8/24/2021

Mitali Thorat

Exposys Data labs

Internship Project

On Software Development

Abstract:

Let us consider a case where the user sends a text or file from his phone to his friend. The risk of the data being accessed by a third party is high when the data enters the cloud platform. Hence a new method to secure the messages/file is to be done either by building a new algorithm or by modifying the existing one which can give more security and consume less time.The identity of the user must also be verified by using the authentication, verification and validation methodologies.

Table Of Contents:

1. Introduction
2. Existing Method
3. Proposed method with Architecture
4. Methodology
5. Implementation
6. Conclusion

Introduction:

Cyber security and privacy breaches is a growing concern across the world today. If data gets into the wrong hands and we altered before it reaches the right destination and can have tremendous effects on organizations and people. What if I told you there is a way to keep your privacy and at the same time mind data? One of the ways this can be achieved is by using cryptography or in simple terms encryption and decryption of data. In this project, we will change an existing algorithm to secure data send by the user and also verfify the identity of the other user. We will use python and ML to perform the task.

Existing Method:

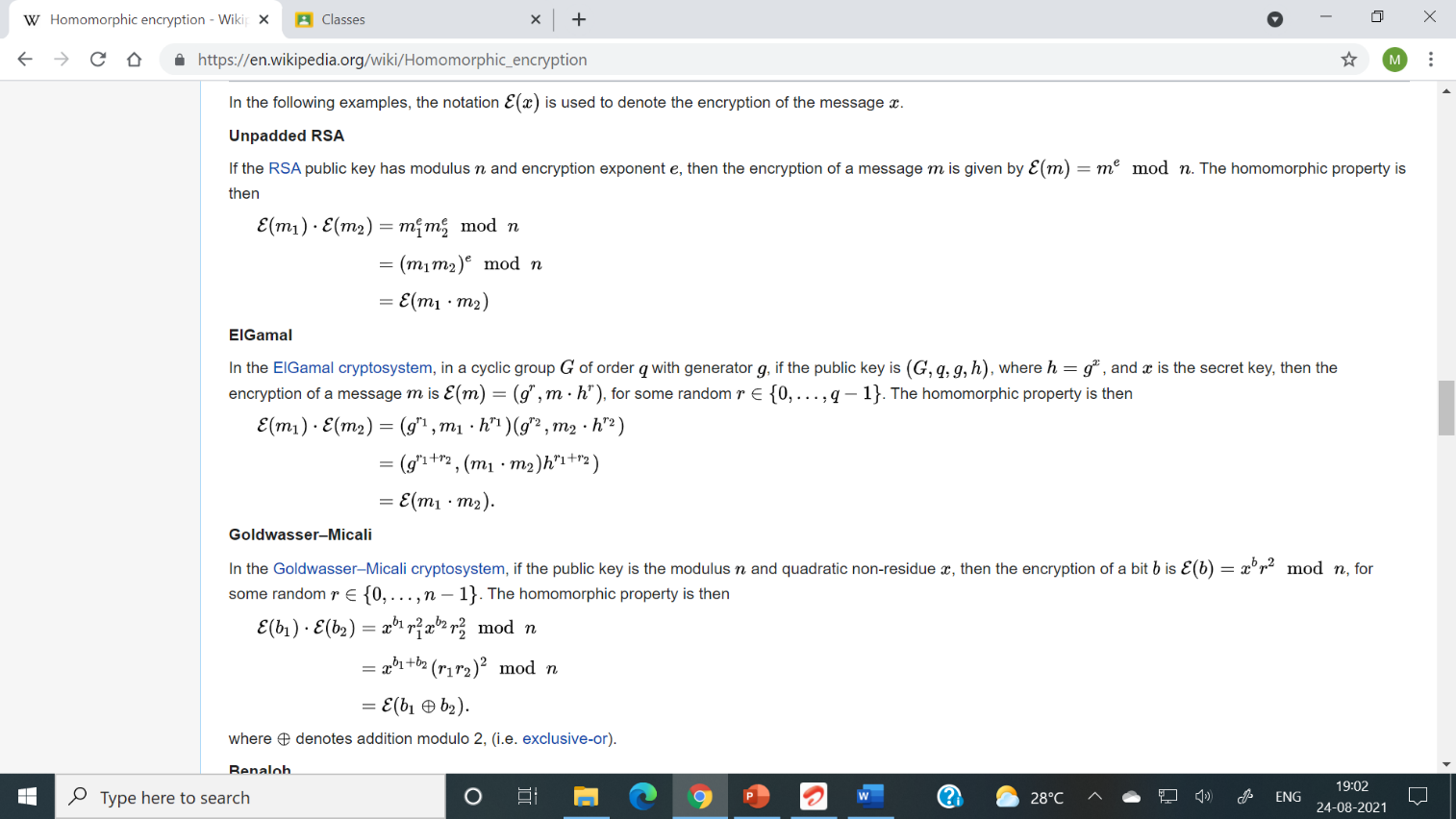
**Homomorphic encryption** is a form of encryption that permits users to perform computations on its encrypted data without first decrypting it. These resulting computations are left in an encrypted form which, when decrypted, result in an identical output to that produced had the operations been performed on the unencrypted data. Homomorphic encryption can be used for privacy-preserving outsourced storage and computation. This allows data to be encrypted and out-sourced to commercial cloud environments for processing, all while encrypted.

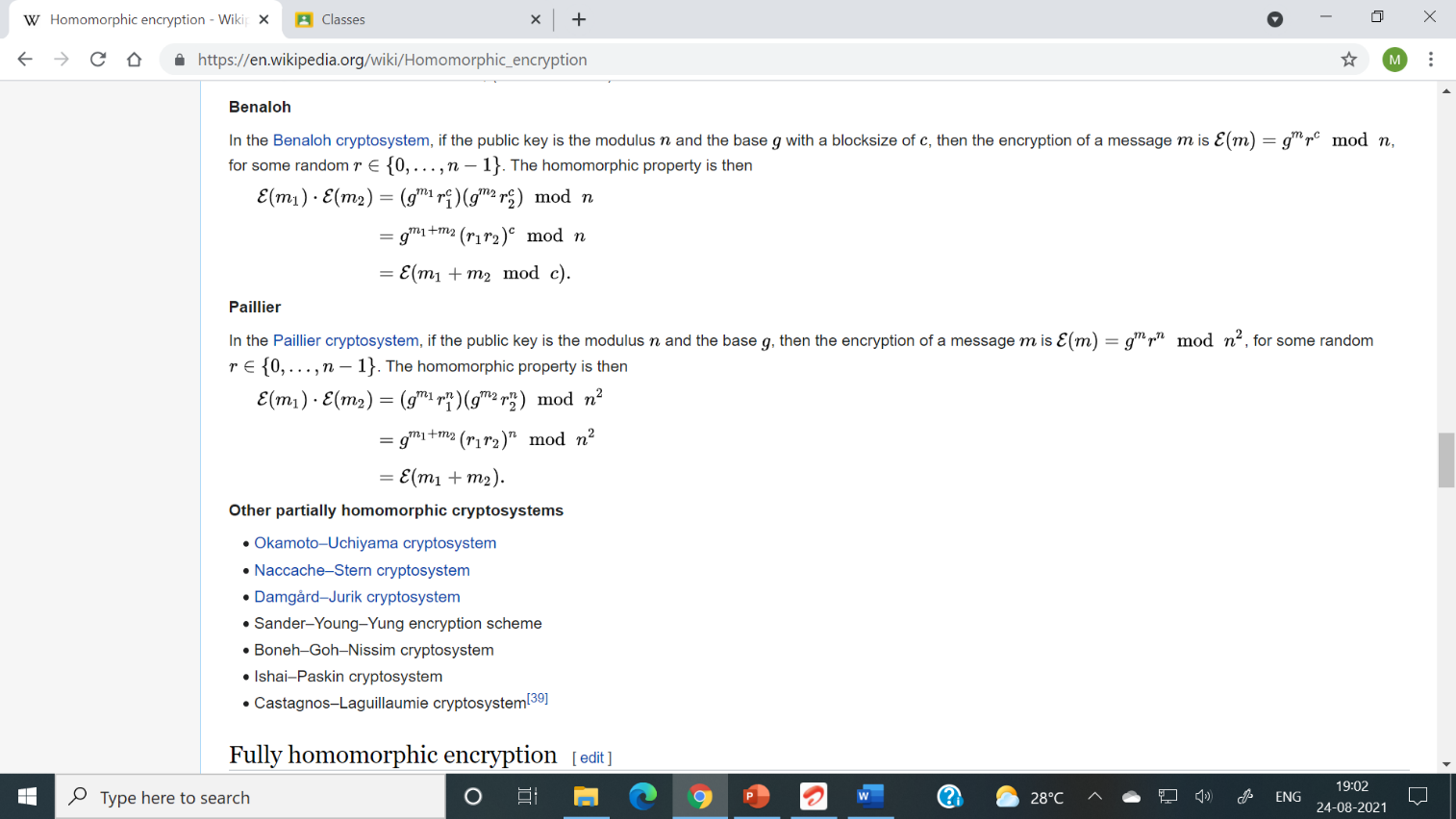
For sensitive data, such as health care information, homomorphic encryption can be used to enable new services by removing privacy barriers inhibiting data sharing or increase security to existing services. For example, predictive analytics in health care can be hard to apply via a third party service provider due to medical data privacy concerns, but if the predictive analytics service provider can operate on encrypted data instead, these privacy concerns are diminished. Moreover, even if the service provider's system is compromised, the data would remain secure.

Homomorphic encryption is a form of encryption with an additional evaluation capability for computing over encrypted data without access to the secret key. The result of such a computation remains encrypted. Homomorphic encryption can be viewed as an extension of public key cryptography. *Homomorphic* refers to homomorphism in algebra: the encryption and decryption functions can be thought of as homomorphisms between plaintext and ciphertext spaces.

Homomorphic encryption includes multiple types of encryption schemes that can perform different classes of computations over encrypted data.[[1]](https://en.wikipedia.org/wiki/Homomorphic_encryption#cite_note-ABG15-1) The computations are represented as either Boolean or arithmetic circuits. Some common types of homomorphic encryption are *partially* homomorphic, *somewhat* homomorphic, *leveled* *fully* homomorphic, and *fully* homomorphic encryption

Partially homomorphic encryption

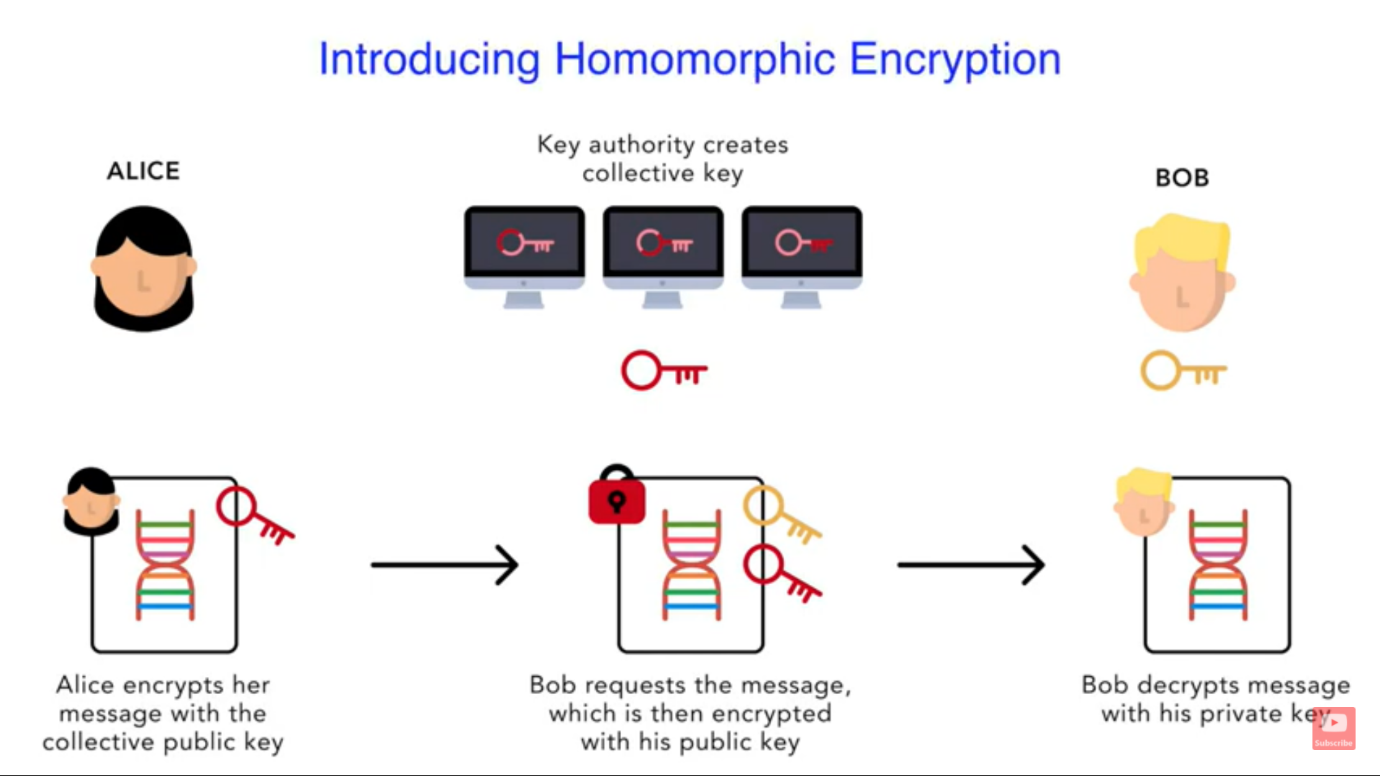




Fully homomorphic encryption {\displaystyle {\begin{aligned}{\mathcal {E}}(m\_{1})\cdot {\mathcal {E}}(m\_{2})&=(g^{m\_{1}}r\_{1}^{n})(g^{m\_{2}}r\_{2}^{n})\;{\bmod {\;}}n^{2}\\[6pt]&=g^{m\_{1}+m\_{2}}(r\_{1}r\_{2})^{n}\;{\bmod {\;}}n^{2}\\[6pt]&={\mathcal {E}}(m\_{1}+m\_{2}).\end{aligned}}}

A cryptosystem that supports arbitrary computation on ciphertexts is known as fully homomorphic encryption (FHE). Such a scheme enables the construction of programs for any desirable functionality, which can be run on encrypted inputs to produce an encryption of the result. Since such a program need never decrypt its inputs, it can be run by an untrusted party without revealing its inputs and internal state. Fully homomorphic cryptosystems have great practical implications in the outsourcing of private computations, for instance, in the context of cloud computing.

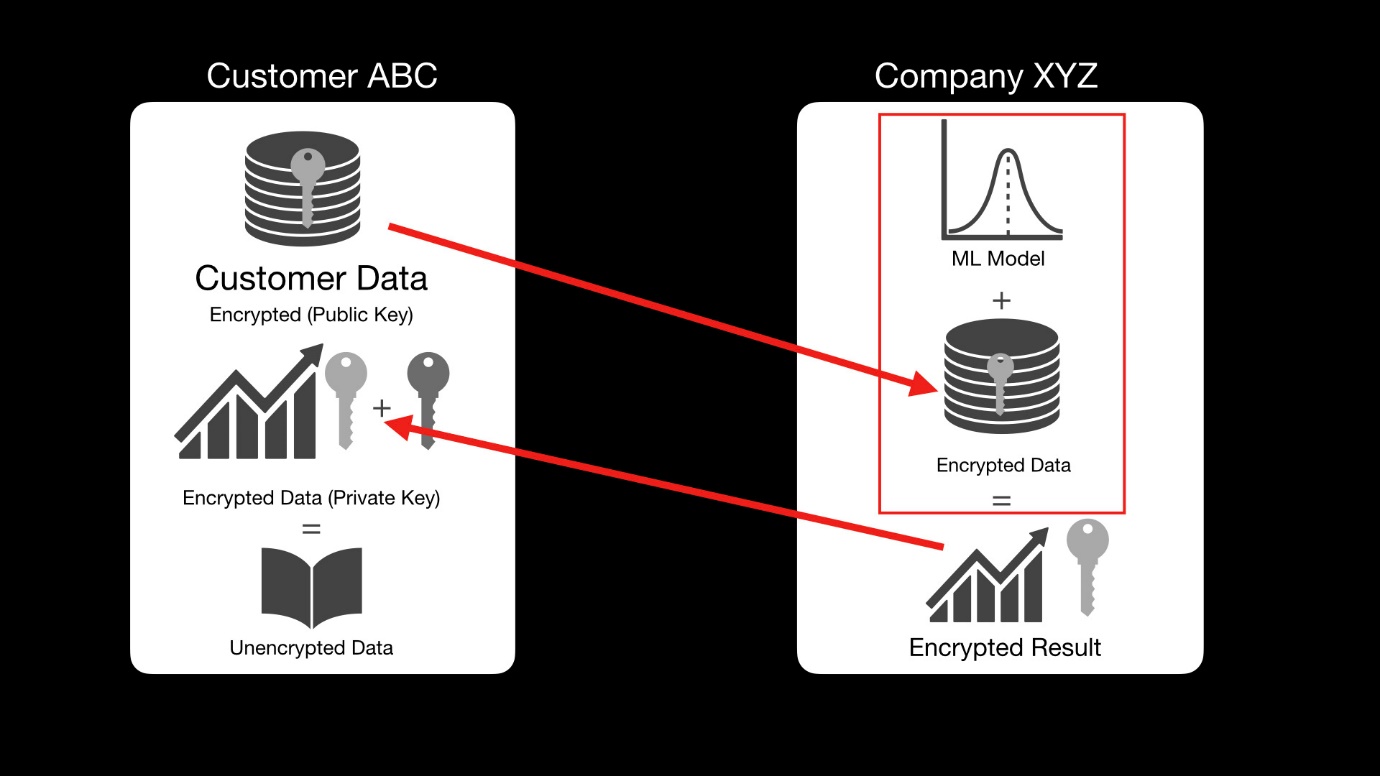
Architecture:



Methodology:

Customer generates encrypted data using a public key. Customer then sends this encrypted data to the Company along with the public key. The data send is in encrypted format and is hidden from the company. The company takes its ML model and applies it against the encrypted data. Which will produce the required result that is also in the encrypted form.

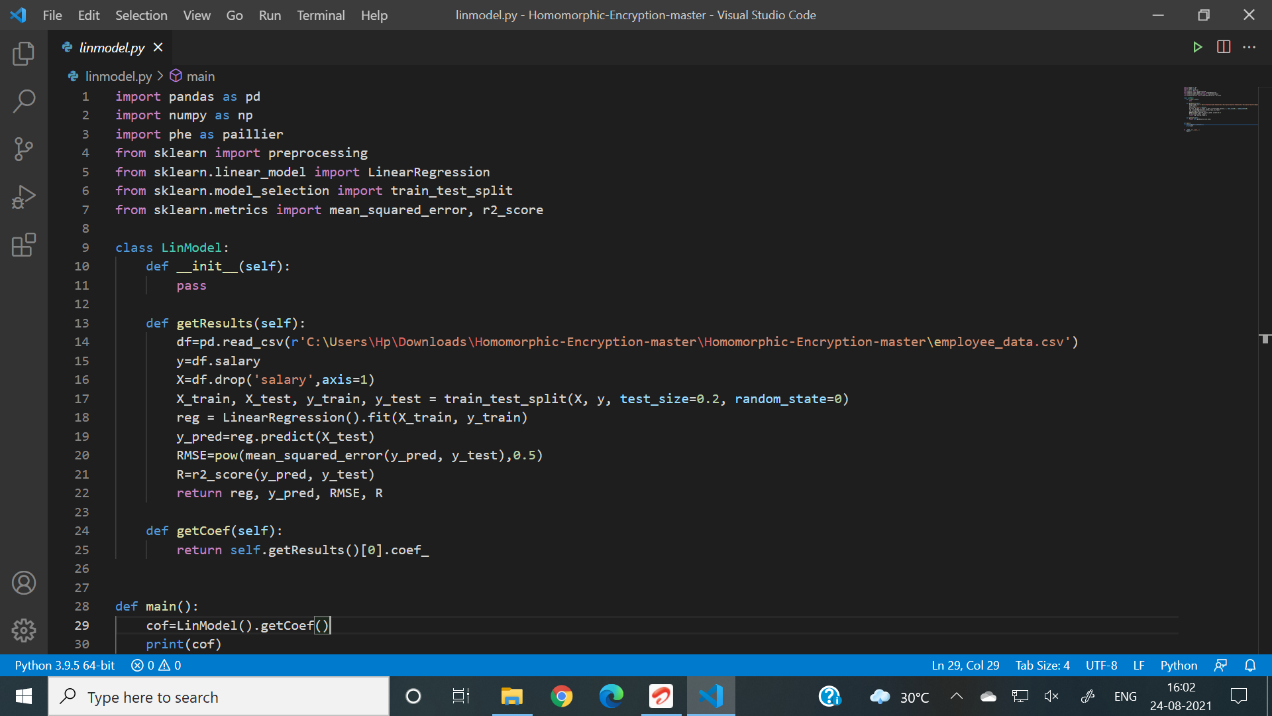
This encrypted result is send to the customer along with the previously received public key. The customer then matches this public key with his own public key. If both the keys match the customer uses his private Key to decrypt the data and receive the result.

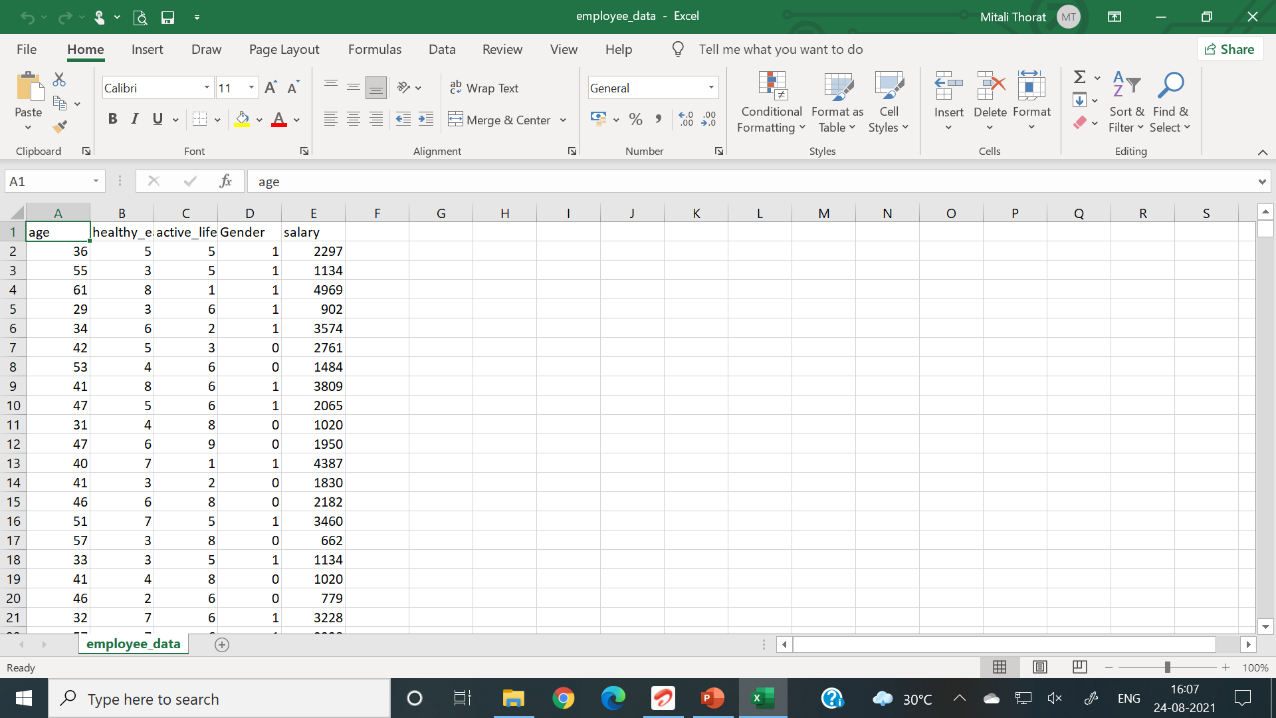


Implementation:

We are going to build a project where we predict someone’s salary based on their age, healthy eating, active lifestyle and gender.

Firstly we create a file called linmodel.py, this will be the company’s ML model, in this model we will read data from employee\_data.csv file.

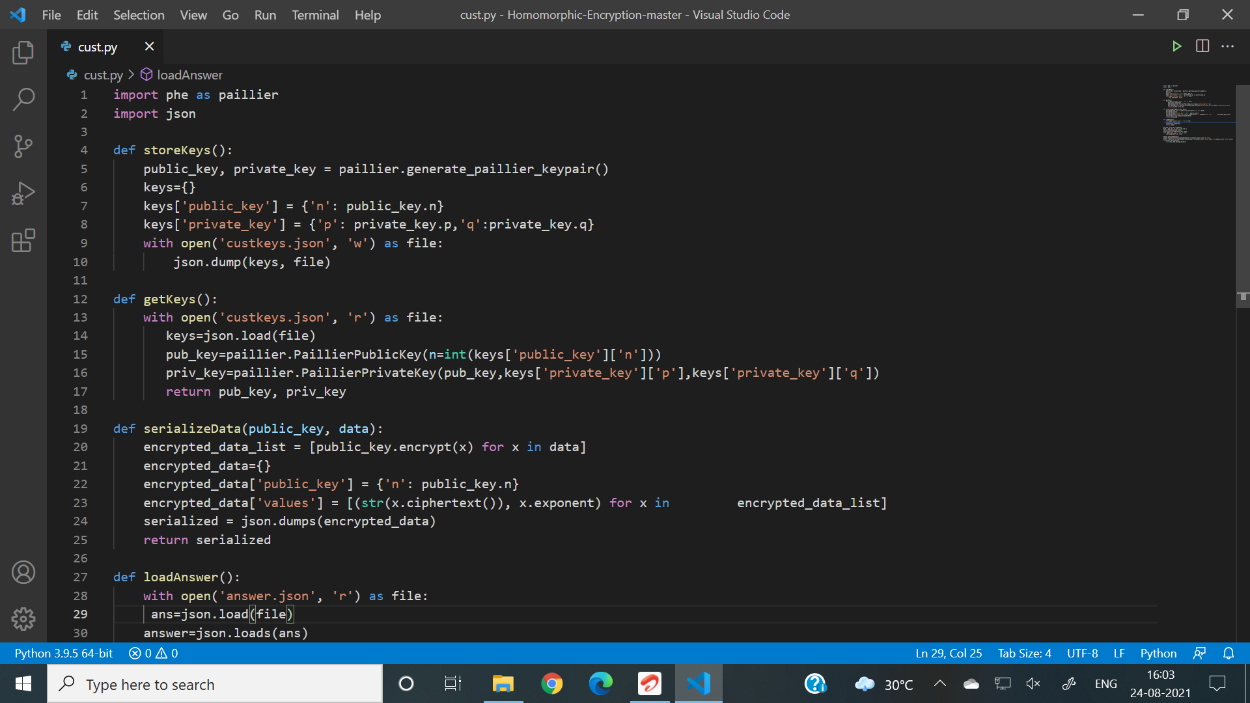
 Linmodel.py



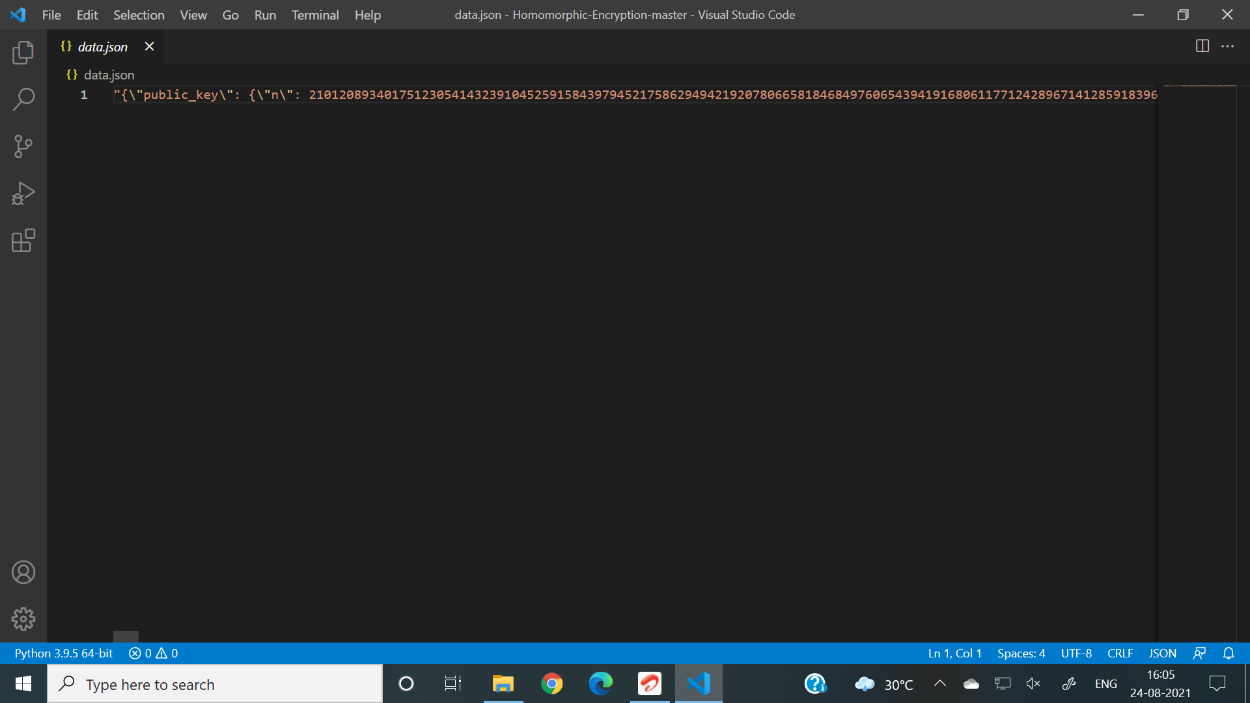
Employee\_data.csv

Then we create a cust.py file

In this file we create an instance of public and private keys, store them in custkeys.json file, then send the encrypted data along with the public key, stored in a file called data.json to the company.



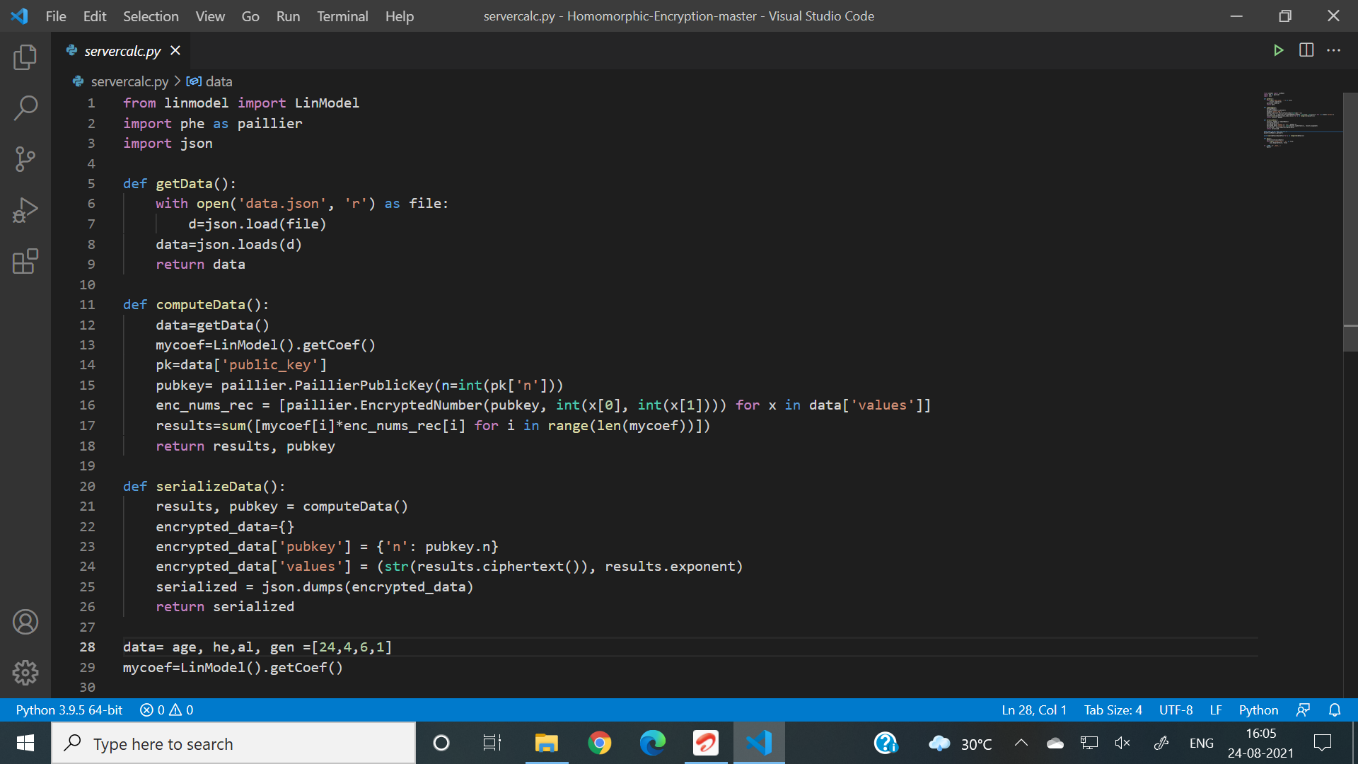
Cust.py



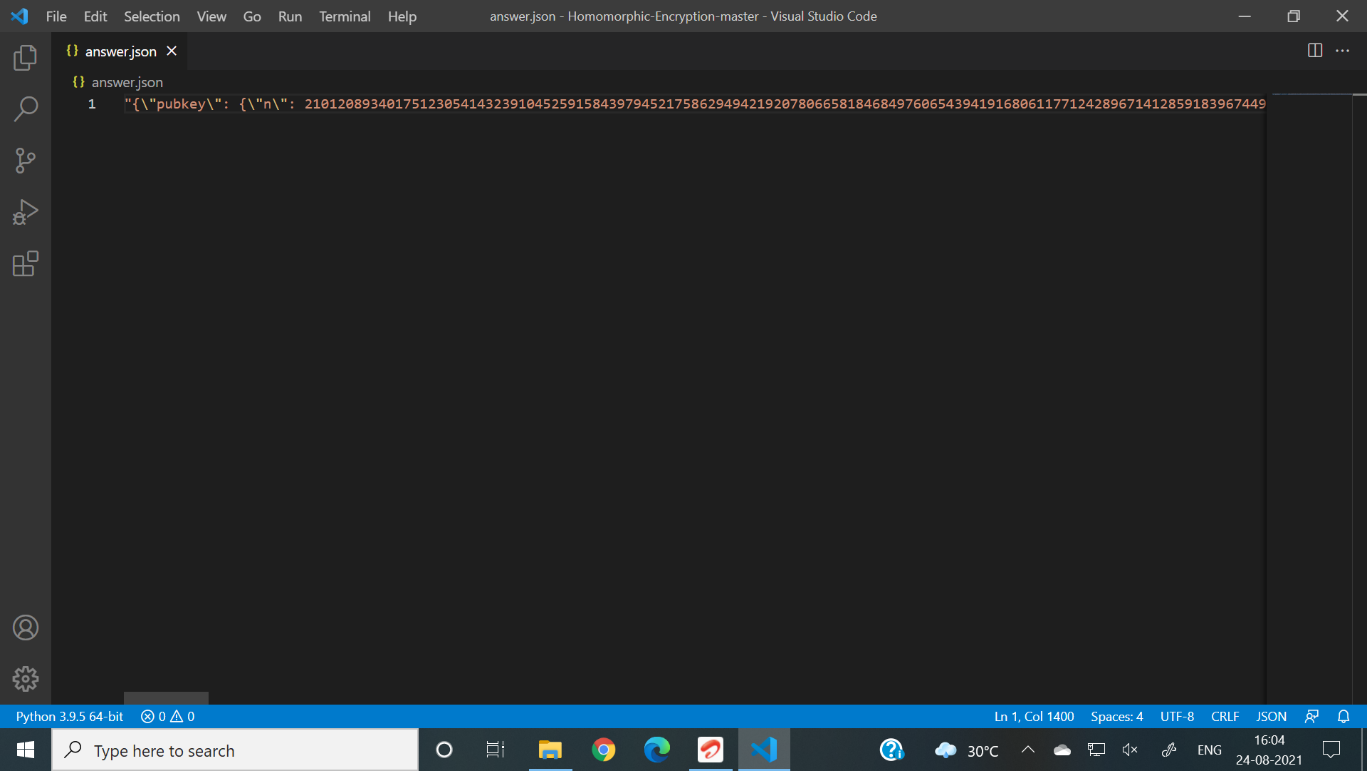
Data.json

Now we create a file called servercalc.py.

This file gets data from data.json file send by the customer, performs calculations using the data from customer and the companies ML model, generates a encrypted result, stores it in a answer.json file, and sends it to the customer along with the previously send public key.

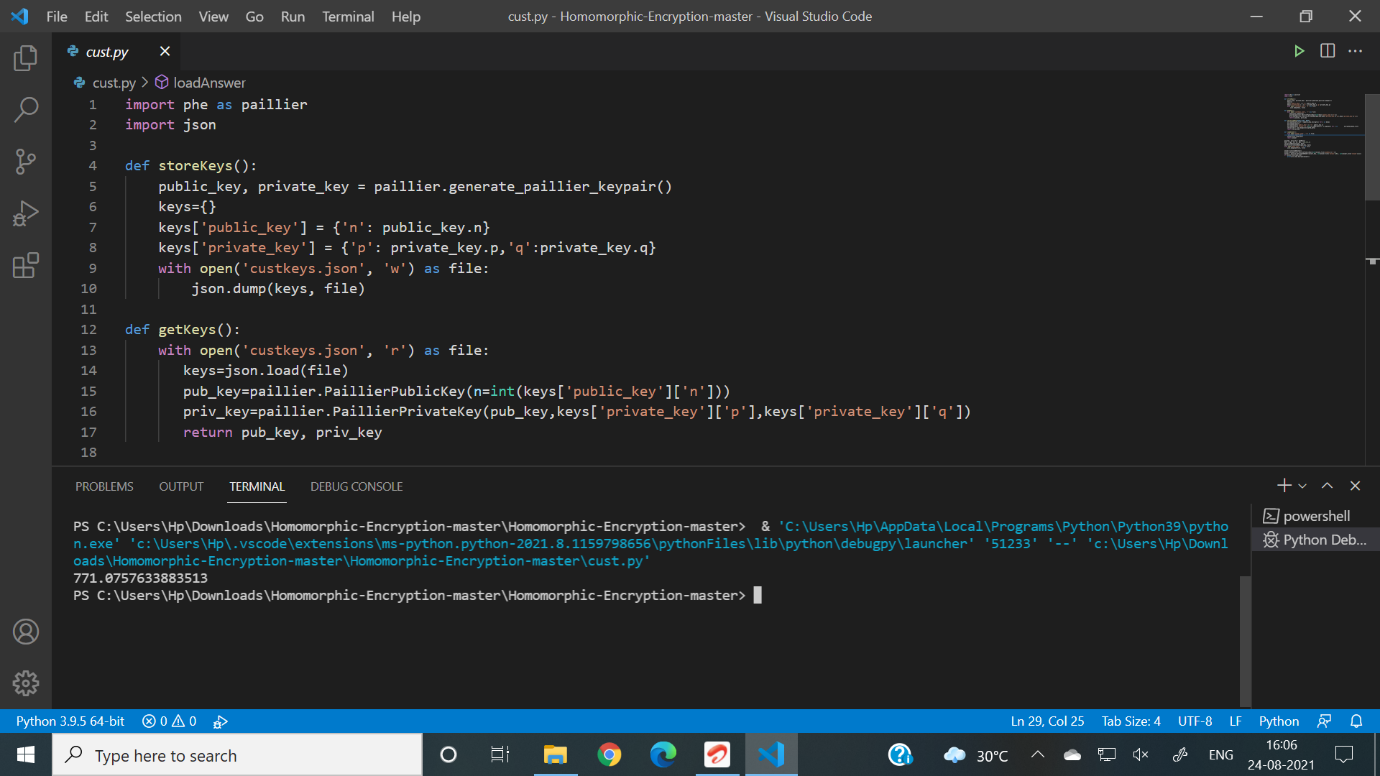


Severcalc.py



Answer.json

Now the customer opens the cust.py file, loads the answer.json file send by the company, matches the public key send by the company with its own public key, if matched decrypts the data using their private key and gets the result.



Result is 771.0757633883513

Conclusion:

The customer was successfully able to send data to company in encrypted form. The company was able to apply their machine learning model against the encrypted data send by the customer and produce the encrypted result. Since both the public keys were identical therefore the customer could successfully get the produced result. Therefore, in this project, we were able to change an existing algorithm to secure data send by the user and also verify the identity of the other user. We were able to implement python and ML functions effectively.