Feasibility of Homomorphic Encryption for Sharing i2b2 Aggregate-Level Data in the Cloud

Semester project report at EPFL

Author : Melchior Thambipillai
Supervisors : Jean-Louis Raisaro, Juan Troncoso-Pastoriza
Professor : Jean-Pierre Hubaux

January 21, 2018

Abstract

It is known that, as opposed to other fields, the biomedical community is still skeptical about the massive adoption of cloud computing for the management of sensitive medical data. Privacy concerns are certainly one of the main reasons. In this report, we try to address these concerns by enhancing i2b2 [1] functionalities in order to enable statistical queries on sensitive medical data stored under homomorphic encryption on an untrusted public cloud provider. Our goal is to make clinical sites more comfortable with sharing data in the cloud. In particular, we focus on EHR quality data whose unintended leakage to third parties could severely damage the reputation of the different sites. The proposed method relies on the combination of additively homomorphic encryption and proxy re-encryption in order to provide end-to-end data confidentiality while enabling some computations on the encrypted data. The design of the solution was already done and described in a previous report. This project focuses more on the implementation of that solution.

1 Introduction

With the increasing availability of EHR data, the ability to share them effectively and transparently is becoming a crucial requirement for precision medicine to scale up and fully realize its potential. Because of its capabilities, cloud computing is nowadays the most popular and efficient way to enable massive data sharing in a variety of fields. Yet, when it comes to healthcare, because of privacy concerns and stringent regulations, institutions such as hospitals, research centers or universities are still reluctant to outsource the storage of their sensitive medical data to untrusted cloud providers (e.g., Amazon AWS, Microsoft Azure, Google Cloud, etc.). The number of health data breaches is constantly increasing and there is significant public pressure to ensure that the privacy of patients and probands is protected. Indeed, unintended leakage of patients medical data can significantly harm the feeling of trust that people put in the healthcare system and, as a consequence, hinder the development of any precision medicine effort. As a result, implementing new technical solutions that, in synergy with regulations, can address these concerns by providing strong technical privacy and security guarantees is now more urgent than ever.

In the last few years, the IT security and privacy community has developed several tools, also named privacy-enhancing technologies (PETs), that try to solve the challenge of securely outsourcing sensitive data to an untrusted cloud provider (e.g., homomorphic encryption (HE), secure multi-party computation (SMC), functional encryption (FE), etc.). Yet, because of the cultural gap that still separates the security and privacy community from the biomedical informatics one, and the cost that some of these PETs introduce, there are only a few existing systems that are actually used in a clinical operational setting and leverage on one of these tools for the privacy-preserving management of sensitive medical data. Among the different PETs, homomorphic encryption (HE) is certainly the one that has the greatest potential for securely sharing medical data in the cloud. HE provides the ability to encrypt data and still perform some computations directly on the ciphertexts at the cost of an increased (but often acceptable) computational and storage overhead. As such, it can be used to protect sensitive data stored at an untrusted third party, such as a public cloud, and still benefit from the provided flexibility and massive computational resources. In particular, HE guarantees end-to-end confidentiality as the data on the cloud is never decrypted, neither at rest nor during computation. Only the owner of the decryption key can obtain the decrypted final result.

In this project, we want to bring homomorphic encryption to i2b2 by proposing a new architecture that enables to securely manage sensitive clinical data stored at an untrusted cloud provider. The proposed archi-

tecture sets the lowest bar possible, aimed at making clinical sites more comfortable with sharing data in the cloud, by enabling an investigator to perform i2b2-type queries directly on homomorphically-encrypted EHR data-quality (DQ) data. In particular, we combine partially-homomorphic encryption with proxy re-encryption to allow different i2b2 users with different cryptographic keys to access encrypted clinical data in the cloud in a flexible and efficient way. Our solution will be implemented as an i2b2 plugin and deployed in the ARCH clinical data research network to support the exploration of sensitive EHR DQ data variability. Despite the customized adaptation of our architecture to the EHR data quality exploration use case, the proposed solution can be easily extended to support the secure outsourcing of sensitive clinical and genomic data to the cloud without altering the standard i2b2 data model.

1.1 Contribution to the global project

At the beginning of this semester project, the modelling of the problem and the design of the solution as they are presented in Section 2 were already done [4]. A standalone application called the Crypto Engine [3] was already implemented, the main goal of this semester project was to integrate this application with the i2b2 framework. It required the implementation of a new i2b2 cell [6], some additional implementation and improvements in the Crypto Engine, as well as additional implementation in the i2b2 Webclient [5]. Several i2b2 use cases had to be implemented in a secure way. The details of the implementation done during this semester project are presented in Section 3. Section 4 will discuss the performance results of that implementation and finally Section 5 will present some possible future work.

2 Problem Statement

In this section, we will state the problem, the goals we want to achieve and the design of the solution. Then we will analyze the security of that design.

2.1 Model and Actors

We will first describe the model, the different actors and how they interact. We consider the system model depicted in Figure 1. This is representative of several PCORI Clinical Data Research Networks (CDRNs) that centrally aggregate their sites data for sharing purposes. Although the majority (about the two thirds) of PCORI CDRNs are still fully decentralized, there is a need to shift towards centralization because of insufficient resources (both human and technical) at the different clinical sites to maintain an inter-operable network. As a result, our system model consists of the following entities:

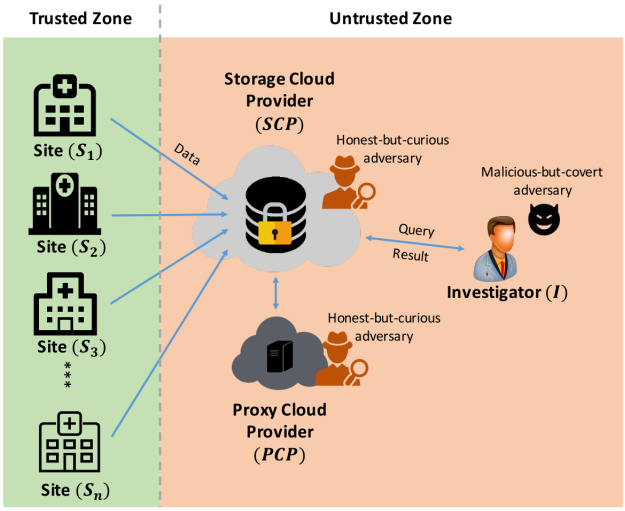

Figure 1: System model including several clinical sites, two cloud providers and one or multiple investigators; threat model including honest-but-curious adversary at both cloud providers and a malicious-but-covert adversary at the investigator.

- A federation or network of healthcare sites (S_i), such as hospitals or research centers, that own sensitive EHR quality data and want to share them with investigators inside and outside the network through the cloud
- One or several investigators (I) willing to query these data for exploratory analyses
- A storage cloud provider (SCP) responsible for securely storing the EHR quality data of the different sites and processing investigators requests
- A proxy cloud provider (PCP) responsible for the access control and for allowing an authorized investigator to decrypt the result of his query. We note that the two cloud providers must be independent (e.g., Amazon AWS and Google Cloud). Moreover, the proxy cloud provider can also be a private cloud administered by the networks sites or a coordination center as it only needs minimal computing and storage resources

2.2 Security and Privacy Requirements

We consider the different sites (S_i) in the network to be individually trusted for the management of their own data, but unwilling to share data with each other in the cloud without the proper security and privacy guarantees. In this ecosystem, we consider two main potential privacy/security threats:

- A honest-but-curious (or semi-honest) adversary at the storage cloud provider or at the proxy cloud

provider. This type of adversary honestly follows the protocol but tries to passively infer private information from the data stored at its premises. Such an adversary can be represented by a careless or disgruntled employee of the cloud providers (i.e., an insider) or a hacker who has gained illegitimate access to the data. Note that we assume the two cloud providers to not collude, that is, they cannot be compromised simultaneously by the same adversary.

- A malicious-but-covert adversary who wants to obtain sensitive information about one or several patients or probands by actively performing consecutive, but legitimate, queries to the system with the goal of re-identifying the presence of a person into one of the sites databases. Such an adversary can be represented by a malicious investigator or a hacker having legitimate access to the system and using it as an oracle in order to reconstruct the desired information.

The proposed solution must protect the confidentiality of sensitive data stored on the storage cloud provider (SCP) at rest and throughout the whole query work-flow against the honest-but-curious adversary described above. Any single point of failure in the system must be avoided and trust must shared across multiple entities. The investigator issuing the query must be the only entity able to obtain the final result. Moreover, the proposed system should also protect patients privacy by mitigating the risk of re-identification incurred from releasing EHR quality data against the malicious-but-covert adversary described above. We note that, the protection of query confidentiality as well as data integrity is not a requirement for the proposed system but can be integrated at a later stage.

2.3 Design and Work-flow

In this subsection, we will present the design of the solution. We will describe the data model and then the work-flow of a secure query.

2.3.1 Data Model

In order to be fully compatible with i2b2, we represent EHR quality data in a model that is complementary to the original i2b2 star schema (see Fig. 2). To this purpose, we propose a data model consisting of three aggregation attributes and a count. In particular, we use a table *encryptedtotalnums* with the following columns :

- *location_cd*: This column stores the code of the location at which quality data has been collected, e.g., a specific clinic within a site or a site.

- *concept_path*: This column stores the ontology path corresponding to the concept's node in the ontology tree, e.g., a clinical condition, a medication, a lab result, a procedure, etc..

- *time*: This column stores the time unit, which can be set at any interval (i.e., month, quarter, year) as long as the unit is consistent across the dataset.

- *totalnum*: This column stores the number of patients at location X and time unit Y who have an observation specified by the ontology concept Z.

This data model can provide a high degree of flexibility as the three aggregation attributes can be defined at any spatial level, time scale, and ontology concept of interest. The privacy-sensitive (hence confidential) information for clinical sites is represented by the patient count. In the following subsection, we describe how to protect this information in the cloud.

2.3.2 Work-Flow

We will first describe how the key pairs are generated, then how the clinical sites use it to put securely the data on the cloud and finally all the steps of how the investigator makes a secure query.

Key Generation In order to achieve privacy-preserving sharing of sensitive EHR quality data across different sites, the data must be encrypted under a shared encryption key. Such a shared key cannot be generated by a single authority as this would centralize the trust of the whole system in a single point of failure. Therefore, during the system setup, the two cloud providers, SCP and PCP , need to collaborate in order to create a shared public key K. In particular, after creating their own ElGamal key pairs, respectively (k_{SCP}, K_{SCP}) and (k_{PCP}, K_{PCP}), they combine their public keys in order to obtain $K = K_{SCP} + K_{PCP}$. Once generated, K is stored at both cloud providers.

Data Encryption Clinical sites can fetch the shared public key K from one of the two cloud providers and use it to encrypt their data. Before encryption, each site extracts clinical data stored in the i2b2 database and pre-aggregates them according to the new data model for EHR quality data. The *totalnum* described in the data model is also encrypted in the exact same way. We note that no single party alone, in the proposed system, can decrypt the encrypted patients counts stored at SCP . This is due to the fact that the secret key corresponding to the public shared key K doesn't exist in its full form as it is shared between the two cloud providers. Such a secret-sharing mechanism ensures trust decentralization and no single point of failure.

Query Generation The secure query work-flow starts with an investigator that wants to retrieve some aggregate information about sensitive EHR quality data

stored at the Storage Cloud Provider SCP . After authentication to the i2b2 Web Client, the investigator generates an ElGamal key pair (k_I, K_I) for the current session. Each request is sent along with the public key K_I.

Query Processing The SCP receives the query and runs it on the encrypted database in order to select the tuples that satisfy the query. Then, based on the homomorphic properties of the ElGamal encryption scheme, it aggregates the encrypted counts yielding an encrypted result $E_K(R) = (C_1, C_2)$.

Result Re-Encryption To be decrypted by the investigator, the encrypted query result $E_K(R) = (C_1, C_2) = (rG, R + rK)$ must be re-encrypted under the investigators public key K_I . To this purpose, the Crypto cell at the SCP starts a two-step re-encryption protocol with the correspondent Crypto cell at the PCP . The description of the protocol can be found in this technical report [4].

Result Obfuscation Depending on the investigators role and trustworthiness, PCP can obfuscate the query result by homomorphically adding some random laplacian noise in order to achieve differential privacy. The best strategy to do it will be explored in future work.

Result Decryption The decryption of the (obfuscated) query result takes place in the Web client by using the investigators secret key k_I as input of the ElGamal decryption algorithm.

3 Implemented Solution

In this section, we present how the design of the solution explained in the previous section can be implemented using different technologies.

3.1 Technologies used

In this subsection, we will present the different technologies used for every component involved in the overall design.

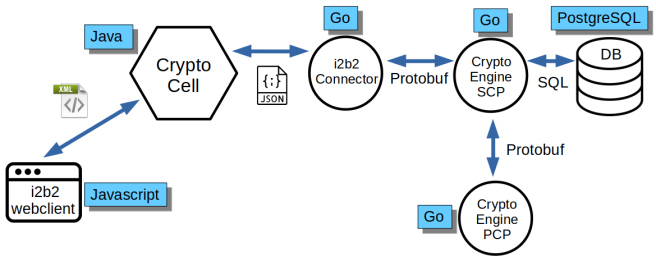

Figure 2: Overview of the main components and how they interact.

3.1.1 i2b2

i2b2 [1] is organized as a collection of different webservices that interact with each other as well as with the clients. The collection is called the hive and each webservice is called a cell. The cells are built with Java using a JBoss Application Server (now called WildFly). Most of them work with a postgreSQL database to store i2b2 data. They exchange this data using XML, providing REST endpoints to other cells and clients. In this project we use only 3 of these cells : The Project Management (PM) cell, The Ontology cell and the Crypto cell [6] that was created in this semester project and that we show in Figure 2. The PM cell is used to store all metadata of a project and manages the access control of the users. It also stores the URLs of all other cells and provides them to the client. The Ontology cell stores the so-called ontology tree : every concept (disease, diagnosis, etc) is represented as a node in a structured tree. The Ontology cell provides endpoints to obtain information about a concept, like getting the children concepts of that particular concept. The Crypto Cell is the new cell that we add in this project, it is responsible for integrating the other components of the design with the i2b2 system. To this end, the work-flow of some use cases will be modified to use the Crypto cell. The client doesn't interact immediately with the different endpoints of the cell, he first connects to a Apache Web server that provides him with the PM cell URL and a graphical user interface built with Javascript, HTML and CSS that we call the i2b2 Webclient [5]. But for readability and simplicity, we will describe the i2b2 Webclient as requesting directly the Crypto cell as shown in Figure 2.

3.1.2 Crypto Engine

The Crypto Engine [3] regroups the other components of the solution. Most of it was already implemented at the beginning of the project. The main goal was to integrate it with i2b2. The Crypto Engine was built in Go and provides different functionalities. The first one is to encrypt a table as described in 2.3.2 with the shared public key of the SCP and the PCP. In our implementation we use keys of 128 bits. Then the engine can be deployed as 2 servers, one on the SCP and one on the PCP. A PostgreSQL [2] database runs at the SCP and contains the encrypted table. A client can interact with the server deployed at the SCP to query the encrypted database using SQL. Only the leaves of the ontology tree are stored in the database, which means that if we want to get the total number of patient for an intermediate concept, we need to aggregate over all of its leaves. Since we use homomorphic encryption we can sum different encrypted total numbers and since we store the full path for each concept, finding all leaves of a specific intermediate concept can

easily be done with the SQL *LIKE* operator. The results can then be further aggregated if we want to sum over locations and/or time periods, then they are sent back to the client following the work-flow described in 2.3.3 and all data communication is done using Protobuf. In this project a new component was added to the Crypto Engine : the i2b2 connector. It is also built in Go and acts as a client to communicate using Protobuf with the Go server deployed at the *SCP*. But it is also a web server itself. It provides different REST endpoints with JSON data exchange to allow the i2b2 Crypto cell to query the encrypted database. The details of this REST API are provided in appendix A.

3.2 Use Case : Tree expansion with totalnums

In this subsection, we will focus on one use case in i2b2 that makes use of the new components (the Crypto cell and the i2b2 connector) : in the i2b2 Webclient, the user may want to get an idea of the number of patients matching some concept in the ontology tree before actually making a more precise query.

3.2.1 Requirements

The basic requirement is to display the total number of patients along each expanded concept in the tree view of the i2b2 Webclient. The total number of patients should be aggregated across all locations and all time periods. When the user clicks on a concept it expands the children concepts as usual but it must also show their total numbers of patients. The expansion should be fast (a few seconds at most) because it isn't a complete query that the user necessarily wants to make. It just gives him some insight about the concepts.

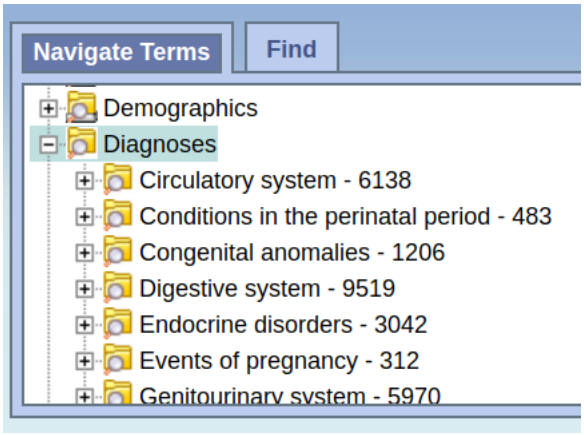

Figure 3: Screenshot of the expected end result. The total numbers of patients across all locations and time periods are shown alongside the child concepts.

Regarding security, it should comply with the requirements described in 2.2, which means the total numbers should be decrypted only in the i2b2 Webclient, every total number in transit or at rest in the cloud should be encrypted either with the client's public key or with the cloud providers' shared public key.

3.2.2 Design and Implementation

Request to the Crypto cell Prior to the implementation the work-flow was the following : the user clicks to expand the tree which sends a request to the Ontology cell to get the children of the concept. The resulting concepts are sent back to the client. So the first step of the implementation was to modify the i2b2 Webclient so that it sends the same request to the Crypto cell instead of the Ontology cell as shown in Figure 4. Since we will protect the confidentiality of the total numbers, the client also sends his public key K_I in the request. The public-private key pair (K_I, k_I) is generated at the beginning of the session, not at each request.

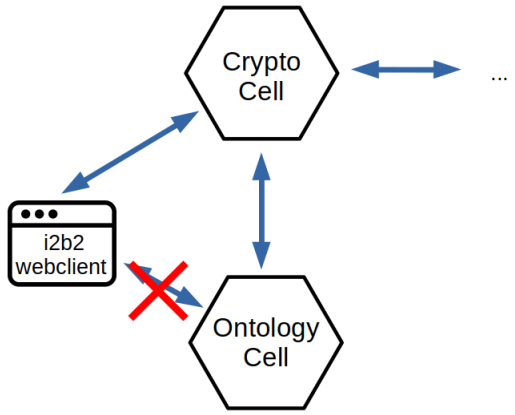

Figure 4: Overview of the modification of the workflow so that the Webclient requests the Ontology cell via the Crypto cell.

Request to the i2b2 connector So the Crypto cell implements a REST endpoint */getChildren* such that upon reception of such a request, it forwards it to the Ontology cell to get back the child concepts. Then the Crypto cell won't forward immediately the results back to the Webclient because we first need to insert the total numbers along each child concept in the XML message. This is where the Crypto Engine comes in : the i2b2 connector implements a REST endpoint */totalnum* that accepts a list of concepts and the client's public key K_I in JSON format and will return their corresponding total numbers. So the Crypto cell will send a request to the i2b2 connector with the results it got from the Ontology cell. If the requested concept is a leaf in the ontology tree, the Ontology cell returns no child concepts, in this case the Crypto cell returns directly an empty result to

the i2b2 Webclient and doesn't send a request to the Crypto Engine.

Inside the Crypto Engine As described above the i2b2 connector acts as a client to the server deployed on the *SCP*. It will send multiple queries in parallel, one for each of the child concept. This required to add implementation of concurrent capabilities in the server. For each concept c, the SQL query is of the following form :

SELECT *totalnum* FROM *encryptedtotalnums*
WHERE *concept_path* LIKE 'c'

We use the $LIKE$ operator to aggregate over all leaves as described in 3.1.2. We don't need to worry about *location_cd* and *time* since we aggregate over all locations and time periods. The results are then re-encrypted under the client's public key K_I with the help of the PCP as described in 2.3.3 and then sent back to the Crypto cell.

Results sent to the client and decrypted Once the Crypto cell gets back the encrypted total numbers, it inserts them along each concept in the XML received from the Ontology cell and sends it to the i2b2 Webclient which decrypts them with the client's private key k_I and displays them in the tree view.

3.3 Use Case : totalnums per location and time period

In this subsection, we will present another use case : the simple aggregated total number of patients might not be helpful enough for the user. In order to get more insight for future queries, the user may want to see more details about the total number of a given concept, depending on location and time.

3.3.1 Requirements

The goal is for the user to be able to right-click on any concept in the tree view of the i2b2 webclient and see 2 items in the context menu (figure 5) : the first one to see the different total numbers of patients matching the selected concept across all locations but aggregated over time, the second one to see the evolution over time of the total numbers of patients across all locations. So this use case is actually split in 2 subuse cases, but since they are very similar they could be implemented and presented as a single use case.

A click on each of these 2 items should open a new window with a graphical display of the data as well as buttons to refine the considered time range as shown in figures 6 and 7. We should also be able to choose which locations we want to display or not.

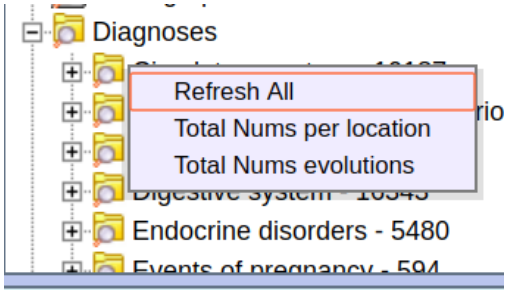

Figure 5: Screenshot of the expected end result. The user will have 2 more items in the context menu of any concept.

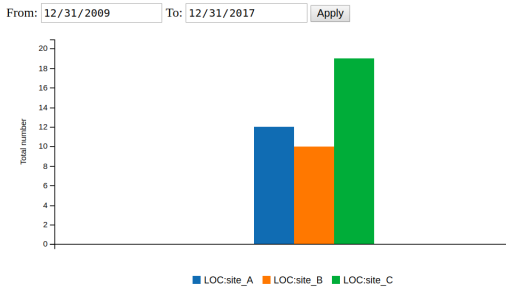

Figure 6: Screenshot of the expected window to be displayed by the item "Total Nums per location"

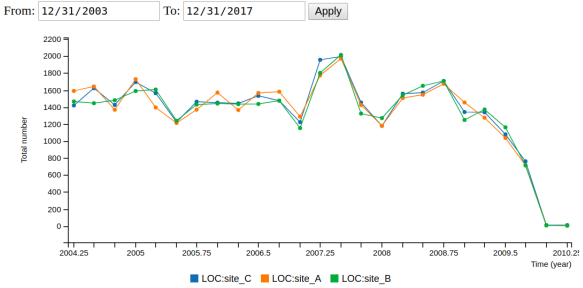

Figure 7: Screenshot of the expected window to be displayed by the item "Total Nums evolutions"

The security must still comply with the requirements described in 2.2, so the data should be decrypted only in the i2b2 Webclient. Obviously the information given by the first item is given by the second item, so we could have implemented only one item that displays everything in a single window. But it is better to split in 2 items because some users might not be allowed to see all the information but only a part of it. We will come back to this point in detail in the subsection 3.4.

3.3.2 Design and Implementation

Request to the Crypto cell Even if we add 2 items to the context menu, a click on each of them triggers a request to the same endpoint */getTotalNums* in the Crypto cell. The client sends XML data with again its public key K_I, the name of the selected concept c, 2 date-

times $from$ and to to specify the considered time range and we differentiate the 2 items by a field *distribution* in the XML which can take the values *point* (if the user wants the time evolution) or *cumulative* (if the user wants to aggregate over the time period).

Request to the i2b2 connector For this use case, the Crypto cell doesn't need to interact with the Ontology cell. It forwards the request in a JSON format to the i2b2 connector which implement a REST endpoint $/totalnums$. It is still useful to go through the Crypto cell for access control which will be described in subsection 3.4.

Inside the Crypto Engine Upon reception of the request, the i2b2 connector will make a single query for the selected concept c to the server. Since we want to consider only total numbers between $from$ andto, it required to extend capabilities of the server to handle such queries. Depending on the *distribution* parameter, the i2b2 connector will request the results to be grouped by location only (*cumulative*), or by location and time (*point*). The server will then make an SQL query of the following form to the postgreSQL database :

SELECT $location\_cd, (time,)totalnum$ FROM $encryptedtotalnums$ WHERE $concept\_path$ LIKE 'c' AND $time >= from$ AND $time <= to$ GROUP BY $location\_cd(, time)$

The results are again re-encrypted under the client's public key K_I using the same procedure described in 2.3.3. The i2b2 connector will send back the results to the Crypto cell in a JSON format : a list of elements containing a total number and a location code. If the *distribution* is *point*, it also contains a specific date time.

Results sent to the client and decrypted Basically the Crypto cell will forward the results back to the client who decrypts each total number of the elements. We then use this data and the Javascript framework C3 D3 to build a graphical display of the results, a bar chart in the case of the total numbers per location and a line chart in the case of the total numbers evolution over time.

3.4 Access Control

Clearly the time evolution gives more information to the user than the simple aggregation per location. And these two give more information than the simple total number displayed along each concept in the tree view that is aggregated over all locations and time periods. So in terms of security, the leakage of information should depend on the rights of the user. Some might be eligible to see all the information and some other might not be allowed to see all the details of the information but only an aggregation of it. So this requires access control.

3.4.1 Requirements

The goal is to control which users can access what information. To this end, a database should store different roles for each user. At login, the user rights are identified and only the information he is allowed to retrieve is displayed. So the total numbers in the tree view might not appear and the context-menu might display only one of the 2 items or none, depending on the roles of the user. Of course the front-end security is not enough since a malicious user could bypass the i2b2 Webclient and make requests directly to the Crypto cell, so there should also be a back-end access control.

3.4.2 Design and Implementation

User Roles Luckily the PM cell already implements access control functionalities. It works with a postgreSQL database which has a table called $pm\_project\_user\_roles$. This table stores all the different roles and which users have these roles. So we only needed to add roles for each user to control their access to the previous use cases. The following roles were added :

- TOTALNUMS : Every user with this role is allowed to see the total number of patients for each concept in the tree view of the i2b2 Webclient.
- TOTALNUMSCUMUL : In the context menu of each concept of the tree view of the i2b2 Webclient, every user with this role can see an item that will open a window displaying the total number of patients per location aggregated over a time period.
- TOTALNUMSTIME : In the context menu of each concept of the tree view of the i2b2 Webclient, every user with this role can see an item that will open a window displaying the evolution over time of the total number of patients for each location.

Different dummy users were added with the possible combinations of these roles for testing purposes.

Front-end Access Control At login, the i2b2 Webclient automatically requests the user credentials and roles to the PM cell. The PM cell then uses the table described above to send back the roles. This way we could use that information now stored in the i2b2 Webclient to choose whether to display or not the total number along each concept in the tree view and each of the 2 items in the context menu. As a malicious user could easily modify the Webclient to directly request the Crypto cell without caring about the role, back-end access control also had to be implemented.

Back-end Access Control Inside the Crypto cell, for each of the endpoints we now first look at the user credentials in the XML request and send a request to the PM cell to check both the credentials of the user and retrieve his roles as shown in Figure 8. If the user credentials are correct, the Crypto cell checks for each of the three kinds of requests if the list of roles contains the required role for that request. Only in this case it will proceed with the next steps as described in the previous subsections for the 2 use cases.

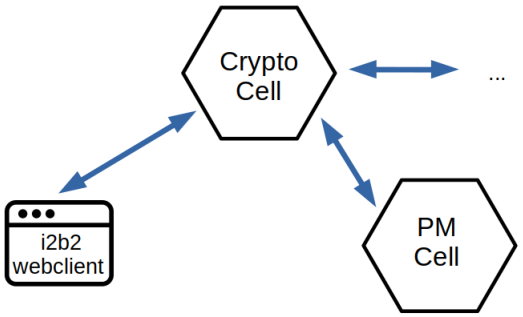

Figure 8: Overview of how we first query the user roles to the PM cell before proceeding to the next steps.

3.5 Results Obfuscation

In all the use cases we always returned the exact information, but for privacy protection reasons we might want to obfuscate these results. There are multiple ways to do it and is not a trivial task as we will discuss in subsection 5.1. This was considered out of the scope of this project, however we still implemented basic structure in the different components to be able to easily obfuscate the results in some future work.

3.5.1 Requirements

The basic requirement was to decide depending on the user roles whether some results will be obfuscated or not. This decision should be then sent to the Crypto cell and the Crypto Engine so that they can adapt the results accordingly.

3.5.2 Design and Implementation

User Roles We used the same technique described in the first paragraph of 3.4.2 and added 3 different roles : TOTALNUMS_NOISY, TOTALNUMSCUMUL_NOISY and TOTALNUMSTIME_NOISY. Each of these roles correspond to one of the role presented above. The only difference is that these roles specify that the user does not have the right to see the exact results but can see the obfuscated results.

Request to the Crypto cell The i2b2 Webclient will now check whether the user has the required role, either the one for exact results or the noisy results. For example, the total number along each concept in the tree view will be displayed if and only if the user has the role TOTALNUMS or the role TOTALNUMS_NOISY. In each XML request a new field called *noisy* is added. It is a boolean value which is set to true in case the role ends with _NOISY and to true otherwise.

Request to the i2b2 connector The Crypto cell will request the PM cell as before to obtain the user's roles and will allow the user if he has either the exact-results role or the noisy-results role. In the request to the i2b2 connector, the *noisy* field is still present in the JSON object.

Inside the Crypto Engine The i2b2 connector will send the request as before to the server but the query object is extended with the *noisy* field. The server then uses the field to obfuscate the results. One possibility is to add a new column called *noisytotalnum* in the table which will contain the corresponding obfuscated result for each total number of patients. At the time of the preparation of the SQL query statement, in case *noisy* is true, we will query *noisytotalnum* in the table, and *totalnum* in case *noisy* is false. For now, we query *totalnum* all the time since the creation of that new column is considered part of some future work.

4 Performance Evaluation

In this section, we will test the different use cases and evaluate the performance in terms of time of computation. We also talk about the storage overhead of the encrypted database.

4.1 Settings

For convenience we use only 2 machines $M1$ and $M2$. Their main hardware components are described below :

$M1$:

- CPU : Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz
- RAM : 16 GiB

$M2$:

- CPU : Intel(R) Xeon(R) CPU E31270 @ 3.40GHz
- RAM : 15 GiB

On $M1$ all i2b2 cells are running as well as the Apache Web server for serving the i2b2 Webclient, we also deploy the i2b2 connector and the *SCP* on $M1$. On $M2$ we

deploy the PCP. As query parallelization will be done on $M1$ it is important to note that its CPU configuration has 2 physical cores and 4 threads. The data set is a demo data set used only for testing purposes. The top concept of the ontology tree is called "Diagnoses" and we have 2176 leaves. As explained in 3.1.2, we only store the leaves and not the intermediary nodes. For each leaf we have different rows for the different time periods and locations but the number of rows per leaf concept is not constant, we have 3 different locations and 55 different time periods but not every concept is defined at every time period. In total we have 114'394 rows.

4.2 Tests

For each query made to the Crypto Engine for a given concept, the computation time depends on the position of the concept in the ontology tree : since we store only the leaves, selecting rows for a leaf concept doesn't require any aggregation but selecting rows for the top concept requires to aggregate over the whole tree. Aggregation aside, the computation time of the rows selection depends on the size of the dataset and the internal design of the DBMS used (PostgreSQL in our case). In the first use case, we make one such query per child concept, as we are able to parallelize queries, the overall computation time depends on the number of cores and threads of the hardware used (2 cores and 4 threads in our settings). For the second use case, we only have a single query to make.

Then for every resulting encrypted total number, the decryption time in the i2b2 Webclient depends on the size of that number since the current implementation of the lookup in the reverse mapping table performs a brute force search by incrementing the number. The overall computation time depends linearly on the number of these encrypted values that have to be decrypted sequentially, since for the moment there is no parallelization in the i2b2 Webclient.

All these considerations led to perform the following 3 tests :

- Test 1 : The overall time from the user clicking on a concept c until the total numbers are displayed along the child concepts. As discussed above, the overall computation depends both on the number of direct child concepts of c for the number of queries made, and on the number of all the children until the leaves of every direct child of c for aggregation. So we will test different concepts as c :
  - D : "Diagnoses" has 21 direct children and these 21 children have a total of 44'556 children combined.
  - P : "Diagnoses\Injury and poisoning\Poisoning" has 30 direct children that have a total of 1'096 children.
  - H : "Diagnoses\Hematologic diseases" has 10 direct children that have a total of 1077 children.
  - N : "Diagnoses\Symptoms, signs, and ill-defined conditions\Nonspecific abnormal findings" has 7 direct children and they have a total of 195 children.
  - E : "Diagnoses\Respiratory system\Other diseases\Empyema" has 2 direct children that are leaves so they don't have any children.
- Test 2 : The overall time from the user clicking on the context menu item of the concept "Diagnoses" for displaying the total numbers per location aggregated over a time period. The performance depends on the size of the selected time range : in each time range the total number of patients is defined for a finite set of points in time, the overall computation time depends on the number of these points and their values, as discussed above. So we will first try a time range from 31.12.2003 to 31.12.2017, there are 26 defined total numbers within this range. We will then try a time range from 31.12.2008 to 31.12.2017 in which we have 5 defined total numbers.
- Test 3 : The overall time from the user clicking on the context menu item of the concept "Diagnoses" for displaying the total numbers evolution for all locations. We will test with the same two time ranges as for the previous test.

For each test, we perform the same query 20 times and then compute the average and standard deviation. We also break down the computation time in parts to see how much time was taken for each task :

- $WC1$: time taken in the i2b2 Webclient from the moment when the user clicks until the request to the Crypto cell is sent.
- $CC1$: time taken in the Crypto cell from the reception of the request, at which point we request the roles to the PM cell, until reception and validation of these roles.
- $CC2$: time taken in the Crypto cell from the validation of the roles until we send the JSON request to the Crypto Engine. For the first use case, this consists of fetching the child concepts from the Ontology cell and parsing XML to JSON. For the second use case, this is only parsing XML to JSON.
- $CE1$: time taken in the Crypto Engine from the reception of the JSON request to the moment when we query the PostgreSQL database.

- $CE2$: time taken to process this query and get the SQL results back.
- $CE3$: time taken to aggregate the results using homomorphic encryption.
- $CE4$: time taken for the key switching protocol between the SCP and the PCP. In our setting, this is the only communication that actually goes over the Internet between the 2 machines $M1$ and $M2$.
- $CE5$: time taken from the moment when we computed the final results encrypted under the client's public key until the moment when we send the JSON response to the Crypto cell.
- $CC3$: time taken in the Crypto cell to process the response, parse it to XML and forward the i2b2 Webclient.
- $WC2$: time taken in the i2b2 Webclient from the reception of the response to decrypt the results and display them to the user.

4.3 Results

4.3.1 Test 1

Since we must compute the average and standard deviation for 10 tasks for 5 concepts, the exhaustive results would consist of 100 numbers. So for readability, as the computation time for $WC1$, $CE1$, $CE5$ and $CC3$ were not very significant, we omit them in the results shown in Figure 9. We provide the total computation times averages and standard deviation for every of the 5 concepts in table 1 in appendix C. The computation times $CE2_i$, $CE3_i$, $CE4_i$ happen in parallel for each subconcept's query i. So it is easy to find out their sum $CE2_i + CE3_i + CE4_i$ for each query i or to find out the overall computation time $T = CE2 + CE3 + CE4$ for all the queries to happen in parallel, but we cannot consider the sum of all $CE2_i$ to be equal to the global $CE2$ for all these queries since they happened in parallel. The computation time for a set of n parallel queries starts when the first query starts and terminates when the last query terminates. If we approximate and assume that all queries start exactly at the same time, the overall computation time T is equal to the computation time of the slowest query. So we obtain the following :

$$T = \max\{CE2_1 + CE3_1 + CE4_1, ..., CE2_n + CE3_n + CE4_n\} \leq \max\{CE2_1, ..., CE2_n\} + \max\{CE3_1, ..., CE3_n\} + \max\{CE4_1, ..., CE4_n\}$$

A proof of this inequality can be found in appendix B. So we choose to define our computation times the following way :

$$CE2 = \max\{CE2_1, ..., CE2_n\}$$
$$CE3 = \max\{CE3_1, ..., CE3_n\}$$
$$CE4 = \max\{CE4_1, ..., CE4_n\}$$

Even if mathematically this gives us an upper bound and not the exact results, the test showed a difference of less than 10 milliseconds between the observed T and the computed T.

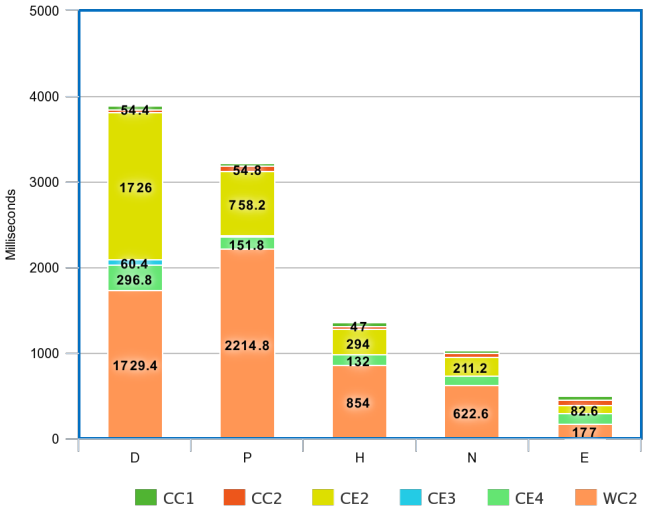

Figure 9: Main different computation times for Test 1. All parts that are too small to display the number have a computation time less than 1 millisecond.

As expected, the decryption time is greater for the concepts that have more direct child concepts. It is not exactly linear since the decryption depends also on the values to decrypt. We also see that the query time and aggregation time depend as expected on the total number of children of the direct children of the concept. So even if the decryption time for P is greater than the one for D (since P has 30 direct children and D 21), the query and aggregation times for D are greater than the ones for P because D's direct children have 444'556 children and P's direct children have 1096 of them.

4.3.2 Test 2

For a better readability, in Figure 10 we omit the computation times for $WC1$, $CC2$, $CE1$, $CE5$ and $CC3$ since they are not very significant and present only the averages for the 2 time ranges. In the tables 2 and 3 provided in appendix C we show the resulting average computation time and the standard deviation for the different components for the big time range (TR1) and the small one (TR2). All numbers are in milliseconds.

The query time and the aggregation time depend on the number of points defined over the time range. The decryption time is not affected by the time range because we aggregate over the time range, so in both cases

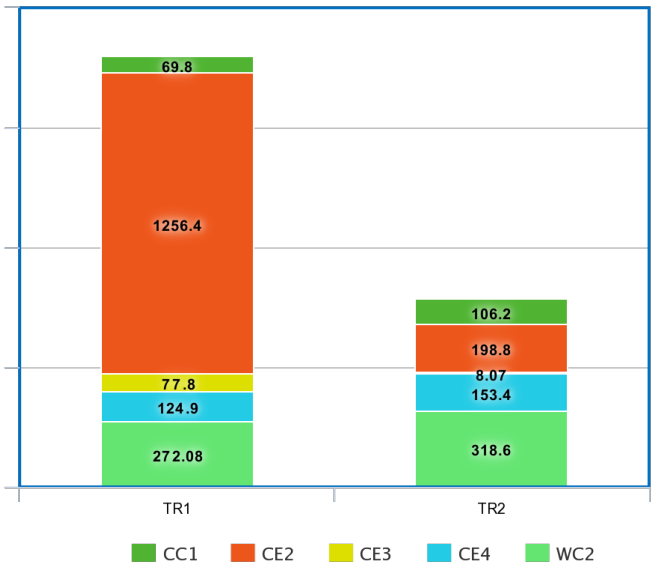

Figure 10: Main different computation times for Test 2.

we have the same number of values to decrypt (one per location).

4.3.3 Test 3

As before, for a better readability, in Figure 11 we omit the computation times for $WC1$, $CC2$, $CE1$, $CE5$ and $CC3$, and present only the averages for the 2 time ranges. In the tables 4 and 5 provided in appendix C we show the resulting average computation time and the standard deviation for the different components for the big time range (TR1) and the small one (TR2). All numbers are in milliseconds.

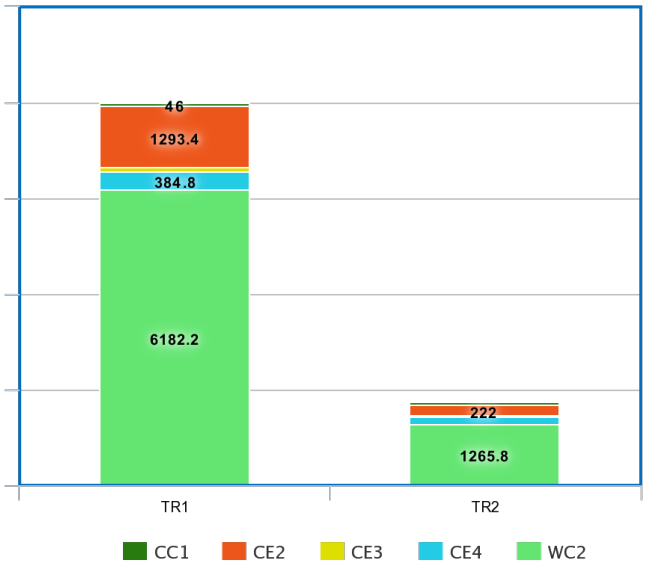

Figure 11: Main different computation times for Test 3.

The query and aggregation time still depend on the time range but for this test, we decrypt every point defined over the time range for every location. So the decryption is the bottleneck in this case.

4.4 Storage overhead

The security of the total numbers in the database comes at a cost : they are encrypted and the ciphertexts take more space than plaintexts. We will first analyze the theoretical overhead of a secure solution compared to a non-secure one, then we will apply the resulting formula to our implemented solution. In practice the overhead might be smaller depending on how the DBMS used optimizes the data storage.

Let p and c be the sizes in bits of the total numbers in plaintext and ciphertext respectively. We denote by f the size in bits of the other fields of the table (the concept path, the location and the time codes). For a table of n rows, the total overhead in bits is given by $O = n(c + f) - n(p + f) = n(c - p)$. The ratio of the secure solution compared to the non-secure one is given by $R = \frac{n(c+f)}{n(p+f)} = \frac{c+f}{p+f}$.

In our implementation, if total numbers were stored in clear as integers, assuming numbers are between 0 and 1000'000, we could use 32-bit integers. On the other hand, in our implementation of the el Gamal cryptosystem based on elliptic curves, ciphertexts have a size of 512 bits. The other fields of the table used (the concept path, the location and the time codes) are stored as strings. Together they make 757 characters, 6056 bits for 8-bit characters. So according to the formulas above, it gives us an overhead $O = 54909120$ bits $= 6.863$ MB, and a ratio $R = 1.07884362681$ so about 7.88% of additional data.

5 Future Work

5.1 Privacy Protection

The main goal of the project was to protect data from the cloud as well as in transit using homomorphic encryption. But the results are decrypted by the client who can be malicious and try for example to de-anonymize patients. In 3.5.2 we presented how we could add another column with noisy results, but no matter how we compute these results they would be completely static and the same for every user and every query. This lack of randomization could be a problem. We also didn't implement any privacy budget which means a user with the TOTALNUMSCUMUL role can completely reconstruct the time evolution by querying with different time ranges even if he doesn't have the TOTALNUMSTIME role. Differential privacy could be a counter-measure against this kind of attack by obfuscating the results and limiting

the number of authorized queries with a privacy budget. But the solution is not straight-forward since applying DP on every query independently would make the overall tree view inconsistent. We could obfuscate the leaves and then build the intermediate concepts bottom up but this would involve a lot of unnecessary computation since the user is not interested in the whole tree. Some future work could explore the feasibility of different solutions and find the best one to implement.

5.2 Parallel Decryption and Reverse Mapping

For every use case the results must be decrypted by the client. As explained in 2.3.1, after decryption EC ElGamal ciphers still need to be mapped back which is computationally expensive and affects the overall performance. To speed up this process we could look for some more efficient reverse mapping scheme and/or we could parallelize the process. Initially we used a Go library for cryptographic functions. Parallelization is easy in Go and there exists a concurrent version of the map used to reverse back the ciphers. The problem is that we need these functions in the i2b2 Webclient so we used the gopherjs tool to transpile from Go to Javascript and it cannot deal with concurrency at the moment. This may be because Javascript is fundamentally a single-threaded language even if there exists a library called Parallel.js for multi-core processing in Javascript. Some future work could find a solution to do parallel decryption and reverse mapping in Javascript.

```
{
  "conceptpaths": [
    "path1",
    "path2",
    "path3"
  ],
  "clientpublickey":
    "39XZGdiikSFqvkW8cSBy5FRrHpU00wEfe7VpAHp04Rg=",
  "noisy": false
}
```

Figure 12: Input format for the */totalnum* endpoint.

```
{
  "concepts": [
    {"conceptpath": "path1",
     "totalnum":
        "cFRdqXRWoc92rpdIMwhNedBswedJLk1YXviSwz
        Nru1lyrHJaDyfwRhDQBY3SL8wHVXfN0Biv7QG1H
        oMvMH+zgQ=="
    },
    {"conceptpath": "path2",
     "totalnum":
        "JhSVJqMnrIJ9RC9MnFJ7fcX5YxPJjsOTGxsfgJ
        nQONG02ClSFIzur9tyV4mjNFIO0jGyaTyAhaf
        +Xqd2KpEhLA=="
    },
    {"conceptpath": "path3",
     "totalnum":
        "wj8dhHvf19R4421ckErmkAwQwYckmNINIlXAJi
        cMgoXIn4iYmO4d+38OFOfjGbEtUqiP
        /OJDmVXvSs9QTrd0nw=="
    }
  ]
}
```

Figure 13: Output format for the */totalnum* endpoint.

Appendices

A API and deployment instructions for the i2b2 connector

A.1 API

The i2b2 connector provides a REST API with 2 GET endpoints. For each one, we show here the expected JSON format that must be sent in the HTTP request as input and the returned JSON as output.

/totalnum : It takes a list of concept paths as input and returns for each one the encrypted total number aggregated over all locations and time periods. We also need the client's public key as input for re-encryption and the *noisy* parameter as described in 3.5.2. Figure 12 shows the expected input format and 13 shows the output format.

/totalnums : From a concept path, it returns a list of groups each consisting either of a location code and a total number, or both a location code and a time period and a total number. As before, the client's public key and the *noisy* parameter are also part of the input. We have two more parameters as input, *fromtime* and *totime* so that we can specify the considered time range. Figure 14 shows the expected input format and 15 shows the output format for the case where we want the total numbers evolution over time.

```
{
  "conceptpath": "path",
  "clientpublickey":
    "39XZGdiikSFqvkW8cSBy5FRrHpU00wEfe7VpAHp04Rg=",
  "fromtime":"2010.5",
  "totime":"2016",
  "noisy": false,
  "distribution":"point"
}
```

Figure 14: Input format for the */totalnums* endpoint.

```
{
  "groups": [
    {
      "group": "LOC:site_C,2011.25",
      "totalnum": "M5zZsCwSkfy3s1iVtimrEgPG
        +rK+f6kj6Bt1OERcDPpfY18geqCey5uqpM2lo
        BBLnjgVDSUp5SjmUU6IlqfYZg=="
    },
    {
      "group": "LOC:site_A,2011.25",
      "totalnum":
        "yzGhI5vmATzZJgSLNPjpVAmNT0zrR49C4Qkl
        VU7TueX8n9y8QttKZOMUqlcMhQoGazyFWtLlG
        +MQsRrGoCCaig=="
    },
    {
      "group": "LOC:site_C,2012",
      "totalnum": "eI2D32voeDqNCyzFN
        +msaoHyZWoAqW5Vgo1hCex0pakHnUv8Arhvy8
        lwJLVzN4WrqUT+kEQUb4GALF7UCHM1vQ=="
    }
  ]
}
```


Figure 15: Output format for the */totalnums* endpoint.

A.2 Deployment instructions

The *SCP* and the *PCP* can be deployed as described in the repository's wiki of the Crypto Engine [3]. The i2b2 connector can be run with the following command once inside the directory app/test_local_deployment :

./../PDCi2b2 -d 1 cib -f group.toml -a private.toml

The group.toml file contains the addresses, port numbers and public keys of the *SCP* and the *PCP*. The private.toml file contains the address and port number of the i2b2 connector. This address and port should be known by the Crypto cell, they are defined and can be modified in the file CryptoService.java.

B Proof of the inequality used in 4.3.1

Let a_i, b_i, c_i be positive numbers $\forall i$ such that $1 \leq i \leq n$. We want to prove that :

$\max\{a_1 + b_1 + c_1, ..., a_n + b_n + c_n\} \leq$
$\max\{a_1, ..., a_n\} + \max\{b_1, ..., b_n\} + \max\{c_1, ..., c_n\}$

Proof :

We have that $\forall j$ such that $1 \leq j \leq n$,
$a_j \leq \max\{a_1, ..., a_n\}$, $b_j \leq \max\{b_1, ..., b_n\}$ and
$c_j \leq \max\{c_1, ..., c_n\}$.
So we have : $a_j + b_j + c_j \leq$
$\max\{a_1, ..., a_n\} + \max\{b_1, ..., b_n\} + \max\{c_1, ..., c_n\}$.
Since this holds for any j, it holds for the maximum over all j. So we have :
$\max\{a_1 + b_1 + c_1, ..., a_n + b_n + c_n\} \leq$
$\max\{a_1, ..., a_n\} + \max\{b_1, ..., b_n\} + \max\{c_1, ..., c_n\}\square$.

C Additional test results

Concept	Average	Standard deviation
D	3898	115.54
P	3225.99	62.63
H	1356.58	38.6
N	1038.43	42.93
E	495.66	47.35

Table 1: Total computations' averages and standard deviations in milliseconds for each concept in Test 1.

Component	Average	Standard deviation
$WC1$	0.01	0.1
$CC1$	69.8	15.73
$CC2$	5.2	3.83
$CE1$	0.696	0.232
$CE2$	1256.4	13.4
$CE3$	77.8	1.92
$CE4$	124.9	12.29
$CE5$	4	0.707
$CC3$	2	0
$WC2$	272.08	10.78
Total	1812.876	32.24

Table 2: Averages and standard deviations in milliseconds for TR1 in Test 2.

Component	Average	Standard deviation
$WC1$	0.02	0.021
$CC1$	106.2	23.34
$CC2$	3.8	3.03
$CE1$	0.784	0.69
$CE2$	198.8	26.22
$CE3$	8.07	2.45
$CE4$	153.4	42.24
$CE5$	3.8	1.09
$CC3$	2.4	0.54
$WC2$	318.6	45.66
Total	795.85	48.77

Table 3: Averages and standard deviations in milliseconds for TR2 in Test 2.

Component	Average	Standard deviation
$WC1$	0.01	0.14
$CC1$	46	7.07
$CC2$	1.8	1.3
$CE1$	0.538	0.24
$CE2$	1293.4	25.085
$CE3$	87.6	1.94
$CE4$	384.8	59.16
$CE5$	7.6	0.89
$CC3$	1.4	0.54
$WC2$	6182.2	49.16
Total	8005.338	129.083

Table 4: Averages and standard deviations in milliseconds for TR1 in Test 3.

Component	Average	Standard deviation
$WC1$	0.02	0.011
$CC1$	64.8	9.31
$CC2$	6	4.06
$CE1$	1.17	1.28
$CE2$	222	26.64
$CE3$	7.9	0.31
$CE4$	182.4	18.78
$CE5$	5.4	1.14
$CC3$	2.6	2.3
$WC2$	1265.8	16.26
Total	1758.07	42.03

Table 5: Averages and standard deviations in milliseconds for TR2 in Test 3.

References

[1] i2b2 tranSMART Foundation. *i2b2*. URL: `https://www.i2b2.org/`.

[2] The PostgreSQL Global Development Group. *PostgreSQL*. URL: `https://www.postgresql.org/`.

[3] Jean Louis Raisaro, Joao Andre Sa, Jeffrey G Klann, and Melchior Thambipillai. *Crypto Engine*. URL: `https://github.com/JLRgithub/PDCi2b2/tree/melchiorbranch`.

[4] Jean Louis Raisaro, Jeffrey G Klann, Kavishwar B Wagholikar, Hossein Estiri, Jean-Pierre Hubaux, and Shawn N Murphy. "Feasibility of Homomorphic Encryption For Sharing i2b2 Aggregate-Level Data in the Cloud". In: *AMIA Informatics Summit 2018, March 2018* ().

[5] i2b2 team and Melchior Thambipillai. *i2b2 Webclient*. URL: `https://github.com/mthambipillai/i2b2Webclient`.

[6] Melchior Thambipillai. *Crypto Cell*. URL: `https://github.com/mthambipillai/i2b2CryptoCell`.