IMPLEMENTATION OF MULTI AGENT SYSTEM FOR COMPUTATION OVER CIPHER TEXT FOR INTERNET OF THINGS

Scipy 2018

Gajendra Deshpande, KLS GIT, Belagavi

Contents

Homomorphic Encryption

- Several billion devices are currently connected to the Internet, and this number will continue to grow.
- This is a consequence of not only more people becoming interested in consumer electronics but also more sensors and actuators being incorporated into everyday electronics, household appliances, and the general infrastructure.
- Since most of these devices are not able to process data locally, they will often upload it to a third party for processing.
- However, this data may be private, the third party may not be trustworthy, or both. Therefore, the data should be encrypted before it is transferred

Homomorphic Encryption

- Imagine taking all of your credit card statements and locking them into a safe, to which you have the only key. Your statements are now protected from prying eyes. This is what encryption does.
- But what if you wanted to analyse your expenditure on groceries in the last 12 months? First you would have to unlock the safe and retrieve the statements. So now the documents are out in the open and they can be read by anyone. This is what decryption does.
- The difference with Homomorphic Encryption is that you can create your report without taking the documents out of the safe.

Properties of Homomorphic Encryption

- **Additive Homomorphic Encryption:**

A Homomorphic encryption is additive, if

Ek (PT1 $\oplus$ PT2) = Ek (PT1) $\oplus$ Ek (PT2)

As the encryption function is additively homomorphic, the following identities can be described:

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

D (E (m1) · E (m2) mod n) = m1 + m2 mod n.

The product of a cipher text with a plaintext raising g will decrypt to the sum of the corresponding plaintexts,

D (E (m1) · g^{m2} mod n2) = m1 + m2 mod n.

Properties of Homomorphic Encryption

- **Multiplicative Homomorphic Encryption:** Homomorphic encryption is multiplicative, if

$$Ek\ (PT1 \otimes PT2) = Ek\ (PT1) \otimes Ek\ (PT2)$$

- The homomorphic property of the RSA.

Suppose there are two cipher texts, CT1 and CT2.

$CT1 = m1^e \bmod n$

$CT2 = m2^e \bmod n$

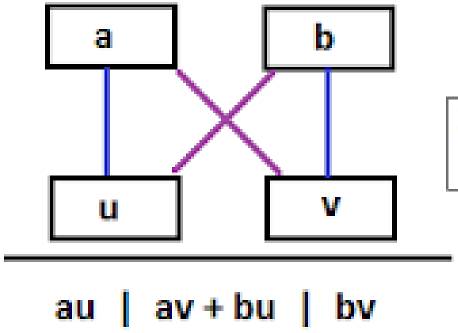
$CT1 \cdot CT2 = m1^e \cdot m2^e \bmod n$

So, multiplicative property: $(m1 \cdot m2)^e \bmod n$

Summary of Homomorphic Properties

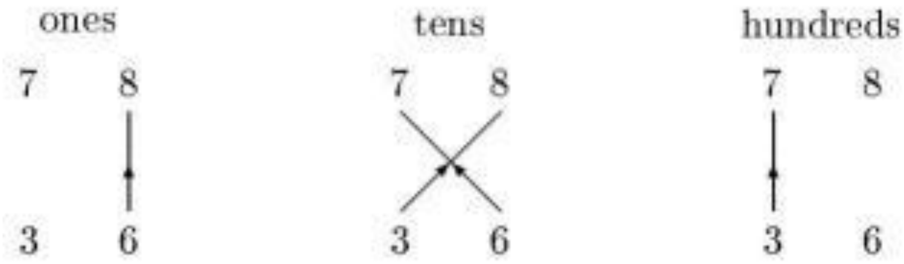
Algorithm	Additive	Multiplicative	Applications
RSA	No.	Yes	To secure Internet Banking and credit card transactions
Paillier	Yes	No	E-voting system
ElGamal	No.	Yes	In Hybrid Systems

Vertically-Crosswise Multiplication

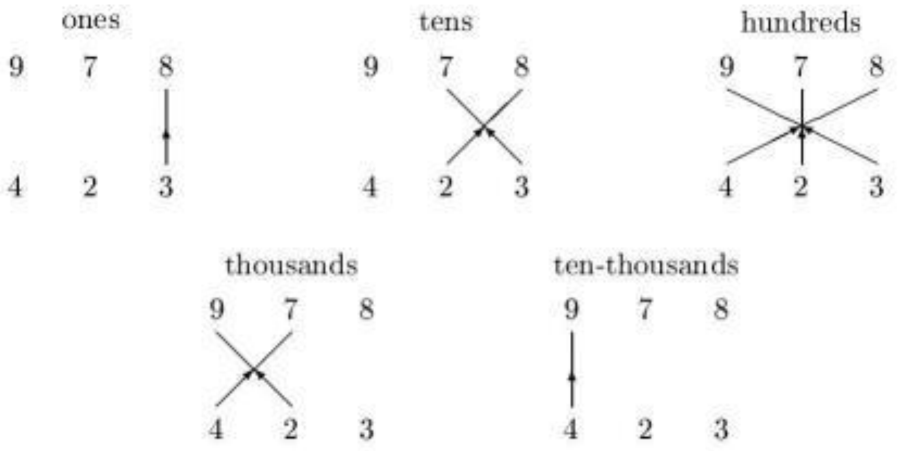


2 by 2 digits, verticals in blue and cross in purple

VERTICAL | CROSS | VERTICAL



Vertically-Crosswise Multiplication



Ones place:
3×8=24; write down 4, remember 2. [4]

Tens place:
2 + 3×7 + 2×8 = 39; write down 9, remember 3. [94]

Hundreds place:
3+3×9+2×7+4×8=76; write down, 6, remember 7. [694]

Thousands:
7+2×9+4×7=53; write down 3, remember 5. [3694]

Ten-thousands: 5+4×9=41; write down 41. [413694]

vedic-py package

- Python library for Vedic Maths sutras. This library implements the vedic maths sutras for performing basic mathematical operations like addition, subtraction, multiplication, square roots, cube roots etc.
- Since vedic maths sutras work on individual digits in a number as opposed to the whole number, they can treat numbers as strings and hence not run into issues with storage and computations on very large numbers.
- URL:https://github.com/techmoksha/vedic-py

vedic-py package

- Two vedic numbers can be multiplied using the * operator of python. The multiplication is performed using Vertical-Crosswise sutra of vedic maths

```
from vedic import VedicNumber

print(VedicNumber(45) * VedicNumber(57))
```

osBrain package

- osBrain is a general-purpose multi-agent system module written in Python and developed by OpenSistemas. Agents run independently as system processes and communicate with each other using message passing.
- In general, osBrain can be used whenever a multi-agent system architecture fits the application well:
  - Autonomy of the agents.
  - Local views.
  - Decentralization.
- URL: https://github.com/opensistemas-hub/osbrain
- Installation: pip install osbrain

osBrain package

```
from osbrain import run_nameserver
from osbrain import run_agent

if __name__ == '__main__':

    # System deployment
    ns = run_nameserver()
    agent = run_agent('Example')

    # Log a message
    agent.log_info('Hello world!')

    ns.shutdown()
```

```
gajendra@gajendra-virtual-machine:~/Desktop/osbrain-master/examples$ python3 hello_world.py
Broadcast server running on 0.0.0.0:9091
NS running on 127.0.0.1:12620 (127.0.0.1)
URI = PYRO:Pyro.NameServer@127.0.0.1:12620
INFO [2018-12-22 02:03:48.383790] (Example): Hello world!
INFO [2018-12-22 02:03:48.427736] (Example): Stopping...
NS shut down.
```

SPADE package

- Smart Python Agent Development Environment
- A multi-agent systems platform written in Python and based on instant messaging (XMPP).
- Develop agents that can chat both with other agents and humans.
- Agent model based on behaviours
- Tutorial: https://spade-mas.readthedocs.io/en/latest/readme.html
- Source: https://github.com/javipalanca/spade/
- Installation: pip install spade

SPADE package

```
import spade

class DummyAgent(spade.agent.Agent):
    def setup(self):
        print("Hello World! I'm agent {}".format(str(self.jid)))

dummy = DummyAgent("agent11@localhost", "gcd1234")
dummy.start()
dummy.stop()
~
~
~
```

Implementation

- Choose two plain texts P1 and P2
- Implement RSA Algorithm to convert P1 and P2 to cipher texts C1 and C2 respectively.
- Multiply plain texts i.e., Product=P1*P2
- Multiply both Cipher texts using * operator i.e., CProduct=C1*C2. Note Computation Time: 16.4404850006 Seconds
- Multiple both Cipher texts using vertically-crosswise technique i.e., Vproduct=C1*C2. Note Computation Time 15.4647901058 Seconds
- Verify that Product=Dec(CProduct) and Product=Dec(VProduct)

Output

```
gajendra@gajendra-virtual-machine:~/Desktop/Scipy$ python3 rsafinal.py
First list of random numbers
[81106012127733]


Second list of random numbers
[98809777787844]


N =  1343699989903543000899529375595875357769694485422750362743548771395805652947095717349849873446599810498238682568701316541791721316613044593620602696208407573660126974303261538045053780030070198091288257047907206670043320365742140237545036958711651645282254471671400004907662779146417277543
21849750023
e =  3
Encryption of First list of Random numbers
[53353036887440152152937021206685217594183 7]


Encryption of Second list of Random Numbers
[964716638859980065302941251760550507584]


Decryption of First list of Random Numbers
[81106012127733]



Decryption of Second list of Random Numbers
[98809777787844]
```

Output

```
Product of Two Encrypted Lists
[[51470562419023796157760880436248680846722447002491033204259134817174600361139180 8]]

Product of Two Decrypted Lists
[[80140670429473585390267765 2]]

Decryption of Product of Encrypted Lists
[801406704294735853902677652]


Vedic Product of Two Encrypted Lists
514705624190237961577608804362486808467224470024910332042591348171746003611391808


Vedic Product of Two Decrypted Lists
801406704294735853902677652
```

Conclusion

- Homomorphic Encryption enables computation on untrusted resource. The Computation time over cipher text can be reduced by using vertically-crosswise technique.
- Need to test the computation time with respect to homomorphic properties of Elgamal, Paillier and ECC Systems.
- Multi agent system can be used for load balancing

Widescreen Test Pattern (16:9)

Aspect Ratio Test

(Should appear circular)

4x3

16x9