



Homomorphic Encryption

Secure Information Aggregation



October 14, 2017

NIT Raipur

Submitted by:

Shubham Gaur

Under the guidance of:

Dr. Govind P. Gupta

**Homomorphic Encryption**

Secure Information Aggregation

For Smart Grids Using H.E.

**Shubham Gaur**

Master of Technology (Information technology)

Table of Contents

[Abstract 3](#_Toc495830595)

[Introduction 4](#_Toc495830596)

[Prerequisites 5](#_Toc495830597)

[Main.py 9](#_Toc495830598)

[Paillier H.E. 5](#_Toc495830599)

[Body 12](#_Toc495830600)

[Conclusion 13](#_Toc495830601)

[References 14](#_Toc495830602)

# Abstract

In the digital world we are living in; the conventional power system has to meet the demands of a digital society. Thus, upgradation was required. This included smart meters. In this project, we emphasized on the security of data forwarded by smart meters. Data over a network has a high risk of being hacked and modified by attackers. Pure Paillier Homomorphic Encryption is used to encrypt data at each node. In this way, our approach supports efficient data aggregation while fully protecting user privacy. It is helpful for smart grids with repetitive routine data aggregation tasks.

# Introduction

Encryption is a system of mathematical algorithms that encodes user data so that only the intended recipient can read it. As simple as it sounds, the math and extra steps can become onerous for beginners. Smart Grids are envisioned by numerous and diverse stakeholders as the next-generation approach of delivering electricity to military of household worldwide. Security and privacy are major concern of smart meters. Hackers could compromise smart meters to manipulate power usage and energy costs. Thus, homomorphic encryption is applied to enhance the security of the system as it provide us the accessibility to perform operations like addition and multiplication on encrypted data.

**What is encryption and how does it work?**

In terms of security, there’s always a trade-off as it won't be difficult for an intruder to intercept any connection, which could result in stolen user credentials and other sensitive data. This is why data has been communicated in encrypted form over a channel. While this doesn’t necessarily guarantee absolute security, the risks are reduced as information being transmitted can only be decrypted by the receiver it was sent to. When it comes to Homomorphic Encryption it uses extremely large prime numbers as secret keys to perform encryption. This is also known as endpoint encryption, which basically adds an extra layer of protection for the confidential information residing on.

**Why do we need it?**

While encryption doesn’t magically convey security, it can still be used to protect a user's identity and privacy. If we are ever being watched, inadvertently or not, we can hide our data by using properly implemented crypto systems. According to cryptographer and security and privacy specialist Bruce Schneier :

“*Encryption works best if it is ubiquitous and automatic. It should be enabled for everything by default, not a feature you only turn on when you’re doing something you consider worth protecting.*”

# Prerequisites

Python is the most versatile language to use while implementing Pure Homomorphic Encryption where it is highly readable and writable language to deal with it provide flexibility to inherit various modules and perform operations simultaneously.

# Paillier H.E.

This is a very basic pure Python implementation of the Paillier Homomorphic Cryptosystem.

**Homomorphic Cryptosystems:**

The idea of homomorphic computation is to encrypt some numbers, perform algebraic operations like "add" and "multiply" on *cyphertexts*, then decrypt the result and find it to be exactly the same as if corresponding "+" and "\*" operations were applied to the plaintexts.

In other words, a homomorphic cryptosystem enables cryptographically secure computations in an untrusted environment.

**Paillier cryptosystem:**

Paillier cryptosystem is a probabilistic asymmetric algorithm for public key cryptography. Paillier cryptosystem is partially homomorphic as it can only add encrypted numbers or multiply an encrypted number by an unencrypted multiplier.

## Encryption

1. Let m be a message to be encrypted where
2. Select random r where
3. Compute cipher text as:

## Decryption

1. Let c be the cipher text, where
2. Compute the plaintext message as

## Homomorphic Properties

A notable feature of the Paillier cryptosystem is its homomorphic properties along with its non-deterministic encryption (see Electronic voting in Applications for usage). As the encryption function is additively homomorphic, the following identities can be described:

Homomorphic Addition of plain text:

The product of two cipher texts will decrypt to the sum of their corresponding plaintexts,

The product of a cipher texts with a plaintext raising {\displaystyle g} will decrypt to the sum of the corresponding plaintexts,

Homomorphic Multiplication of plain text:

An encrypted plaintext raised to the power of another plaintext will decrypt to the product of the two plaintexts,

More generally, an encrypted plaintext raised to a constant *k* will decrypt to the product of the plaintext and the constant,

However, given the Paillier encryptions of two messages there is no known way to compute an encryption of the product of these messages without knowing the private key.

## Implementation

This pure Python implementation exploits Python's long type with its arbitrary precision arithmetics. Public key is serializable, thus it can be pickled along with the encrypted numbers and sent to a remote server for computation.

The code is loosely based on [Thep](http://code.google.com/p/thep/) and a few ActiveState recipes.

Please note that this implementation's primary purpose is education; it is **not suitable for production use** as it is.

## Installation and Tests

The paillier.py module has no external dependencies besides included primes.py. Simply run demo.py to see it in action.

Python 2.7.11 (v2.7.11:6d1b6a68f775, Dec 5 2015, 20:40:30) [MSC v.1500 64 bit (AMD64)] on win32

Type "help", "copyright", "credits" or "license" for more information..

In [1]: from paillier.paillier import \*

In [2]: priv, pub = generate\_keypair(128)

In [3]: x = encrypt(pub, 2)

In [4]: y = encrypt(pub, 3)

In [5]: x,y

Out[5]:

(72109737005643982735171545918..., 9615446835366886883470187...)

In [6]: z = e\_add(pub, x, y)

In [7]: z

Out[7]: 71624230283745591274688669...

In [8]: decrypt(priv, pub, z)

Out[8]: 5L

# Main.py

#!/usr/bin/env python2.7.11

“

A spanning tree rooting at the collector device. /

Each node collects data from its children, aggregates from /

its own data and sends the intermediate result to the parent /

node. /

”

Main.py implements following predefined modules such like “sys”, “csv” and user defined modules such as “spin”, “feed”, “encrypt”. Further modules like “mygraph” and “paillier” are used within defined function to create a minimum spanning tree and provide encryption at node data.

import sys

import spin

import feed

import encrypt

import csv

Creates a new *instance* of the class and assigns this object to the local variable “gi” to mark input to the “GraphInput()“.

gi = feed.GraphInput()

Function Calling

gi.tree(): It provides visual tree representation of input data in the form of minimum rooted spanning tree where root is the collector node and all the other nodes are either substations or smart meters.

gi.destinations(): It provides list of all the nodes set as destination in the GraphInput().

gi.implement(): It provides the input to MyGraph and create a minimum spanning tree on the basis of input provided in csv file.

gi.path(): Dijkstra’s path is calculated providing the shortest path between the source and the destination.

gi.encrypt(): Encrypt function is called to perform encryption on received data by each node and run simultaneously.

encrypt.getList(): Encrypted data at each node is stored in a list

encrypt.encRes(): Cummulutative result is encrypted under this function using Paillier Encryption.

gi.srcData(): It provides fetching of data at collector node.

encrypt.dec(): It decrypt the encrypted data using public and private key.

Initializing a program uses a class module “Spinner” to create an illusion of data processing.

**Note**: *No actual data is processed. Can be made useful by implementing threading in “Spinner”.*

sys.stdout.write('\nInitializing ')

spin.Spinner(' : Completed!')

Demo code gives the user interactive commands to perform operations as defined.

#<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<" Demo ">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

condition = True

while(condition):

print('\n\n==============Commands=================')

print('To see tree “ t ”)

print('To start “ s ”)

print('To print destinations “ des ”)

print('To encrypt “ e ”)

print('To see encrypted list enter “ el ”)

print('To see encrypted result enter “ er ”)

print('To see collector data enter “ cd ”)

print('To decrypt result enter “ d ”)

print('To quit press “ q ”)

print('==================Commands===================\n\n')

x = raw\_input("\nInsert value and press Enter : ")

if(x == 't'):

gi.tree()

elif(x == 'des'):

gi.destinations()

elif(x == 's'):

gi.implement()

gi.path()

elif(x == 'e'):

gi.encrypt()

elif(x == 'el'):

encrypt.getList()

elif(x == 'er'):

encrypt.encRes()

elif(x == 'cd'):

gi.srcData()

elif(x == 'd'):

encrypt.dec()

elif(x == 'q'):

condition = False

else:

print 'No valid input'

#<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<" Demo ">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

Options includes various cases under which tree formed can be seen and modified respectively by modifying csv file which includes source-destination and its position co-ordinates. CSV file can be used to take input to Feed.py to form minimum spanning tree and then Dijkstra’s algorithm is applied to find the shortest path between two nodes.

# MyGraph

MyGraph.py implements the minimum spanning graph structure in smart grid network. It uses shortest path algorithm like dijkstra. Dijkstra uses distance vector routing to find the shortest route between the source node and the destination node. Here the source node is smart meters and their data is cummtatively sent to base station.

import heapq

def dijkstra(aGraph, start):

print '''Dijkstra's shortest path'''

# Set the distance for the start node to zero

start.set\_distance(0)

# Put tuple pair into the priority queue

unvisited\_queue = [(v.get\_distance(),v) for v in aGraph]

heapq.heapify(unvisited\_queue)

while len(unvisited\_queue):

# Pops a vertex with the smallest distance

uv = heapq.heappop(unvisited\_queue)

current = uv[1]

current.set\_visited()

#for next in v.adjacent:

for next in current.adjacent:

# if visited, skip

if next.visited:

continue

new\_dist = current.get\_distance() +

current.get\_weight(next)

if new\_dist < next.get\_distance():

next.set\_distance(new\_dist)

next.set\_previous(current)

print '|| updated : current = %s | next = %s |

new\_dist = %s' \

%(current.get\_id(), next.get\_id(),

next.get\_distance())

else:

print '|| not updated : current = %s | next = %s |

new\_dist = %s' \

%(current.get\_id(), next.get\_id(),

next.get\_distance())

# Rebuild heap

# 1. Pop every item

while len(unvisited\_queue):

heapq.heappop(unvisited\_queue)

# 2. Put all vertices not visited into the queue

unvisited\_queue = [(v.get\_distance(),v) for v in aGraph if

not v.visited]

heapq.heapify(unvisited\_queue)

# Conclusion

Under this project, Aggregation is performed in a distributed manner accordance to the aggregation tree – each node collects data from its children, aggregates them with its own data, and sends the intermediate result to the parent node. Homomorphic Encryption is applied to protect the privacy of data. Aggregated data is made secure on the aggregation path and ensured that results are not revealed to smart meters while being traversed on a network. This module does not cover user maliciously forging their own data to manipulate aggregation result. Thus ensuring privacy control from foreign attacks only. Surely better techniques are coming to check for forged data and recovering the loses.

# References

* <https://github.com/mikeivanov/paillier>
* <http://www.trendmicro.co.in/vinfo/in/security/news/online-privacy/encryption-101-what-it-is-how-it-works>
* <https://docs.python.org>