiming to replace the current communication methods among financial institutions, but rather to enhance their security measures, introducing an innovative approach to safe data processing and synchronization. Implementing a system that leverages an unique, confidential framework for data processing can boost competitiveness and underscore the commitment to providing high-quality services, while also deterring any fraudulent activities.

Utilising Fully Homomorphic Encryption will not just facilitate the creation of diverse statistics and metrics based on sensitive data, but also pave the way for fresh dialogues about how this technology could be beneficial to financial systems and beyond.

Effective risk assessment is essential for financial institutions to ensure the safety and soundness of their operations. To this end, we need a formula that can accurately determine an individual's or business's financial risk based on history with other organisations, before they are granted access to new banking services such as loans. Nevertheless, the question remains: should the requester or the provider of the data be responsible for calculating the formula that determines the financial risk level?

This question raises important issues related to data privacy, competition, and processing resources. To address these challenges, I propose the use of Fully Homomorphic Encryption, which provides a viable solution to these impediments. The increase in processing power of our devices has made FHE more accessible, various high-level or low-level frameworks becoming available and making the complex mathematics behind this encryption less of an obstacle.

In this paper, I present a minimum viable product (MVP) that utilises FHE to determine the degree of risk to a possible customer. The focus is on the system's architecture and the

technologies used to achieve the desired goal. The risk formula used in the MVP is a proof of concept and has not been subjected to any extensive study.

The proposed solution has the potential to improve the security and soundness of financial institutions and contribute to the overall stability of the financial system, including areas like open banking. Such a system, or the technology itself, opens up new horizons for other fields that could take advantage of the capabilities of FHE.

Chapter 1. Present-day financial data exchange and synchronisation

1.1. Financial Institutions

1.1.1. Banks

Banks, as licensed financial institutions, have a significant role in providing different financial services to individuals, businesses, and governments. These services range from accepting deposits and providing loans, to offering investment and insurance products. Central to the economy, banks serve to facilitate payments, extend credit, and act as intermediaries between those with funds (savers) and those in need of funds (borrowers). [2]

Banks encounter various risks in their operations, including credit risk, liquidity risk, market risk, operational risk, reputational risk, and macroeconomic risk. The effective management of these risks is crucial for a bank's profitability and dictates the amount of capital it needs to hold. Bank capital primarily consists of equity, retained earnings, and subordinated debt. [2]

Furthermore, banks are subject to rigorous regulation and oversight. In most jurisdictions, a central bank or another regulatory authority supervises banks to ensure the stability of the financial system, protect depositors, and maintain trust in the system. [3] [4]

1.1.2. Non-Banking Financial Institutions

Non-banking financial institutions (NBFIs), also referred as non-bank financial companies (NBFCs), are crucial components of the financial system, offering a subset of services similar to those provided by banks, but without holding a banking license. Their roles can vary significantly depending on the specific business model and the jurisdiction in which they operate. [5] [6]

NBFIs often fill gaps in the provision of financial services, complementing traditional banking institutions and expanding access to financial services. They frequently serve customers who may be underserved by traditional banks. For instance, some NBFIs may specialize in providing credit to individuals or businesses with lower credit ratings. [7]

Examples of NBFIs include:

- insurance companies that protect individuals and businesses from various risks;
- money transfer services that allow for easy cross-border transactions;
- payday lenders providing short-term high-interest loans;
- asset management firms overseeing investment funds;
- microloan organizations offering small loans to underserved populations.

Like banks, NBFIs are typically regulated, although the extent and nature of regulation can vary. NBFCs can also pose risks, such as those related to consumer protection and financial stability, which are important areas of focus for regulators. [7]

1.2. Ways of Communication Between Financial Institutions

1.2.1. Regulators

Communication through regulators is an important type of interaction between financial institutions, especially in matters of compliance, risk management, and financial stability.

Regulatory authorities such as central banks, financial conduct authorities, and exchange commissions, among others, maintain the stability and integrity of the financial system.

Regulators often facilitate communication between financial institutions. For example, in a scenario where a bank or a non-banking financial institution (NBFI) has information about a potential risk or fraudulent activity that could impact other institutions or the broader financial system, the regulator could be the channel through which this information is shared. [8]

Financial institutions are typically required to submit various reports to their regulator, including data about their financial condition, risk exposures, and compliance with various rules. The regulator might aggregate this data and share certain information with other institutions or the public, to support the stability and transparency of the financial system. [9]

The interaction through regulators ensures that sensitive information is handled appropriately and that institutions are aware of and comply with their legal and regulatory obligations. It's worth mentioning that the way communication occurs through regulators can significantly differ depending on the local jurisdiction. Therefore, for the most accurate and relevant information, it's best to refer to the rules and guidelines provided by the respective country's regulatory authorities.

1.2.2. Financial Networks and Platforms

Financial networks and platforms provide the infrastructure for various types of financial transactions, ranging from domestic and international fund transfers to securities trading and foreign exchange transactions.

One of the most widely recognized financial networks is the Society for Worldwide Interbank Financial Telecommunication (SWIFT), which provides a network that enables financial institutions worldwide to send and receive information about financial transactions in a secure, standardized, and reliable environment. [10] [11]

Other examples of financial networks and platforms include: automated clearing houses (ACHs) used for domestic interbank transfers, payment card networks like Visa or Mastercard, securities exchanges, as well as the Single Euro Payments Area (SEPA) for facilitating electronic fund transfers within the European Union. More recently, blockchain networks have emerged, which serve as the foundation for cryptocurrencies and decentralized finance platforms. [12] [13] [14]

These financial networks and platforms not only provide the technical infrastructure for conducting transactions, but they also set rules and standards for how transactions should be conducted. They often provide a range of value-added services, such as transaction monitoring for fraud detection, data analytics, and other services. [15] [16] In many ways, these networks and platforms can be seen as the "plumbing" of the global financial system, enabling the flow of funds and financial information between institutions. [17]

Nevertheless, the use of these networks and platforms presents certain issues, including operational risks and cybersecurity risks. For this reason, FIs and regulators pay close attention to the governance, security, and resilience of these ways communication and data exchange.

1.2.3. Third-Party Service Providers

Third-party service providers are essential actors in the financial ecosystem, offering specialized services to financial institutions. These services enhance the functioning and communication within the sector. Here are a few examples based on domain:

Payment Processing - handle transactions between various stakeholders (e.g. Visa, MasterCard, PayU, MobilPay);

Technology Providers - providing specialized software or platforms for banking operations (e.g. IBM, Oracle);

Consultancy Services - offering strategic planning and compliance advice (e.g. Cegeka, KPMG);

Data Providers - aggregating, analysing, and selling financial and economic data (e.g. The Bucharest Stock Exchange);

Credit Bureaus - assessing the creditworthiness of individuals and businesses (e.g. TransUnion, Romanian Bureau of Credit);

Audit Services - independently reviewing an institution's financial practices (e.g. Deloitte, EY, PwC);

Cybersecurity Services - securing financial institutions' data and digital infrastructure (e.g. Bitdefender).

Third-party service providers extend the capabilities of financial institutions, enabling innovation, ensuring regulatory compliance, and providing cost efficiencies. While beneficial, reliance on these services can introduce potential risks, which financial institutions mitigate through rigorous processes, performance reviews, and contingency planning.

1.2.4. Central Banks

A central bank, also known as a reserve bank or monetary authority, is an institution responsible for managing a country's currency, implementing monetary policy, and supervising its commercial banking system. [18]

Unlike commercial banks, central banks have the exclusive authority to expand the monetary base. Additionally, central banks typically possess supervisory and regulatory powers to maintain the stability of member institutions, prevent bank runs, and discourage irresponsible or fraudulent activities within the banking sector. [18]

In the context of communication, central banks guide financial institutions through policy updates and market insights. In addition to their other functions, central banks oversee and maintain essential payment and settlement systems. These systems form the backbone of a country's financial infrastructure, allowing for the transfer of money and financial securities between various parties. More specifically, payment systems handle the transfer of money, while settlement systems ensure that transactions are completed accurately and securely. This includes everything from large-scale interbank transfers to everyday transactions between businesses and consumers. By managing these systems, central banks enable smooth financial transactions and effective communication between banks and other financial institutions. [19]

Examples include the National Bank of Romania, which fulfills these roles locally, and larger institutions such as the European Central Bank and the Federal Reserve, which operate

at a regional and international level. Their activities ensure a transparent and reliable financial system, fostering trust and stability.

1.2.5. Direct communication

Direct communication represents one of the most basic and traditional forms of interaction between financial institutions. This communication can take many forms depending on the specific needs and relationships between the institutions involved. [20]

For example, banks and non-banking financial institutions (NBFIs) might communicate directly with each other to negotiate terms for interbank loans or to collaborate on syndicated loan deals. Similarly, a bank might communicate directly with an insurance company to negotiate terms for insurance coverage for its operations.

Direct communication can also involve sharing of information for due diligence purposes. For instance, when a bank is considering whether to lend to a specific business, it might communicate directly with other financial institutions that have previously lent to that business to gather information.

This type of communication is also employed in financial markets where institutions communicate with each other to execute trades or negotiate contracts.

Chapter 2. An innovative Peer-to-Peer approach using Fully Homomorphic Encryption

2.1. Effects of a new method of direct and secure communication

In the past, financial institutions faced challenges in communicating and expressing concerns about problematic customers. Bureaucracy, lack of tracing, and human error hindered information sharing among financial institutions, making it difficult to prevent fraud or other negative outcomes. Over time, advances in technology and regulations have enabled the implementation of tools and processes that track and monitor their users more effectively.

Despite these improvements, a system capable of offering insights into a customer's past engagements with other financial institutions could further enhance security. Such a system could help prevent the enrollment of problematic consumers and reduce the risk of fraudulent or criminal activity.

Most of this information exchange is carried out indirectly through third-party intermediaries and platforms. Naturally, direct communication between financial institutions does occur, although it's generally reserved for specific and infrequent situations. Public institutions, such as central banks, play a significant role in facilitating this data exchange, given their overarching regulatory function and access to critical banking data.

The various methods of communication among financial institutions, while essential for their operations and collaboration, come with their unique sets of disadvantages. As we proceed, I will highlight the primary drawbacks associated with each of these communication modes previously discussed, all of which my proposed solution aims to address.

1. **Communication through regulators:**
   - ⇒ **Delay** (communication through regulators can be slow due to bureaucratic processes);
   - ⇒ **Limited scope** (the type of information shared through this channel may be limited to regulatory and compliance matters).
2. **Financial Networks and Platforms:**
   - ⇒ **Interoperability** (different platforms may not work well together, limiting the seamless exchange of information);
   - ⇒ **Technical issues** (downtime or glitches in the systems can disrupt communication).
3. **Third-Party Service Providers**:
   - ⇒ **Dependence** (over-reliance on third-party service providers can pose risks if the provider faces issues);
   - ⇒ **Data privacy** (third-party service providers having access to sensitive data can pose privacy concerns).
4. **Central Banks:**
   - ⇒ **One-Way Communication** (communication from central banks often circulates in a one-way manner, without much room for feedback).

5. **Direct Communication**:
    - ⇒ **Compliance risks** (exchanges of sensitive information could lead to inadvertent violations of regulations);
    - ⇒ **Efficiency** (managing direct communication with numerous institutions, usually done through contracts and meetings can be time-consuming and inefficient).

The development of a system capable of processing sensitive data from various financial institutions, without directly accessing the raw data, could yield substantial benefits in areas such as risk assessment. It would also reduce the response time, making the stats about a client available in early stages of business engagement.

This kind of interaction would be a "give-to-get group". To join and request information, you first need to provide information. Taking advance of the Fully Homomorphic Encryption's capabilities, the key elements that make this new communication method unique are the following:

- ♦ The provided data is exclusively in encrypted form, and all the processing is done solely on encrypted data.
- ♦ The processing algorithm is confidential, resulting in different outcomes for each participant despite providing similar types of data.

Initially, you will think that the lack of transparency will be a disadvantage, but as I mentioned before, this solution is not meant to replace the current approach. The relevant data would still be shared with regulatory authorities and needed third parties. A key benefit in this scenario is the potential to develop a processing system that maximizes data utility, which could increase competitiveness. Particularly, this would benefit smaller financial institutions (especially NBFCs) that may not possess the same resources as a larger bank.

Similar to blockchain technology, this data exchange solution might lead to a multitude of complex consequences. Therefore, further research, particularly from a legal perspective, is required.

Before we dive into the system's specifications and implementation, it's vital to first gain an understanding of what Fully Homomorphic Encryption is and how it can serve our purpose.

2.2. Fully Homomorphic Encryption

2.2.1. Homomorphic Encryption: definition and types

Homomorphic encryption (HE) is a type of encryption that enables computations to be carried out on encrypted data without decrypting it first. The result of computations is in encrypted form and can only be decrypted by the intended recipient. The most important characteristic is that when decrypted, the output is identical as if the mathematical processes were performed on unencrypted data. [21] [22] [23] Unluckily, there are some limitations to the type of calculations that can be performed with homomorphic encryption, besides the performance issue that has been improved with the hardware advancement. [24]

Homomorphic encryption is generally categorised into three main types: Partial Homomorphic Encryption (PHE), Somewhat Homomorphic Encryption (SHE), and Fully Homomorphic Encryption (FHE).

Partial Homomorphic Encryption (PHE) schemes only support a limited number of operations, like addition or multiplication, but not both. An example of PHE is the RSA (Rivest-Shamir-Adleman) algorithm, which supports multiplication operation on ciphertexts, but doesn't support the addition operation. [21] [25]

Somewhat Homomorphic Encryption (SHE) schemes are designed to support more than one type of operation, usually both addition and multiplication. Unfortunately, they can only handle a limited number of these operations due to an accumulation of noise in the encryption process, after which the original message cannot be accurately recovered. An example of SHE is the BGV (Brakerski-Gentry-Vaikuntanathan) scheme. [26]

Fully homomorphic encryption (FHE) schemes are an advancement over SHE and they are considered "the holy grail" of homomorphic encryption. [27] FHE overcomes the limitations presented above by allowing any number of computations to be performed on the encrypted data, including complex computations like machine learning algorithms. [28] [29] This means that any function, no matter how complex, can be computed on the encrypted data without the need for decryption, thereby ensuring privacy. It's worth noting that this type of encryption was purely theoretical for a long time until Craig Gentry proposed the first practical FHE scheme in 2009. [30] [31]

In the field of cryptography, an unprocessed value is known as a plaintext or cleartext. When this plaintext is encrypted using an encryption algorithm, the resulting unreadable text is called ciphertext.

The principle of HE is that you can compute on ciphertexts while not having knowledge of the messages that have been encrypted. To be considered fully homomorphic, a system should support at least two of the following operations (where x, y are plaintexts and $E[x], E[y]$ are the corresponding ciphertexts) [32]:

- Homomorphic univariate function evaluation:
  $\boldsymbol{f(E[x]) = E[f(x)]}$, where f is the computation function.
- Homomorphic addition:
  $\boldsymbol{E[x] + E[y] = E[x + y]}$
- Homomorphic multiplication:
  $\boldsymbol{E[x] * E[y] = E[x * y]}$

The essence of FHE is to generate ciphertext results for any given functions and encrypted messages, as long as no information is leaked. For a better understanding of Fully Homomorphic Encryption's main objective, Figure 1 illustrates the expected operation of an FHE scheme z using the classic black box model in computer systems. [31]

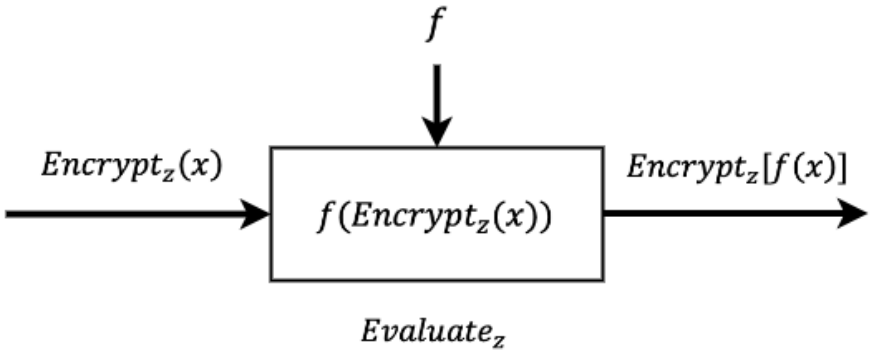

Figure 1 Fully homomorphic encryption conceptualization

The demanding target is to identify the appropriate mechanism $Evaluate_z$ that can produce the output efficiently within a reasonable amount of time.

Fortunately, Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène from Zama France presented such a scheme in their 2018 paper "TFHE: Fast Fully Homomorphic Encryption over the Torus", a scheme underlying the Rust *TFHE-rs* Framework. This framework plays a crucial role in the solution's FHE processing applications. [22] [32] [33]

2.2.2. Bootstrapping

The security of an FHE scheme is based on concealing the original message with a certain level of noise. In the past, before the introduction to FHE, this could only be removed through decryption. This limitation restricted the number of operations that could be performed on ciphertexts, as the error grows with each homomorphic operation, making decryption impossible when the error surpasses a threshold. [31]

Most fully homomorphic encryption solutions are based on challenging lattice problems. Consequently, the generated ciphertexts must possess a specific amount of noise to ensure encryption security. However, performing homomorphic computations amplifies the noise level within the ciphertext. While the noise remains below a particular threshold, the ciphertext can still be decrypted. Nevertheless, excessive noise growth can potentially overflow onto the data itself, making it nearly impossible to decrypt. [34]

The noise-reduction operation, called **bootstrapping**, was introduced to keep the error within the threshold and to create the first FHE scheme based on a bootstrappable SHE (Somewhat Homomorphic Encryption) scheme. [21] This type of scheme can homomorphically evaluate its own decryption function. [24] [34]

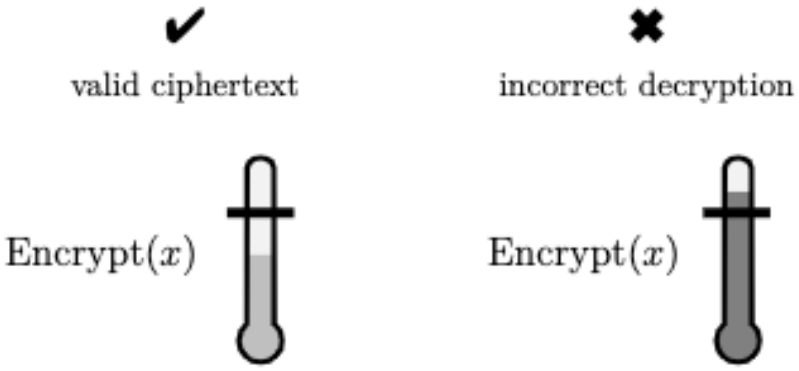

Figure 2 Dealing with noise [34]

In FHE, the bootstrapping process's core part is the “recrypt” function, which effectively resets the noise in the ciphertexts, within parameters. This procedure can be repeated indefinitely to obtain fresh ciphertexts, ensuring correct decryption after an unlimited number of operations. [35] [36]

2.3. System design analysis

2.3.1. Overview

As previously stated, in order to become part of the secure data exchange and processing "ecosystem", it is essential to fulfill the initial requirement of providing data in order to make a request. Consequently, the strategy I am proposing is designed to work within the existing infrastructure of a financial institution (FI).

A critical aspect that requires careful consideration revolves around the question: "What data should the participant provide?". This subject necessitates further investigation and research, especially on the economic, legal and risk management sides. In this thesis, we will assume that if a financial institution can provide around 80% of the information required, it can join the new communication schema. We will also assume that all the specifications provided about a client are equally important.

In continuation, I describe the one-to-one relationship between the **requester** FI and the **provider** FI. The primary focus is placed on their interaction, data exchange, and, most importantly, the encrypted processing of that data using a confidential schema.

The Minimum Viable Product (MVP) associated with this thesis serves as a demonstration, showcasing the capabilities of Fully Homomorphic Encryption. It utilises this encryption technique to compute a risk score by following a secret risk assessment schema, all while operating on encrypted data. The algorithm is developed to reinforce the primary focus without being tested in real-world scenarios or practical applications.

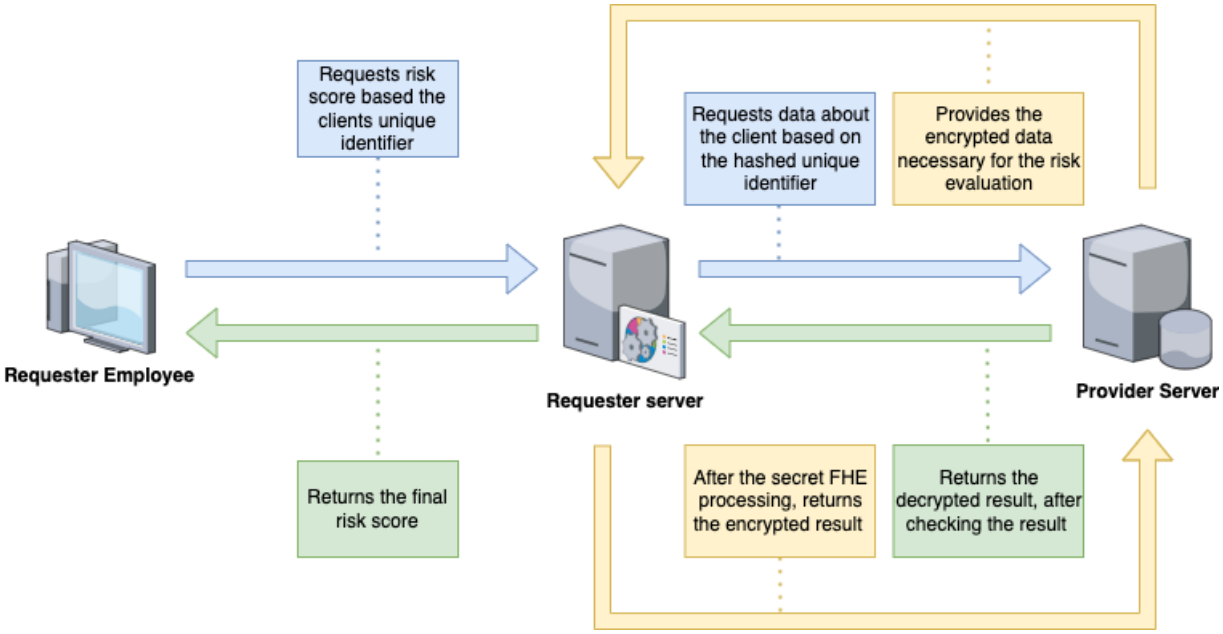

Figure 3 One requester – one provider interaction overview

As depicted in Figure 3, the diagram presents three distinct components, each component being assigned to one of the two primary entities: the requesting financial institution side (referred as **RFI** or **the requester**) and the providing financial institution side (referred as **PFI** or **the provider**).

Briefly outlining the main roles of each entity, RFI is responsible for processing the provided encrypted data using its proprietary FHE schema and computing the needed risk score. On the other hand, the PFI is tasked with providing the required data, encrypting it, and decrypting the ultimate outcome. Before returning the decrypted result to the requester, it is recommended for the provider to thoroughly examine the value for any potential conflicts with the initial data.

2.3.2. Prerequisites

As a first prerequisite, each participant must create a mechanism to efficiently verify if they possess information related to the requested client, based on a provided hashed unique identifier (such as a passport ID or personal identification number).

To accomplish this, participants can create a database table akin to a rainbow table. This table serves as a mapping between the hashed value and the corresponding actual value. It is worth noting that in certain cases, different financial institutions may utilise distinct keys for retrieval, leading to the possibility of the "rainbow table" containing multiple relevant values depending on the system in place. The implementation of this mechanism can be customised by each participant according to their specific requirements and preferences, the “rainbow table” version being just an example.

Maintaining the correct order and type of data provided is essential, particularly when processing encrypted information. To ensure consistency and accuracy, each enrolled institution will receive a documentation outlining the necessary guidelines for their implementation. This documentation will provide detailed information regarding the data types, the precise order of data transfer, and other essential details such as the specific data's

meaning or representation. By following the documentation, participants can ensure smooth and standardised data processing within the system.

In theory, a financial institution can join only as a provider without having the requester side, so having a proprietary risk algorithm is not considered a mandatory prerequisite. However, it is important to note that the focus of this paper lies in the secret Fully Homomorphic Encryption (FHE) processing schema.

2.3.3. Workflow

The UML Activity Diagram, as depicted in Figure 4 facilitates the visualization and understanding of business processes and workflows, in addition to providing technical insights that will be discussed in the upcoming chapter.

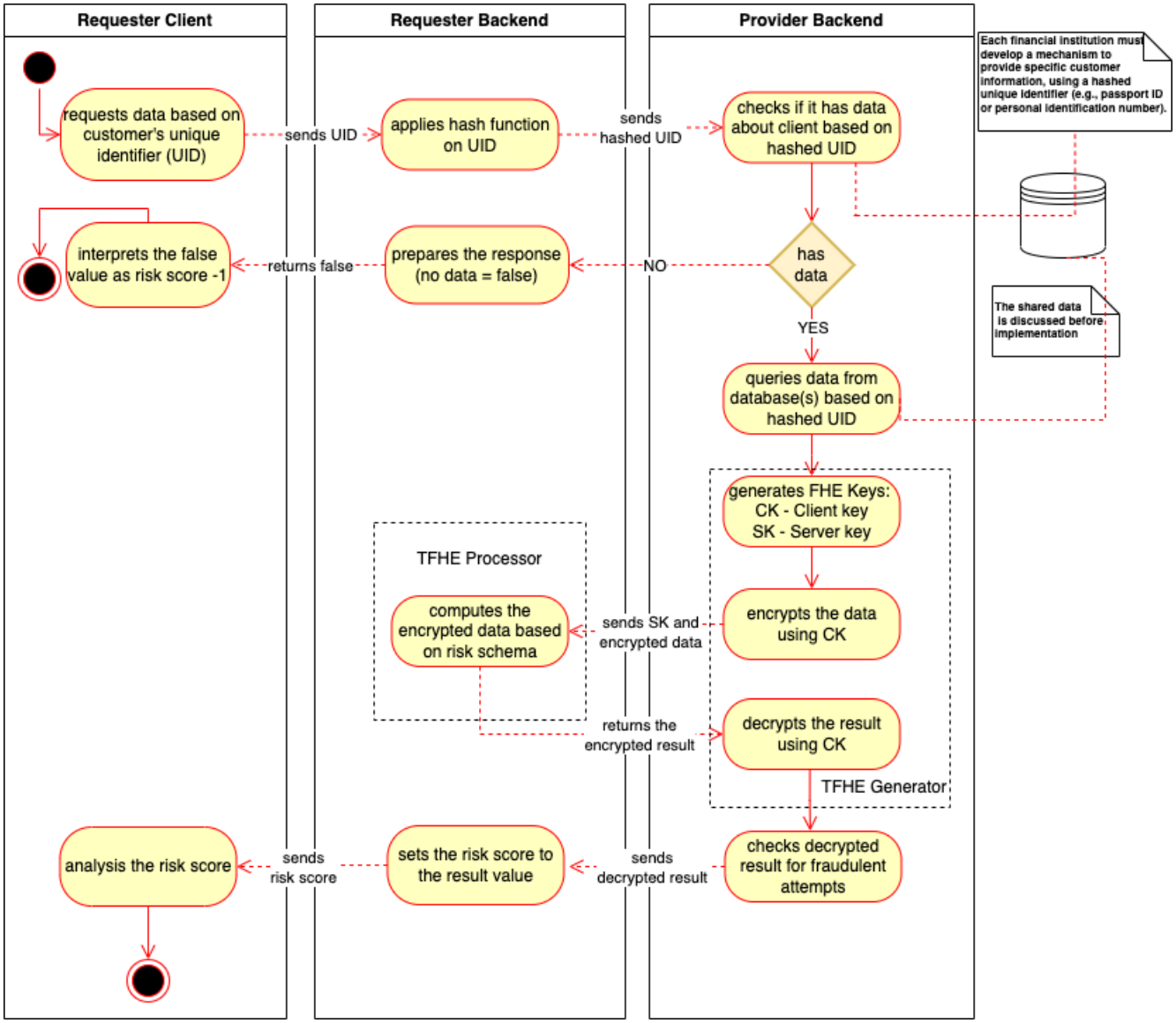

Figure 4 Activity Diagram

Continuing further, we will outline the possible scenarios of the solution. It is important to note that we will focus on the business aspects rather than technical situations, such as network connectivity issues or interruptions.

The employee of the requesting financial institution (***Requester Client***) calls their backend server to obtain the risk score associated with client's unique identifier (UID). The server (***Requester Backend*)** applies a hash function on the uid and then creates a request to providing financial institution ***(Provider Backend*)**.

At the provider level, the hash value is examined to locate a corresponding value. The primary purpose of hashing the unique identifier is to provide protection against sharing any client information in cases where the provider has no prior relationship with the individual or business in question.

If there is no data, the requester is notified and the risk score is set to -1, meaning there is no information about the person. Otherwise, the provider queries its database to retrieve the needed data and encrypts it using FHE.

Before the encryption, the required Fully Homomorphic Encryption (FHE) keys are generated. These keys consist of **the client key**, serving as the actual private key used for encrypting the data and decrypting the final result, and **the server key**, which functions as a public key. The server key is necessary for FHE processing on the requester side. To avoid confusion, these key names are maintained from the Rust Framework utilised for the FHE operations.

The sequential order of information transmission is one of the most important aspect of the specification. Since the exact value is not visible, it becomes mandatory to have precise knowledge of the specific order and data type to accurately incorporate it into the computational algorithm.

After receiving the server key and the encrypted data, the requester starts the computation for the risk score. Then, returns the ciphertext to the provider for the decrypting stage.

The final outcome is sent back to the provider, initiating the verification stage. In this stage, the result undergoes examination for potential fraudulent attempts, such as leveraging the risk schema to extract the initial value.

Finally, the verified result is shared with the requester, who then generates the risk score report. The assessment is then conducted by the employee, drawing upon the formula's documentation and potentially consider other statistics available to the financial institution from alternative communication channels.

It is noteworthy to include, in the schema interpretation documents, the scenarios where the provider may lack data for certain types. Unfortunately, the requester may not be directly aware of this information, but can assign additional weight in the formula for the different data types. As previously mentioned, in our Minimum Viable Product (MVP), we will treat each received data type as equally important.

2.3.4. Interaction analysis

2.3.4.1. One provider -one requester

Until now, we have provided a general overview of the backend aspects for each side. In the "backstage", each backend consists of two applications that communicate with one another:

The first is the API, which is responsible for data exchange and external communication.

The second is the TFHE (Fast Fully Homomorphic Encryption over the Torus) application, which specifically handles the FHE operations.

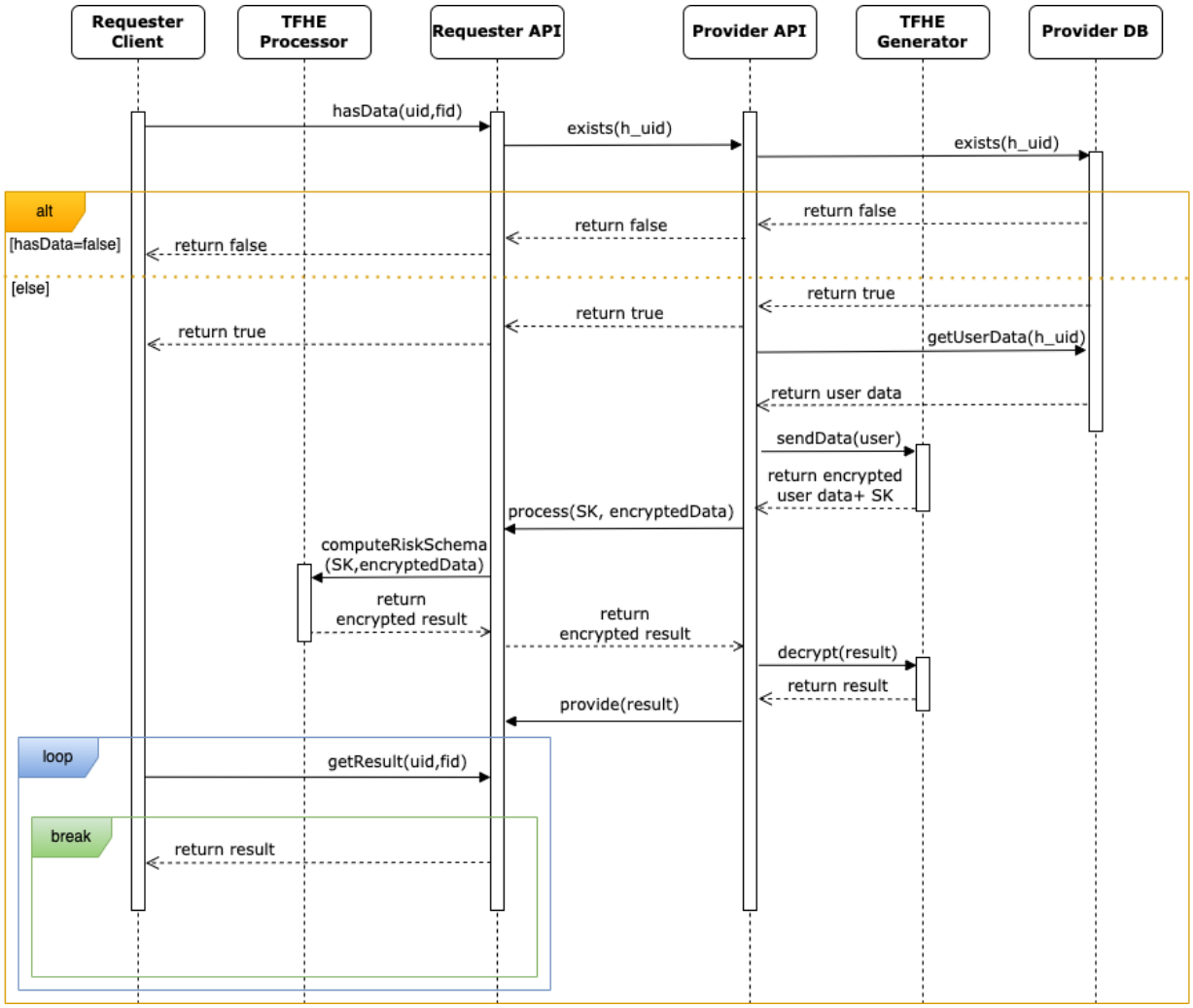

Figure 5 Sequence diagram

Figure 5 presents all the components of the solution, with the exception of the provider database, which - in real world scenario - can consist of one or more databases or potentially another application. For our purposes, we assume that all the necessary information can be obtained from this source. It is important to note that the provider DB serves primarily as a demonstration and does not have a direct impact on the solution.

In the sequence diagram depicted in Figure 5, we can visualise the chronological order of interactions and messages (information) exchanged between one requester's components and one provider's components. This diagram allows us to visualize the collaboration and communication between these entities as well as within each entity over time. It enables a comprehensive understanding of the system's functioning, the roles of each application, and their respective lifecycles.

In comparison to the workflow, we can now establish a technical understanding of the precise data retrieval, transfer, and processing mechanisms, including the timing, location, and responsible components.

We can observe a significant inclusion in the functions ***hasData(uid,fid)*** and ***getResult(uid,fid)***, wherein the ***fid*** parameter represents the **F**inancial Institution **ID**. Each enrolled entity possesses a distinct identifier. In this scenario, the id will serve as a guiding factor determining the recipient of the call.

It is important that, although the loop frame starts after the dispatch of ***provide(result)***, the actual calls begin earlier as there is no mechanism to notify the client when the data has been processed and decrypted. Consequently, the client must periodically query the backend until a valid result is received, at which point the loop is terminated.

2.3.4.2. One provider -many requesters

Thus far, our focus has been on comprehending the interaction of components within the one provider - one requester relationship. In a practical implementation, the requesting financial institution may concurrently request data from multiple providing financial institutions.

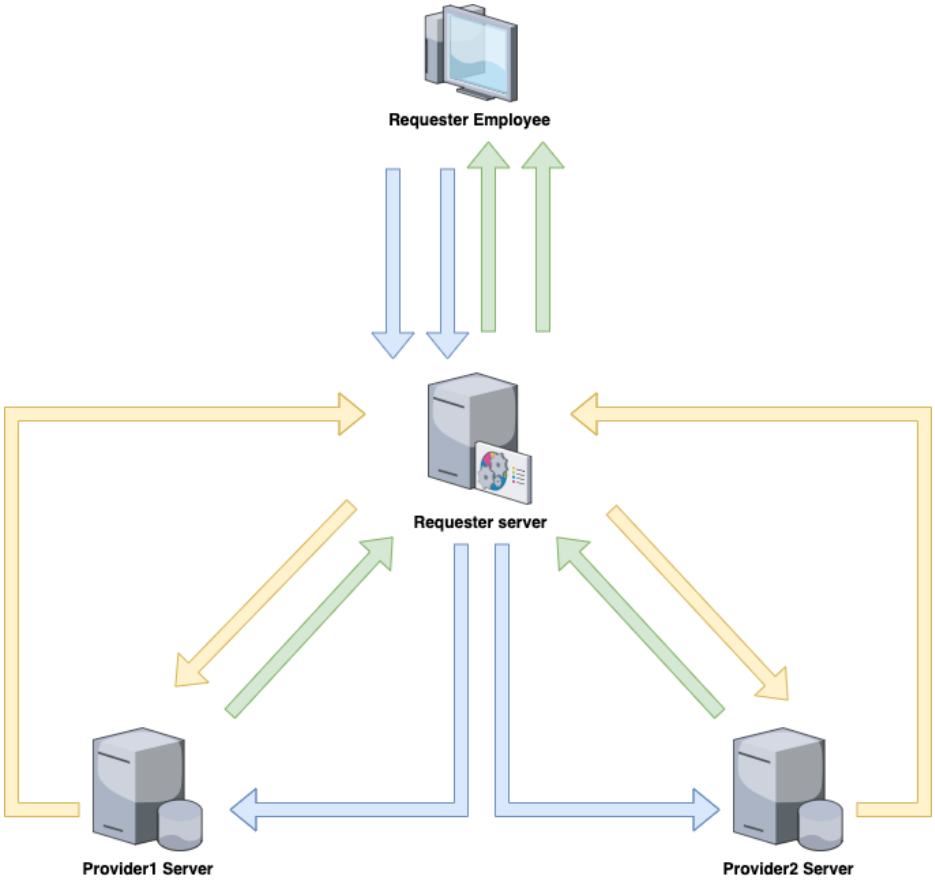

Figure 6 One requester – many providers interaction overview

Keeping the color structure and, implicitly, the specifications from Figure 3 One requester – one provider interaction overview, Figure 6 illustrates an overview on how the one-to-many communication will occur.

For each providing participant will be created a separate request. This allows the requester to query providers individually or collectively. Additionally, a distinctive header will be adopted for each call, ensuring the mapping of the client with the corresponding result from the specific provider.

Chapter 3. MVP Implementation

3.1. Risk formula

Inspired by the statistical techniques and categorisation methods which are starting to appear in banking applications, the risk formula I developed for this Minimum Viable Product lacks any foundation in economic or risk theory. It is created to perform as many FHE operations as possible.

Before we move forward, it may be important to note that this schema is centered around the individual, and some of the metrics employed are purely theoretical, without any solid background.

The information requested from the provider, based on its data from the last twelve months, is the following:

- **Gambling percent of total transactions (g);** it takes values from 0 to 100.
- **Overspending score**, a metric that analyses the consistency of spending more money than earning **(os);** it takes values from 0 to 24.
- **Impulsive buying score**, a metric that analyses the tendency to make unplanned and big transactions **(ib);** it takes values from 0 to 24.
- **Mean value of deposits (dm);**
- **Mean reported income (id);**
- **Number of months** (out of twelve) when **the individual deposited (nd).**

Now, we will turn our attention to the following mathematical formula:

$$RISK = g + MAX(os, ib) - r * nd,$$ where $$\begin{cases} r = 2, when\ dm > id \\ r = 0, when\ dm \leq id \end{cases}$$

Range	*Risk qualifier*
≤ 0	**NONE**
1 - 10	**LOW**
11 - 20	**MEDIUM**
21 - 40	**HIGH**
> 40	**CRITICAL**

Table 1 Risk formula interpretation

The preceding table provides a variant on how the final results could be interpreted. Despite the seemingly narrow range, it is crucial to acknowledge that a significant percentage of transactions in the gambling sector pose a genuine risk. In the context of stability and continuity, prioritising or setting aside money for emergencies, retirement, or other long-term goals, could be a factor that reduces the level of risk.

In addition, the potential negative impacts on financial health and individual behavior should not be underestimated, further amplifying the importance of careful risk management in these situations. According to our schema, this spending carelessly behaviour can be countered with saving money. For this reason, the value for the two score goes up to 24. This value represents the relevance of deposits ($r = 2$) multiplied by twelve months.

3.2. Architecture

Using Figure 7 as a reference, we will proceed to describe the system's general architecture. The message exchange calls are represented by arrows, with the components using various protocols and APIs to communicate with each other:

HTTP (Hypertext Transfer Protocol)

This protocol is used between Requester Client – Requester API – Provider API, due to the fact they will transfer data and resources over the internet. In a future production environment, Secure HTTP (HTTPS) should be used to secure the communication channel.

TCP (Transmission Control Protocol)

The decision of using TCP is its key design: "ensuring reliable, ordered, and error-checked delivery of data packets over a network connection". For the solution, it is crucial that the data be transferred and processed in the agreed order.

This transport layer protocol will take place between the TFHE applications, which will act as a TCP server, and the API applications, which will have embedded a TCP client. This communication type will happen on the same server, as we can see in the architecture schema.

JBDC (Java Database Connectivity)

Due to the decision to use Java – Spring Framework, this Java API will provide standardised and convenient way to interact with relational databases, allowing for efficient and structured database operations.

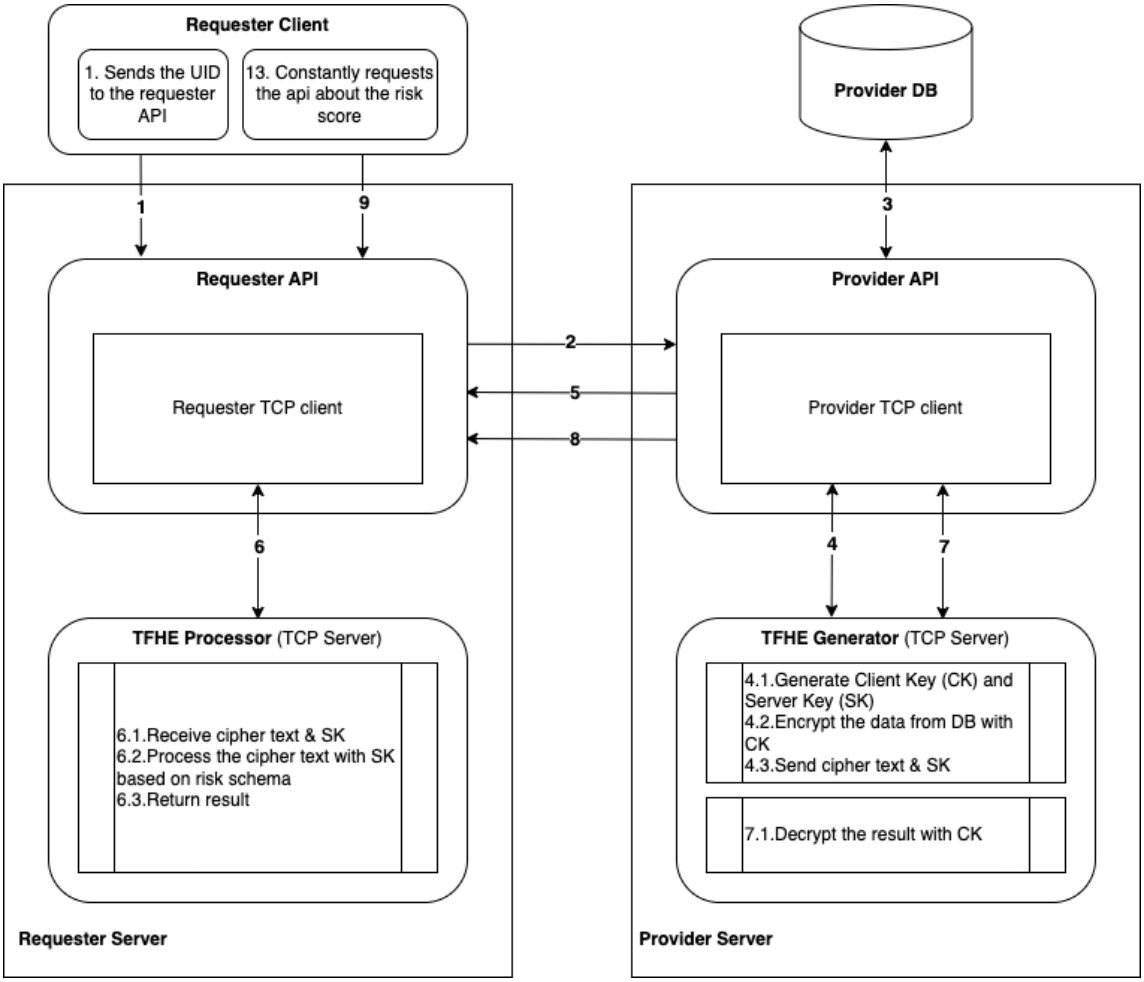

Figure 7 System architecture

3.2.1. Requester side overview

The requesting financial institution consists of three applications, each one having a specific role, as follows:

1. **Requester Client;**
2. **Requester API;**
3. **TFHE Processor.**

Requester Client is the front-end application where the organisation's delegated employee requests and receives the risk level of a possible new customer from the enrolled entities. It is worth mentioning that, in practical scenarios, this client can be seamlessly integrated into the Backoffice applications of a bank or a non-banking financial institution.

The Requester API serves as the first backend application having the primary objective of facilitating the exchange and manipulation of data between different components. Additionally, it incorporates a TCP Client to establish communication with the TFHE Processor.

TFHE Processor is the second backend application. Here, the risk level is calculated using the available FHE operations (that depend on the ServerKey) in the proprietary risk algorithm. It will communicate only with the TCP Client from the Requester API, without having any internet exposure. To prioritize data security and accommodate the characteristics of encrypted data, both entities will securely transmit data in the form of byte chunks through the localhost interface.

3.2.2. Provider side overview

Despite the requester, the provider does not have a frontend application since there is no need for direct human intervention in the workflow, but the structure is similar:

1. **Provider API;**
2. **TFHE Generator.**

The Provider API has a comparable structure like the requester one and serves the same purpose. In real-world situations, it might interact with multiple databases or applications depending on each participant's core system. For our MVP, we will assume that all the required data will be obtained from the **Provider DB**, a relational database.

TFHE Generator has three distinct roles, in this exact order:

- Generates the necessary keys for encrypting, decrypting, and processing data using FHE calculus (ServerKey and ClientKey);
- Encrypts the client data that will be sent to the requester to be processed, using the ClientKey;
- Decrypts the result after the requester applies its own risk schema over the data, using the ClientKey.

Based on the 3rd role, an important question arises: "Why does the provider have access to the result of the requester's secret computation schema?".

The purpose of the result is to be a value that does not reveal significant information to the provider due to a lack of interpretational knowledge. It may be the final output or an intermediate one. This mechanism helps prevent fraudulent attempts by the requester to acquire concrete information about the client, such as their income information.

3.2.3. Components interaction specification

The table below provides an in-depth explanation of each call and the data flow among the components of the solution. The arrows from Figure 7 System architecture showcases several important aspects: the sequence of operation, the communication, the dependency, and the data payload.

Index	**Source**	**Destination**	**Communication Type**	**Data Payload (Request)**	**Data Payload (Response)**	**Response Dependency**
1	Requester Client	Requester API	HTTP - REST	uid, fid	true/false	2
2	Requester API	Provider API	HTTP – REST	hashed uid	true/false	3
3	Provider API	Provider DB	JBDC – Database Query	***SELECT*** on the hashed uid	user data if exists	N/A
4	Provider API	TFHE Generator	TCP	user data	encrypted user data and server key (in bytes)	N/A
5	Provider API	Requester API	HTTP - REST	encrypted user data and server key (in bytes	encrypted result (in bytes)	6
6	Requester API	TFHE Processor	TCP	encrypted user data and server key (in bytes)	encrypted result (in bytes)	N/A
7	Provider API	TFHE Generator	TCP	encrypted result (in bytes)	decrypted result	5
8	Provider API	Requester API	HTTP - REST	decrypted result	N/A	7
9	Requester Client	Requester API	HTTP - REST	uid, fid	Decrypted result (risk score)	8

Table 2 Components interaction and data flow

3.3. Implementation and technologies

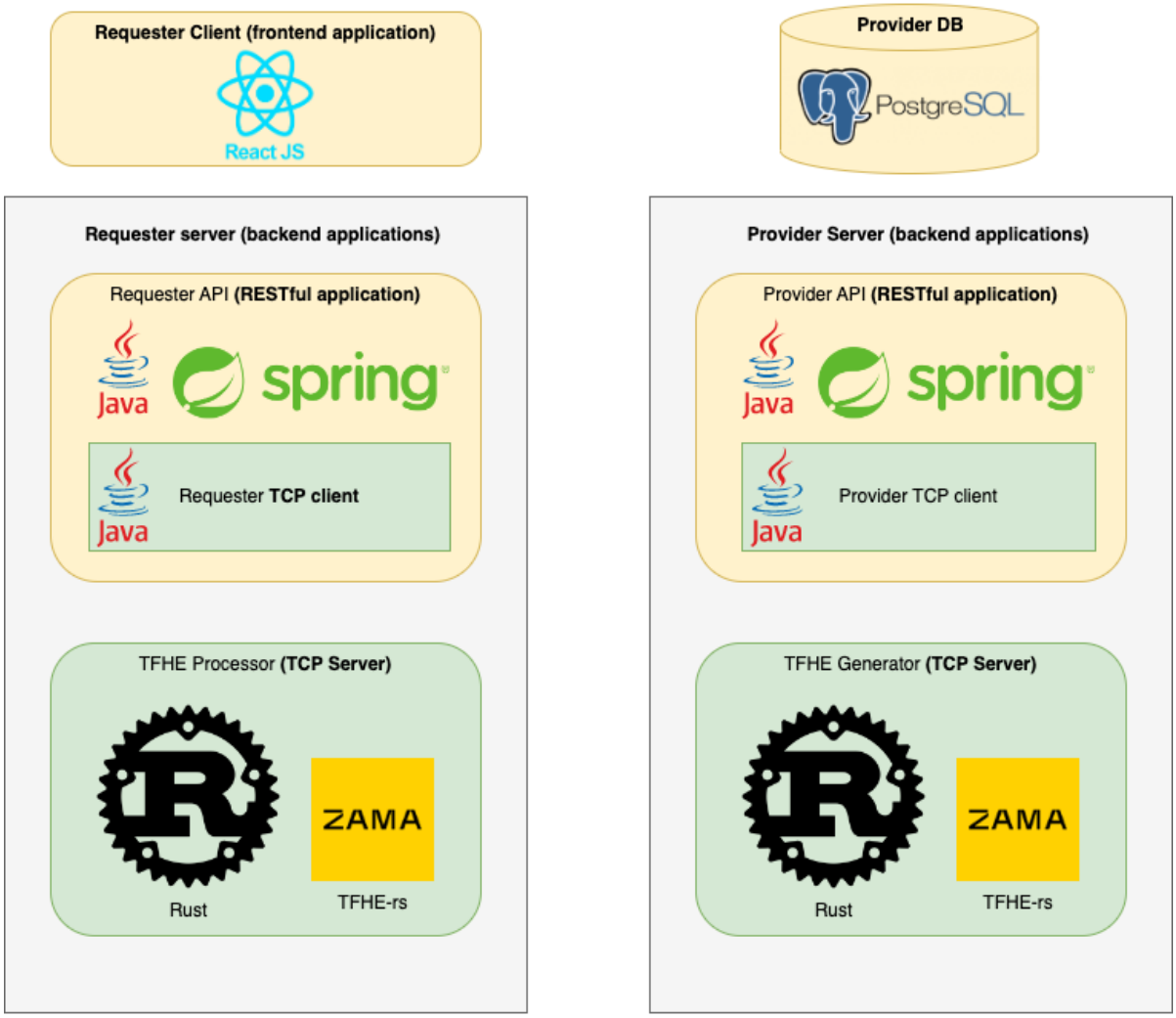

Figure 8 Stack of technologies used based on architectural design

3.3.1. TFHE Processor/Generator

The core of the solution revolves around performing Fully Homomorphic Encryption computations, which are executed on the TCP servers mentioned earlier – TFHE Processor and Generator.

To implement this, I opted for Zama's framework - TFHE-rs (Fast Fully Homomorphic Encryption over Torus), a pure Rust library designed for performing boolean, shortints and integer arithmetic on encrypted data [32], which is based on the concept of programmable bootstrapping. Torus is a mathematical structure representing a number in a finite ring called the "Torus ring". The Torus ring is defined as the set of real numbers modulo 1. In other words, Torus values are real numbers between 0 and 1, with the understanding that values on the boundaries wrap around. [37]

Programmable bootstrapping is an efficient technique in Fully Homomorphic Encryption that reduces noise in ciphertexts by homomorphically evaluating a function, specified as an input, during the bootstrapping process. [22] [29]

The main reason for choosing TFHE-rs is its high-level implementation, which allows us to focus on the business problem at hand, without being concerned about the complex mathematics involved in Fully Homomorphic Encryption operations.

In the MVP, the integer module will undergo implementation, using several of the available operations provided by the framework:

Operation	**Type**
Negation	Unary
Addition	Binary
Subtraction	Binary
Multiplication	Binary
Bitwise OR, AND, XOR	Binary
Equality	Binary
Left/Right Shift	Binary
Comparison (<,>,<=,>=)	Binary
MIN, MAX	Binary

Table 3 TFHE-rs supported operations for integers

3.3.1.1. Key generation

TFHE-rs offers two integer representation methods: Radix and Chinese Remainder Theorem (CRT). I utilised the Radix method for this implementation. [32]

To explain further, the radix-based representation of integers involves breaking down a larger integer into smaller chunks, or "blocks". Each block represents a digit in the radix (base) system we are using. Let's consider an analogy with the decimal number system:

The number 1234 can be broken down into blocks of single digits [1, 2, 3, 4]. In this case, the number of blocks is 4, and the radix (base) is 10.

```
use tfhe::integer::{gen_keys_radix, RadixClientKey, ServerKey};
use tfhe::shortint::parameters::PARAM_MESSAGE_2_CARRY_2;

pub(crate) fn key_gen(num_block: u32) -> Result<(RadixClientKey, ServerKey), Box<dyn Error>> {

// ...code

    let (client_key, server_key) = gen_keys_radix(&PARAM_MESSAGE_2_CARRY_2, num_block as usize);

// ...code

}
```

Code Snippet 1 Key generation function

During the key generation phase, an integer is defined through specifying a base and a quantity of blocks. Currently, only unsigned integers are accommodated, with the developers proactively addressing this limitation.

The ***num_block*** parameter in the function from Code Snippet 1 determines the number of divisions, or blocks, into which an integer is segmented for encryption purposes. By applying a 2-bit radix along with a ***num_block*** of 8, it becomes possible to represent 16-bit integers ($2\ bits/block\ *\ 8\ blocks\ =\ 16\ bits$). This choice allows us to represent numerical values

ranging from 0 to 65.536, which meets the needs of our Minimum Viable Product (MVP). At the time of speaking, this framework can handle integers of up to 256 bits.

PARAM_MESSAGE_2_CARRY_2 is a constant that holds the default set of parameters for the cryptographic scheme.

3.3.1.2. Transposing risk formula into code

After deserialising the variables in the specific order, the next step is to apply the right operations for the formula. Despite the variety of operational "flavors" offered by the framework, I have chosen to use the default type. This decision is driven by the need to maintain data integrity and to guarantee consistent timings across all function calls.

```rust
fn fhe_risk_score(serialized_data: &[u8]) -> Result<Vec<u8>, Box<dyn std::error::Error>> {
    let mut serialized_data = Cursor::new(serialized_data);
    let server_key: ServerKey = bincode::deserialize_from(&mut serialized_data)?;
    let _gambling_percent: RadixCiphertextBig = bincode::deserialize_from(&mut serialized_data)?;
    let _overspending_score: RadixCiphertextBig = bincode::deserialize_from(&mut serialized_data)?;
    let _impulsive_buying_score: RadixCiphertextBig = bincode::deserialize_from(&mut serialized_data)?;
    let _mean_deposited_sum: RadixCiphertextBig = bincode::deserialize_from(&mut serialized_data)?;
    let _mean_reported_income: RadixCiphertextBig = bincode::deserialize_from(&mut serialized_data)?;
    let _no_months_deposited: RadixCiphertextBig = bincode::deserialize_from(&mut serialized_data)?;

    let max = server_key.max_parallelized(&_overspending_score, &_impulsive_buying_score);
    let mut result = server_key.add_parallelized(&_gambling_percent, &max);
    let condition = server_key.gt_parallelized(&_mean_deposited_sum, &_mean_reported_income);
    let r = server_key.unchecked_scalar_left_shift(&condition, 1);
    let risk_counter = server_key.mul_parallelized(&r, &_no_months_deposited);
    result = server_key.sub_parallelized(&result, &risk_counter);

    let serialized_result = bincode::serialize(&result)?;

    Ok(serialized_result)
}
```

Code Snippet 2 Risk score algorithm

The first two operations presented in Code Snippet 2 are straightforward: they compute the maximum of two encrypted values and adds this value to the gambling percent. The ***gt_parallelized*** function (greater than) returns either 0 (for false) or 1 (for true), in encrypted form. This blocks us from using a conditional statement like “if”. To solve this, I devised a left-shift by 1 on the ***condition*** variable to compute the ***r*** parameter. This is because shifting 1 by 1 yields 2, and shifting 0 by 1 remains 0, thus satisfying the formula's requirement for deposit relevance.

Ultimately, the risk counter is deducted from the total risk. If the counter surpasses the total risk, the result defaults to 0 due to the constraints of unsigned integer representation.

3.3.2. API Applications

Regarding the REST APIs, I decided to go with Spring Framework, which is a comprehensive, enterprise-level Java framework providing foundational support for developing robust and scalable web applications. [38] Several reasons behind this decision are the following:

- Versatile Programming Model: Suitable for diverse types of applications;
- Aspect-Oriented Programming (AOP): Allows clean separation of cross-cutting concerns;
- Community and Documentation: Provides robust support and extensive guides;
- Regular Updates: Ensures adoption of latest technology trends;
- Robust Ecosystem: Includes related projects for various development needs;

Each API application contains an embedded TCP Client that's developed using Java core libraries exclusively. One critical detail is that Java employs network byte order, or big-endian, to signify multibyte values like integers. On the other hand, Rust uses the native byte order, also known as little-endian. So, when we transfer data through the transport layer, we have to adjust the byte order.

Aspect-Oriented Programming (AOP) is a programming paradigm that complements the traditional object-oriented programming (OOP) model. It increases modularity by separating cross-cutting concerns, using advice to add behavior to existing code without modifying it directly, and specifying code modifications through pointcut specifications. [39]

In my application, I specifically utilised the Spring Framework's AOP **@After** advice. With **@After**, I was able to automatically trigger **getUserData** method (Figure 5 Sequence diagram) after the requester is informed that the provider has data about the client in question.

3.3.3. Requester front-end

On the front-end side, I settled with React.js for this minimum viable product (MVP). React is a JavaScript library that enables developers to build efficient user interfaces for websites and applications with relative ease. One of its key concepts is the Virtual DOM, which is a JavaScript-based component tree constructed using React, imitating a DOM tree. It ensures minimal DOM manipulation to keep the associated components updated. [40] In our case, these components will be the risk score ones associated with each provider.

Despite the simplicity of the application, I believe choosing React is a wise decision due to several advantages it offers. These include efficient updates and rendering, a rich ecosystem (including various ready-to-go UI components), and a vast community of developers providing support and resources.

3.3.4. Provider database

PostgreSQL is a robust, cross-platform, open-source object-relational database management system (ORDBMS). Known for its exceptional transaction processing and data analysis capabilities, PostgreSQL fits perfectly into our enterprise application scenarios, more exactly, finance.

It is ideal for highly transactional, mission-critical applications, supported by comprehensive free documentation. The primary advantage of PostgreSQL lies in its high data consistency and integrity. Furthermore, it embodies four fundamental features, known as Atomicity, Consistency, Isolation, Durability (ACID), ensuring transactions remain accurate and reliable during data writing or updating processes. [41]

3.4. Findings and implications

The correctness of the result is ensured as long as the values (both input and output) do not surpass the maximum integer value. The maximum integer value in question is determined by the bit representation of the integer.

In addition, I run a set of benchmarks to determine the most demanding segment. I concluded that the most complex part is the risk score computation. This reinforces the idea that the processing should be done on the requester's side. As the complexity of the algorithm increases, it becomes more resource and time-consuming.

The following benchmarks were conducted using a MacBook Pro 2019 equipped with a 2.6 GHz 6-Core Intel Core i7 processor, an AMD Radeon Pro 5300M graphics card with 4 GB of memory, and 16 GB of 2667 MHz DDR4 memory.

```
Integer representation: 8 bits
Key generation: 11153ms
Data used for risk score calculation:
- Gambling percent: 6
- Overspending score: 10
- Impulsive buying score: 3
- Mean deposit sum: 56
- Mean reported income: 50
- No months deposited: 2
Encryption time: 19.585983ms
Input serialization time: 823.525074ms
Deserialization time: 1.478862863s
Processing time (including serialization) is: 155.944258356s
Decryption time (including deserialization)): 759.283µs
Risk score: 12
```

Benchmark 1 Data set 1 with 8-bit integer representation

```
Integer representation: 16 bits
Key generation: 11726ms
Data used for risk score calculation:
- Gambling percent: 6
- Overspending score: 10
- Impulsive buying score: 3
- Mean deposit sum: 56
- Mean reported income: 50
- No months deposited: 2
Encryption time: 41.545776ms
Input serialization time: 876.11482ms
Deserialization time: 1.561691914s
Processing time (including serialization) is: 362.606290037s
Decryption time (including deserialization)): 1.484244ms
Risk score: 12
```

Benchmark 2 Data set 1 with 16-bit integer representation

In the case of Benchmark 1 and Benchmark 2, I employed Data set 1, which is designed to handle 8-bit integer representation, limiting us to a maximum value of 255. This constraint, however, doesn't align well with our business needs, as we expect the ***mean deposit sum*** and ***mean reported income*** to be expressed in euro and typically fall within the range of hundreds or thousands.

Despite this, I found it relevant to highlight the processing time difference: an 8-bit representation takes roughly 2,6 minutes, while a 16-bit representation doubles that time to approximately 5,8 minutes. Although the encryption and decryption time has considerably increased, it does not impact us as the values for both operations remain exceptionally low overall.

Data sets 2 and 3 have been adjusted to align with our business needs, the primary changes being a tenfold and hundredfold increase in salary-based values. Benchmark 3 is the variant incorporated into the Minimum Viable Product (MVP). For the moment, a maximum value of 65.536 should be comprehensive for the presentation.

Upon comparing Benchmarks 2, 3 and 4, a clear correlation emerges, indicating that the processing time is directly influenced by the magnitude of the values being handled. Fortunately, the increase is not that substantial like in the case of changing the integer representation.

```
Integer representation: 16 bits
Key generation: 10908ms
Data used for risk score calculation:
- Gambling percent: 6
- Overspending score: 10
- Impulsive buying score: 3
- Mean deposit sum: 560
- Mean reported income: 500
- No months deposited: 2
Encryption time: 40.178764ms
Input serialization time: 809.216551ms
Deserialization time: 1.474728144s
Processing time (including serialization) is is: 385.664128079s
Decryption time (including deserialization)): 1.659012ms
Risk score: 12
```

Benchmark 3 Data set 2 with 16-bit integer representation

```
Integer representation: 16 bits
Key generation: 11663ms
Data used for risk score calculation:
- Gambling percent: 6
- Overspending score: 10
- Impulsive buying score: 3
- Mean deposit sum: 5600
- Mean reported income: 5000
- No months deposited: 2
Encryption time: 49.540712ms
Input serialization time: 856.068755ms
Deserialization time: 1.515940531s
Processing time (including serialization) is: 403.630273582s
Decryption time (including deserialization)): 1.616786ms
Risk score: 12
```

Benchmark 4 Data set 3 with 16-bit integer representation

When transitioning to a 32-bit representation (Benchmark 5), the processing time experiences a significant increase, reaching approximately 16,9 minutes. This duration is nearly three times longer than the processing time observed in Benchmark 3.

```
Integer representation: 32 bits
Key generation: 11841ms
Data used for risk score calculation:
- Gambling percent: 6
- Overspending score: 10
- Impulsive buying score: 3
- Mean deposit sum: 560
- Mean reported income: 500
- No months deposited: 2
Encryption time: 81.816981ms
Input serialization time: 853.755869ms
Deserialization time: 1.55447896s
Processing time (including serialization) is: 1018.862911741s
Decryption time (including deserialization)): 3.201922ms
Risk score: 12
```

Benchmark 5 Data set 2 with 32-bit integer representation

Based on our conducted tests, it can be concluded that there is a direct relationship between the size of the values and their representation, and the corresponding processing time. Notably, the integer representation appears to be the most influential factor in determining the processing time. These benchmarks were specifically designed to focus on the minimum viable product (MVP) and the selected computation options.

In order to gain a better understanding of the framework's dynamics, it is recommended to redo the benchmarks using multiple schemas and various operation "flavours". This broader scope will provide more comprehensive insights into the performance and behavior of the framework.

Conclusion

To summarise, I managed to create a solution that leverages Fully Homomorphic Encryption (FHE) to evaluate an individual or a business's financial risk before they access new banking services. While the focus has been on the architecture of the system and the technologies used, the risk formula incorporated in the Minimum Viable Product serves merely as a proof of concept and calls for extensive study.

This paper simply sparks a dialogue on this novel form of synchronisation and secure data processing. For it to take on a definitive shape, additional research is required in fields like economics, statistics, and legislation. Until such advancements occur, the proposed solution cannot be wholly integrated into the existing financial system.

Despite the slightly longer processing time observed in the previous benchmarks, this solution remains significantly faster compared to communication processes involving human interaction and bureaucracy. Furthermore, it is important to note that the framework utilised in the minimum viable product (MVP) is still under development, with performance being one of the main objectives that the contributors strive to enhance.

I believe that this type of system will largely benefit non-banking financial institutions (NBFI) and will contribute to achieving a better balance. This belief is based on two main reasons. First, these institutions enjoy more freedom compared to banks, which are subject to stringent regulations and continuous examination. Second, unlike banks, NBFI lack significant monetary resources. Therefore, such a solution could boost their competitive edge and promote fairness in the financial industry.

In conclusion, if implemented effectively, this approach holds the potential to enhance the security and robustness of financial institutions, contributing to the overall stability of the financial system, and offering exciting opportunities for new areas.

Bibliography

[1] N. Shepherdson, "Credit Card America," *American Heritage*, vol. 42, no. 7, 1991.

[2] Wikipedia contributors, "Bank," Wikipedia, The Free Encyclopedia., 22 May 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Bank&oldid=1156315437. [Accessed 18 May 2023].

[3] Wikipedia contributors, "Bank regulation," Wikipedia, The Free Encyclopedia., 6 December 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Bank_regulation&oldid=1058897166. [Accessed 18 May 2023].

[4] J. Benjamin, "Financial law," in *Oxford University Press Inc*, Oxford, UK, 2007.

[5] Wikipedia contributors, "Non-bank financial institution," Wikipedia, The Free Encyclopedia., 23 April 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Non-bank_financial_institution&oldid=1151869979. [Accessed 20 May 2023].

[6] The World Bank Authors, "Nonbanking financial institution," The World Bank, 2016. [Online]. Available: https://www.worldbank.org/en/publication/gfdr/gfdr-2016/background/nonbank-financial-institution. [Accessed 20 May 2023].

[7] J. Chen, "Nonbank Financial Institutions: What They Are and How They Work," Investopedia, 04 April 2022. [Online]. Available: https://www.investopedia.com/terms/n/nbfcs.asp. [Accessed 20 May 2023].

[8] S. Michael, "Financial Regulators: Who They Are and What They Do," Investopedia, 06 December 2021. [Online]. Available: https://www.investopedia.com/articles/economics/09/financial-regulatory-body.asp. [Accessed 15 May 2023].

[9] European Banking Authority, "FINAL REPORT ON GUIDELINES ON COMMUNICATION BETWEEN COMPETENT AUTHORITIES AND AUDITORS," European Banking Authority, Paris, 2016.

[10] S. V. Scott and M. Zachariadis, "A historical analysis of core financial services infrastructure: society for worldwide interbank financial telecommunications (SWIFT)," *Working paper series (182)*, September 2010.

[11] S. V. Scott and M. Zachariadis, "The Society for Worldwide Interbank Financial Telecommunication (SWIFT): Cooperative governance for network innovation, standards, and community," in *Routlege in London*, October 2013.

[12] Wikipedia contributors, "Automated clearing hous," Wikipedia, The Free Encyclopedia., 14 May 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Automated_clearing_house&oldid=1154737258. [Accessed 15 May 2023].

[13] H. Adam, "Visa vs. Mastercard: What's the Difference?," Investopedia, 17 July 2021. [Online]. Available: https://www.investopedia.com/articles/personal-finance/020215/visa-vs-mastercard-there-difference.asp. [Accessed 15 May 2023].

[14] European Central Bank, "Single Euro Payments Area (SEPA)," [Online]. Available: https://www.ecb.europa.eu/paym/integration/retail/sepa/html/index.en.html. [Accessed 14 May 2023].

[15] D. Patel and S. Alapiha, "How do banks talk to each other and what are the different types of messages used by SWIFT?," The finance global, 6 March 2023. [Online]. Available: ttps://www.tradefinanceglobal.com/posts/how-do-banks-talk-to-each-other-and-what-are-the-different-types-of-messages-used-by-swift/. [Accessed 2 May 2023].

[16] P. Simone, "Society for Worldwide Interbank Financial Telecommunication.," *The Wiley-Blackwell Encyclopedia of Globalization,* 29 February 2012.

[17] D. Duffie, "Replumbing our financial system: Uneven progress," *International Journal of Central Banking*, no. 29, January 2013.

[18] Wikipedia contributors, "Central bank," Wikipedia, The Free Encyclopedia., 9 May 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Central_bank&oldid=1153902209. [Accessed 14 May 2023].

[19] M. Ehrmann and M. Fratzscher, "HOW SHOULD CENTRAL BANKS COMMUNICATE?," *WORKING PAPER SERIES,* vol. No. 557, November 2005.

[20] S. Sulochana, P. Seeprata and R. P. Udaya, "Communication in Banking Sector: A Systematic Review," *Quest Journal of Management and Social Sciences,* vol. 1, no. 2, pp. 272-284, 2019.

[21] Wikipedia Authors, "Homomorphic encryption," Wikipedia, The Free Encyclopedia, 2023 March 7. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Homomorphic_encryption&oldid=1143437105. [Accessed 2023 April 30].

[22] I. Chillotti, N. Gama, M. Georgieva and M. Izabach, "TFHE: Fast Fully Homomorphic Encryption over the Torus?," *Cryptology ePrint Archive,* 2 April 2018.

[23] Z. Zheng, K. Tian and F. Liu, "Fully Homomorphic Encryption," in *Modern Cryptography Volume 2*, Springer, 2023, pp. 143-147.

[24] H. Byun, "The Advantages and Disadvantages of Homomorphic Encryption," baffle.io, 27 February 2019. [Online]. Available: https://baffle.io/blog/the-advantages-and-disadvantages-of-homomorphic-encryption/. [Accessed 2023 May 7].

[25] L. Morris, "Analysis of Partially and Fully Homomorphic Encryption," *Rochester Institute of Technology,* vol. 10, no. 1-5, 2013.

[26] B. Michael, X. William, K. Mohammed and C. Weilian, 18 May 2017. [Online]. Available: https://courses.csail.mit.edu/6.857/2017/project/22.pdf. [Accessed 19 May 2023].

[27] S. Cristobal, "Fully Homomorphic Encryption: the Holy Grail of cryptography," The datascience.aero blog, 05 December 2018. [Online]. Available: https://datascience.aero/fully-homomorphic-encryption-the-holy-grail-of-cryptography/. [Accessed 21 May 2023].

[28] M. Joye, I. Chillotti and P. Pascal, *Programmable Bootstrapping Enables Efficient Homomorphic Inference of Deep Neural Networks,* vol. https://whitepaper.zama.ai/, Cryptology ePrint Archive, Paper 2021/091, 2021.

[29] I. Chillotti, M. Joye and a. P. Paillier, "Programmable Bootstrapping Enables Efficient Homomorphic Inference of Deep Neural Networks," *Cryptology ePrint Archive,* 25 November 2021.

[30] C. Gentry, "Fully homomorphic encryption using ideal lattices," *roceedings of the forty-first annual ACM symposium on Theory of computing,* pp. 169-178, May 2009.

[31] B. Pulido-Gaytan, A. Tchernykh and J. e. a. Cortés-Mendoza, "Privacy-preserving neural networks with Homomorphic encryption: Challenges and opportunities.," *Peer-to-Peer Netw. Appl.,* no. 14, p. 1666–1691, 2021.

[32] Zama Community, "TFHE-rs Documentation," zama.ai, [Online]. Available: https://docs.zama.ai/tfhe-rs. [Accessed 2 May 2023].

[33] Docs.rs Team, "Crate tfhe," rust.lang.ro, [Online]. Available: https://docs.rs/tfhe/latest/tfhe/. [Accessed 2 May 2023].

[34] M. Joye, "SoK: Fully Homomorphic Encryption over the [Discretized] Torus," *IACR Transactions on Cryptographic Hardware and Embedded System,* 31 August 2022.

[35] F. Armknecht, C. Boyd, C. Carr, K. Gjøsteen, A. Jäschke, C. A. Reuter and M. Strand, "A guide to fully homomorphic encryption," *Cryptology ePrint Archive,* 16 December 2015.

[36] C. Gentry, S. Halevi and N. P. Smart, "etter Bootstrapping in Fully Homomorphic Encryption," *ryptology ePrint Archive,* 18 December 2021.

[37] M. Joye, "Fully Homomorphic Encryption over the [Discretized] Torus," *IACR Transactions on Cryptographic Hardware and Embedded Systems,* vol. 2022, no. 4, pp. 661-692, 2022.

[38] The Spring Team, "Spring Framework," spring.io, [Online]. Available: https://docs.spring.io/spring-framework/. [Accessed 14 May 2023].

[39] Wikipedia contributors, "Aspect-oriented programming," Wikipedia, The Free Encyclopedia., 4 June 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Aspect-oriented_programming&oldid=1158542620. [Accessed 11 June 2023].

[40] O. Hutsulyak, "10 Key Reasons Why You Should Use React for Web Development," TechMagic, 12 March 2023. [Online]. Available: https://www.techmagic.co/blog/why-we-use-react-js-in-the-development/. [Accessed 6 June 2023].

[41] T. Davidson, "Why Use PostgreSQL For Your Next Project?," CleanCommit, 13 November 2022. [Online]. Available: https://cleancommit.io/blog/why-use-postgresql-for-your-next-project/. [Accessed 7 June 2023].