**TECHNICAL REPORT**

1. **Problem presentation**

The main problem is search into a database encrypted specific information regarding the patient: such as determining the maximum age, calculating the sum of all ages of patients, etc. from a large encrypted database.

Other problems that we want to solve are:

* Protection of valuable data
* Improving performance
* Economics
* Modularity
* Distributed query processing can improve performance

Private statistics is an application which creates statistics about patients' medical information keeping their data private. The application is based on homomorphic encryption which makes possible operating on data without decrypting it. To enable sharing information with privacy we will use private set operations that allow two or more parties to compute operations on their sets (union, intersection etc).

1. **State-of-the-art**

Database encryption can generally be defined as a process that uses an algorithm to transform data stored in a database into "cipher text" that is incomprehensible without first being decrypted. It can therefore be said that the purpose of database encryption is to protect the data stored in a database from being accessed by individuals with potentially "malicious" intentions. The act of encrypting a database also reduces the incentive for individuals to hack the aforementioned database as "meaningless" encrypted data is of little to no use for hackers. There are multiple techniques and technologies available for database encryption .

The algorithm that we've used to encrypt data is Paillier, a probabilistic asymmetric algorithm for public key cryptography, partially homomorphic cryptosystem. This algorithm provides the union encrypted texts and multiplication on encrypted text with unencrypted text.

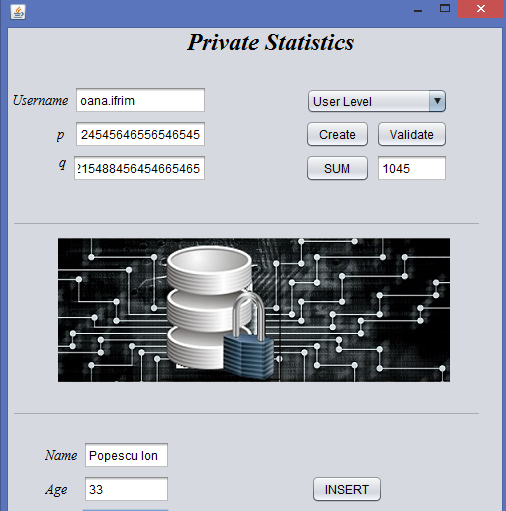
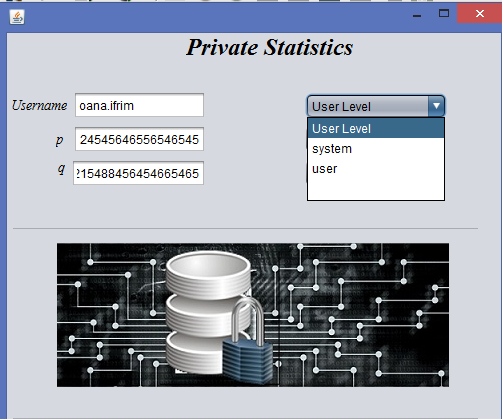
1. **Your solution**

In order to find a solution to our problem, we created a database that stores information about the patients( name, age, etc).

As security protocols, the PM Semi-Honest protocol was implemented. This permits two entities (for example a client and a server) which each has a list of values, to calculate the intersection of the two lists, without leaking extra data.

The client is the one which learns about the common values with the server. There are 2 types of adversaries: semi-honest adversaries and malicious adversaries. The PM Semi-Honest protocol adopts the first model, in which each entity respects the rules of the protocol and neither replaces the input with a random value or exits the protocol premature.

As an optimisation the Neville algorithm was used to calculate the coefficients of a polynomial and not the usual function of polynomial interpolation.



1. **Results, Evaluation**

We succeeded to create the encrypted database and use it.

We succeeded to run INSERT and SELECT statements on our database.

For the INSERT the interrogation is sent to the user (application). The data will be encrypted at the user interface level, and the rest of the modules will work with only the encrypted data. Each insert or select from the database query will retrieve encrypted data.

Also, the application offers the possibility of data validation, provided the keys are correct and different levels of user account types based on key length and certainty. Paillier is an asymmetric encryption scheme and offers better protection of data, depending on the key length. The default key length of a standard user is 512 bit key, a user of level 1 has a 1024 key and a user of level 2 has a key of 4096 bit key.

The sum offers the possibility to compute a sum of BigInteger values without decrypting the data on the server. The values are added on the server side and the response is decrypted on the user interface side. The value is displayed on the UI.

After we finished what we proposed to do, we ran a few tests on different parameters: number of clicks, time and type of users.

We ran 3 tests, 1 for each user type, with database insertion and retrieved the following values:

* For a user of type system it took 630239853 nanoseconds (630 milliseconds)
* For a user of level 1 it took 577437463 nanoseconds (577 milliseconds)
* For a user of level 2 it took 8543891573 nanoseconds (8543 milliseconds)

The tests were run on an Intel i7-6500U @2.5GHZ (dual core) with 16GB DDR3 and generated 10 random numbers that were encrypted and inserted in the database.

1. **Comparison with other solutions**

The application uses a partial homomorphic encryption algorithm which does not offer multiplication on encrypted data. Mylar[4] is an experimental research platform for building web applications with end-to-end encryption. It was developed at MIT and uses CryptDB for safe data storage. CryptDB[5] is a mysql build that offers multilayer, fully homomorphic encryption on the data, tables, views etc that is stored in the database. The application offers feasible speeds for encryption/decryption of data, to be used in current applications.

Helib[6] is a fully homomorphic encryption library that offers strong security and a wide range of applications, but is slow at computation.

1. **Future work**

The application can be extended by making possible the usage of joins or other types of interrogations like: creating a new table, updating tables or altering tables with different information from patients like: disease sufferers.

This helps us at generating statistics regarding the frequency of illness occurrence, for a specific age etc.

For now, the application can be used by two types of user. It would be necessary to be used by other persons, like accountants for example, but only for data visualization (just the select buttons to be used).

1. **Conclusions**

In conclusion, we can say that the application solves some problems that we enumerated on the beginning, like data protection, encrypted queries, and modularity.

The application performance can be improved by using multithreading for parallel computation of large numbers, implement more values

1. **Bibliography**
2. [**https://www.microsoft.com/en-us/research/wp-content/uploads/2015/11/ManualHE.pdf**](https://www.microsoft.com/en-us/research/wp-content/uploads/2015/11/ManualHE.pdf)
3. [**https://www.usenix.org/system/files/conference/usenixsecurity14/sec14-paper-pinkas.pdf**](https://www.usenix.org/system/files/conference/usenixsecurity14/sec14-paper-pinkas.pdf)
4. [**https://eprint.iacr.org/2011/141.pdf**](https://eprint.iacr.org/2011/141.pdf)
5. [**https://eprint.iacr.org/2014/336.pdf**](https://eprint.iacr.org/2014/336.pdf)
6. [**http://eprint.iacr.org/2015/133.pdf**](http://eprint.iacr.org/2015/133.pdf)
7. [**http://eprint.iacr.org/2012/099.pdf**](http://eprint.iacr.org/2012/099.pdf)
8. [**http://eprint.iacr.org/2011/277.pdf**](http://eprint.iacr.org/2011/277.pdf)
9. [**https://crypto.stanford.edu/craig/craig-thesis.pdf**](https://crypto.stanford.edu/craig/craig-thesis.pdf)
10. **Links**

**Documentation links:**

1. [**http://www.pinkas.net/PAPERS/FNP04.pdf**](http://www.pinkas.net/PAPERS/FNP04.pdf)
2. [**https://www.cs.cmu.edu/~leak/papers/set-tech-full.pdf**](https://www.cs.cmu.edu/~leak/papers/set-tech-full.pdf)
3. [**http://www.csee.umbc.edu/~kunliu1/research/Paillier.html**](http://www.csee.umbc.edu/~kunliu1/research/Paillier.html)
4. [**https://css.csail.mit.edu/mylar/**](https://css.csail.mit.edu/mylar/)
5. [**http://css.csail.mit.edu/cryptdb/**](http://css.csail.mit.edu/cryptdb/)
6. [**https://github.com/shaih/HElib**](https://github.com/shaih/HElib)

**Application links:**

1. **GitHub:**

**<https://github.com/hajimemasho/taip2016/tree/master/PrivateStatisticsProject>**

**Members: Prof coord.**

**Cojocaru Carmen prof. Sorin Iftene**

**Ifrim Oana**

**Burghelea Dragos**