Project - Private Information Retrieval (PIR)

This project aims to implement a simple Private Information Retrieval (PIR) protocol. The goal of a PIR protocol is to allow an entity (e.g., a client) to retrieve information from a database of another entity (e.g., a server) without revealing this information to the latter (e.g., the server). In this project, we will represent a database as a table.

Let us take the example of a server S holding the table T with 1000 elements. A client C then would like to retrieve the n-th = 100-th element from the table without revealing the value of n to S. A straightforward solution is to let S send the whole table to C. However, when the data size is consequent, and the table T has many elements, this solution quickly becomes heavy regarding communications. In addition, if the table is frequently updated, the clients would have to ask for the updated table every time they want to retrieve information from it.

In this project, we will develop a PIR protocol based on homomorphic encryption.

1 Paillier Cryptosystem

The Paillier cryptosystem works as follows:

- KeyGen(`nb_bits`)
  1. Generate prime numbers p and q of `nb_bits` bits each, ensuring they have the same bit length. If p and q are of the same length, they must satisfy:
  $$\gcd(p \cdot q, (p-1) \cdot (q-1)) = 1.$$
  2. Compute $n = p \cdot q$, $g = n + 1$, $\lambda = (p-1) \cdot (q-1)$, and $\mu = \lambda^{-1} \mod n$.
  3. Output the public key $pk = (n, g)$ and the private key $sk = (\lambda, \mu)$.
- Encrypt$(pk = (n, g), m))$
  1. Ensure that $0 \leq m < n$ (message must be smaller than n).
  2. Generate a random r such that $0 < r < n$ and $\gcd(r, n) = 1$.
  3. Compute the ciphertext:
  $$c = g^m \cdot r^n \mod n^2.$$
  Output c.
- Decrypt$(sk = (\lambda, \mu), c))$

1. Compute the message:

$$m = L(c^\lambda \mod n^2) \cdot \mu \mod n,$$

   where $L(x) = \frac{x-1}{n}$ (quotient of Euclidean division).
2. Return m.

1.1 Note on the implementation

For the implementation of the Paillier scheme described above, you would work with arithmetic operations of big numbers (usually $\sim$ 2048 bits number). For this, we will use `gmpy2` Python library to perform these arithmetic operations. The library document is available here: `https://gmpy2.readthedocs.io/en/latest/mpz.html`.

In particular, we will use

- `mpz` class for the numbers.
- `gmpy2.invert` for the inversion.
- `gmpy2.f_divmod` for the division.
- `gmpy2.powmod` for the exponentiation.
- ... (see the document for more).

1.2 Questions

1. Implement the Paillier encryption scheme as described above in a file `paillier.py`. Ensure that the `assert` at the end succeeds if the implementation is correct.
2. Given ciphertexts c_1 and c_2 of messages m_1 and m_2, what can you say about $c_1 \cdot c_2 \mod n^2$? Test your answer by adding an adequate test case in the file `paillier.py`.
3. Given a ciphertext c_1 of a message m_1 and a plaintext message m_2, what can you say about $c_1 \cdot g^{m_2} \mod n^2$? Test your answer by adding an adequate test case in the file `paillier.py`.
4. Given a ciphertext c_1 of a message m_1 and a plaintext message m_2, what can you say about $c_1^{m_2} \mod n^2$? Test your answer by adding an adequate test case in the file `paillier.py`.

2 PIR Protocol

We will construct a simple PIR protocol as follows. We will denote by T our database table of n elements and by $\mathsf{Encrypt}(m)$ the encryption of the message m under the Paillier scheme. We will denote by $T[i]$ the element of index i in the table T.

The client wishing to retrieve information from the server will start by generating a Paillier key pair (pk, sk).

- The client wishes to retrieve the element of index i in T from the server. For this, the client generates a vector $v = (c_0, \ldots, c_{n-1})$, where $c_j = \mathsf{Encrypt}(0)$ for any $j \in \{0, \ldots, i-1, i+1, \ldots, n-1\}$ and $c_i = \mathsf{Encrypt}(1)$. The client then sends v and its public key $pk = (n, g)$ to the server.

- The server receives $v = (c_0, \ldots, c_{n-1})$ and pk from the client. Using pk, v, and T, the server homomorphically computes the following value:

$$t = \mathsf{Encrypt}(0 \cdot T[0] + \cdots + 0 \cdot T[i-1] + 1 \cdot T[i] + 0 \cdot T[i+1] + \cdots + 0 \cdot T[n-1]) \quad (1)$$

- The server then sends t back to the client.

- The client receives t and decrypts it using its private key sk.

2.1 Questions

1. How to homomorphically compute the value of t? Describe the necessary operations.

2. Which value does the client retrieve after decrypting the answer from the server?

3. Create the file `client.py`. In this file, write a class `Client` that has three methods:

   - The constructor which generates the Paillier cryptosystem associated with the client, for a number of bits given as input to the constructor.
   - The `request` method which takes as input the size of the database of the server, and the index of the element to be retrieved, and outputs the request to be sent to the server. Think of what exactly is output in this request in terms of what is necessary for the server to compute the result.
   - The `decryptAnswer` method which takes as input the answer of the server and decrypts to retrieve the value.

4. Create the file `server.py`. In this file, write a class `Server` that has a database table as an attribute with a given fixed size as input to the constructor of the class. The instance of `Server` then generates the table T by creating every element in it randomly in the range $[1; 2^{16}]$. The `Server` class typically has two methods :

   - the constructor which generates the table of a size given as input;
   - the `answerRequest` method which answers the request of the client by doing the necessary computation. (You need to think of what this function takes as input).

5. Create the file `exchanges.py`. In this file, instantiate a client and a server by generating a database of a size of your choice. Then simulate a request between the client and the server for a given index in the database, and retrieve the answer. Using an `assert`, make sure that the retrieved value corresponds to the value in the corresponding index from the database.

6. For a fixed number of bits for the Paillier scheme (e.g., 1024 bits per prime number), measure the execution time on the server side and then on the client side, for an increasing database size. You can start with a database of 10 elements, then 20, then 30, etc. From each of the measurements you get (for the client and the server), trace a curve where the x-axis corresponds to the size of the database, and the y-axis corresponds to the execution time. What is the impact on the efficiency for the server and the client as we increase the database size ?

7. Compute the size of the communications in bits from the client to the server and from the server to the client, for a fixed number of bits for the paillier scheme (e.g., 1024 bits per prime number), and an increasing database size. From the measurements you get, trace a curve where the x-axis corresponds to the size of the database, and the y-axis corresponds to the size of the communications (for the server and the client separately). What is the impact on the communications efficiency between the client and the server as we increase the database size ?

8. What are the memory and efficiency drawbacks of this PIR method ?

3 Deliverables

You need to submit all of your source code, as well as a report answering the different questions when needed. The clarity of the report and the code are strongly taken into account for the final grading. The report only needs to answer the questions that need more details. The format can be as simple as a file with a list of answers to the list of questions, as long as the answers are clear. Also, when the questions ask for curves to be traced, these curves are expected in the report (in form on a screenshot is acceptable).

The deadline is indicated in the submission page on `learning.devinci.fr` of your group.