

Danger of using fully homomorphic encryption, a look at Microsoft SEAL

**Zhiniang Peng of 360 Core Security
@Opcde 2019**



360 INTERNET SECURITY CENTER

Who we are

Zhiniang Peng

Ph.D. in cryptography

Security researcher @Qihoo 360

Twitter: @edwardzpeng

Research areas:

Software security

Applied cryptography

Threat hunting

About the topic

Introduction to homomorphic encryption

Introduction to SEAL

Security pitfalls of SEAL

- CCA attack on BFV

- Data recovery against FPSI

- Circuit privacy of SEAL

- Information leakage

- Countermeasures

Other issues

Conclusion

Computing on encrypted data

Data leakage become more serious nowadays

Data security, privacy become a public concern

It will be nice to be able to....

Encrypt my data before send to the cloud

While still allowing the cloud to search/sort/edit the data on my behalf

Keep the data in cloud in encrypted form

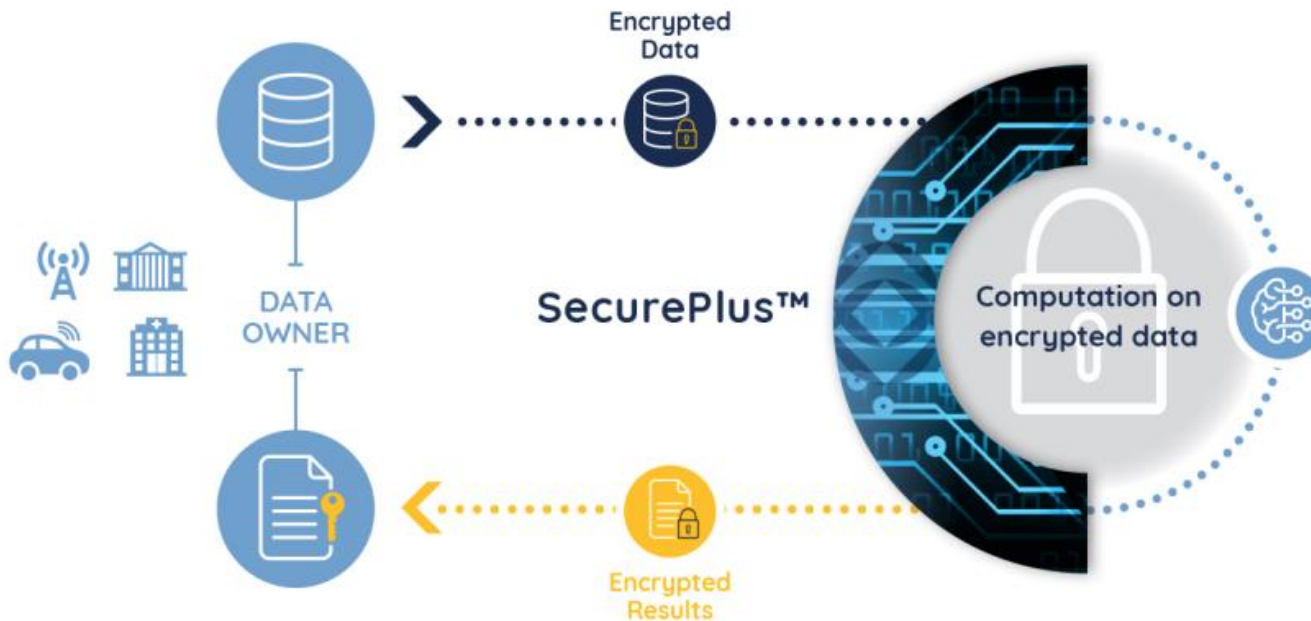
Computing on encrypted data

It will be nice to be able to....

Encrypt my query to the cloud

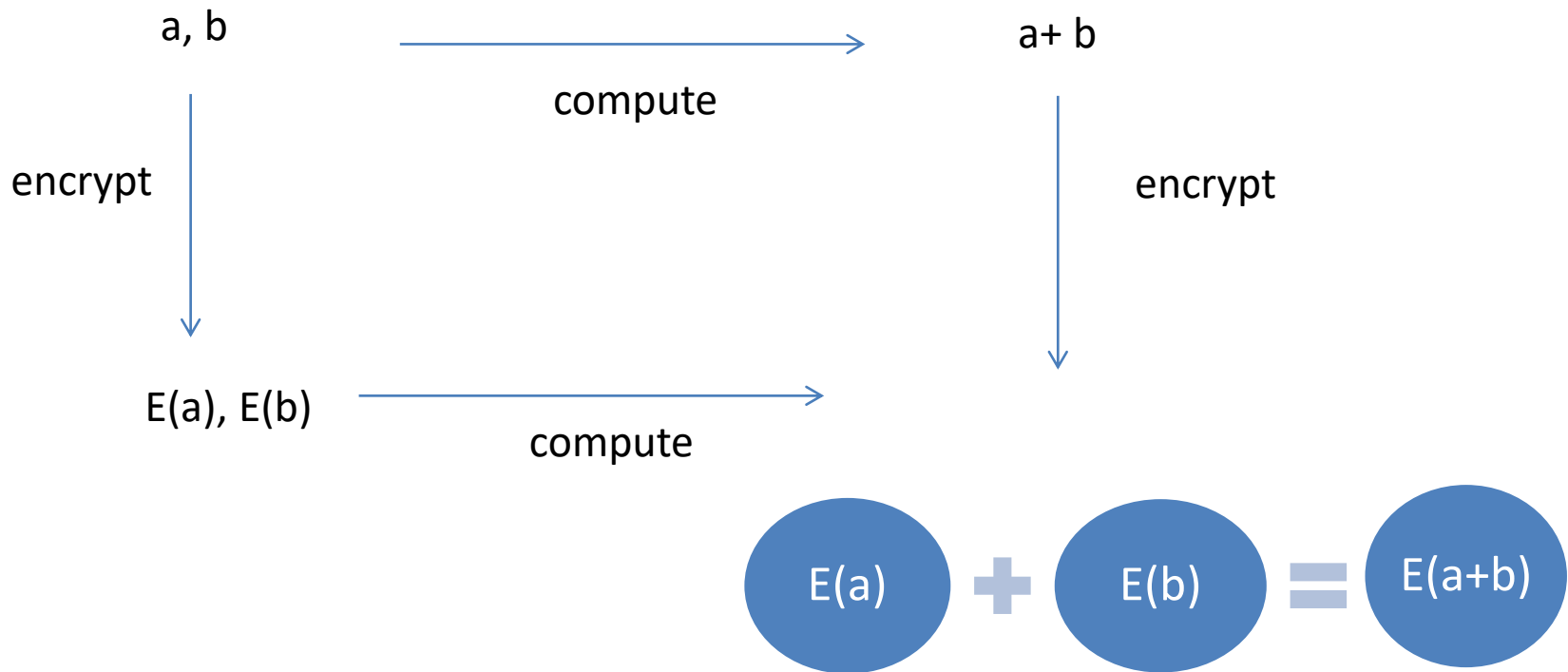
While still allowing the cloud the process them

Cloud returns encrypted answers
that I can decrypt



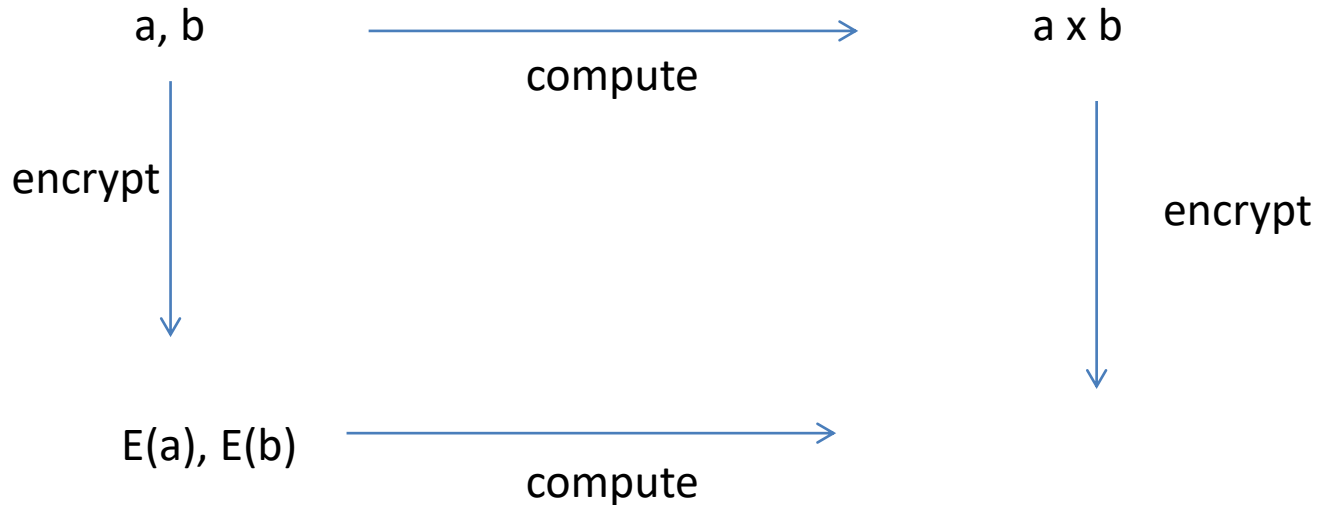
This picture is from duality.cloud

Homomorphic addition



Pure RSA support homomorphic addition!

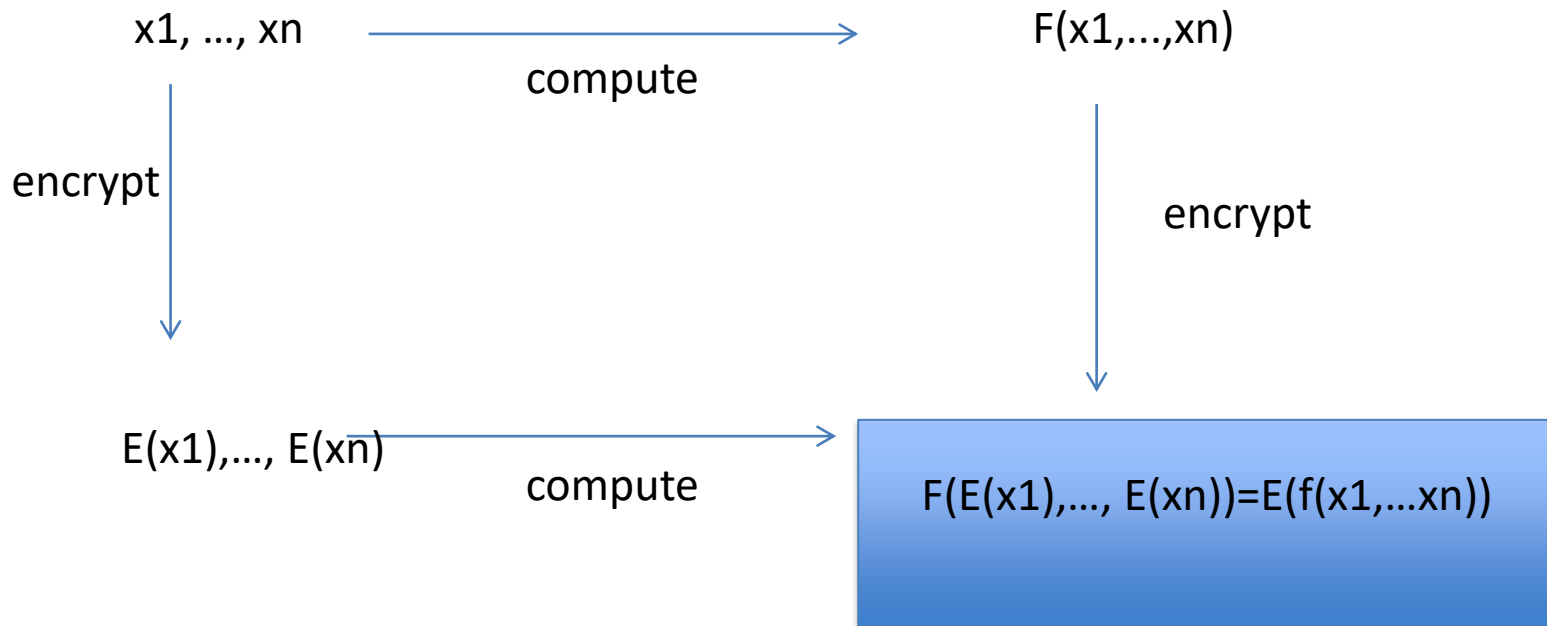
Homomorphic multiplication



$$E(a) \times E(b) = E(ab)$$

Pure Elgamal support homomorphic multiplication!

Fully homomorphic encryption



Craig Gentry proposed the first fully homomorphic encryption in 2009

Protecting Data via Encryption

A famous metaphor



1. Put your gold in the locked box
2. Keep your key
3. Let the worker work on it through a glove box
4. Unlock the box and get the jeweler

A p p l i c a t i o n s o f H E

Outsourcing computation

Machine learning on encrypted data

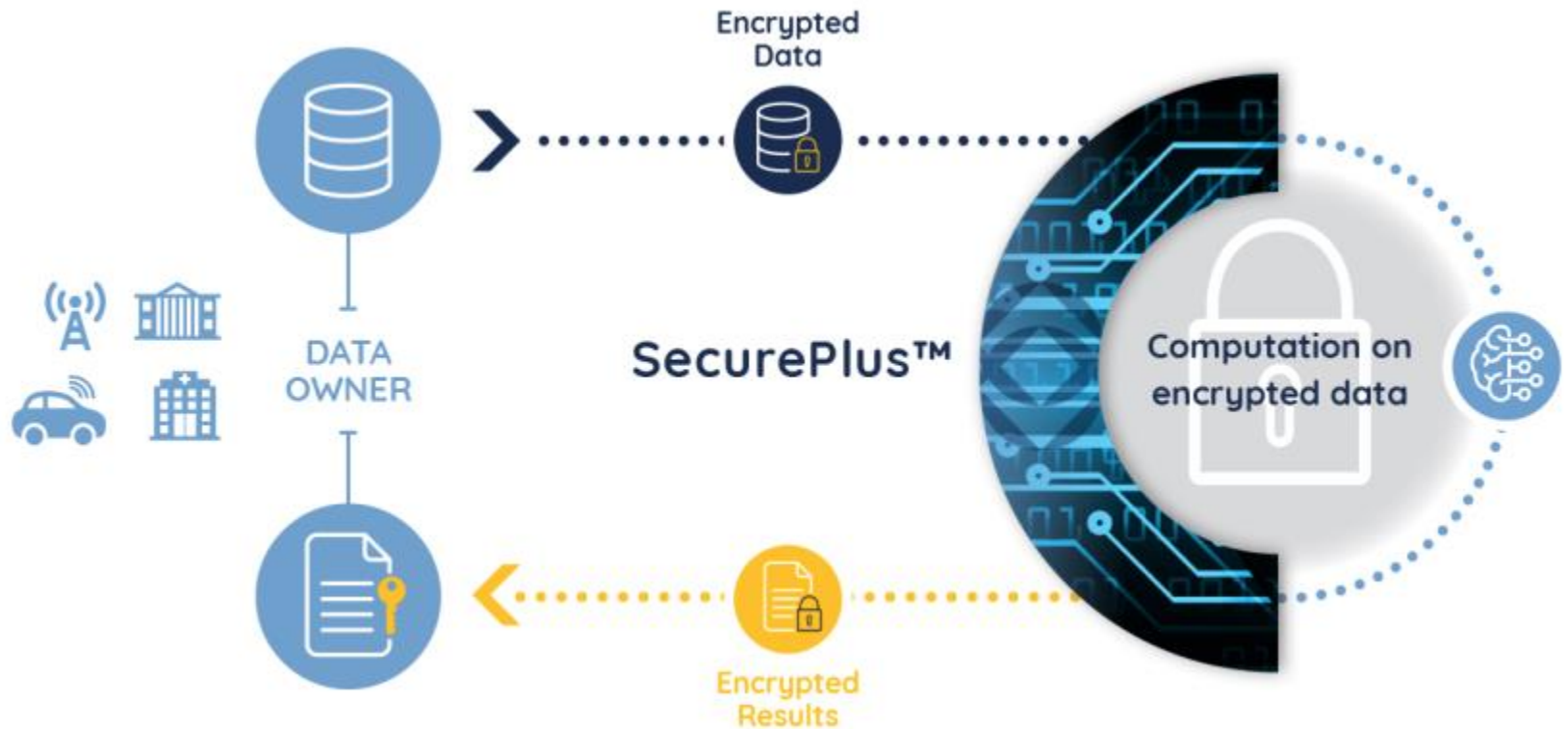
Cloud service without knowing clients' privacy

There are two kinds of applications:

private data , public function

private data , private function

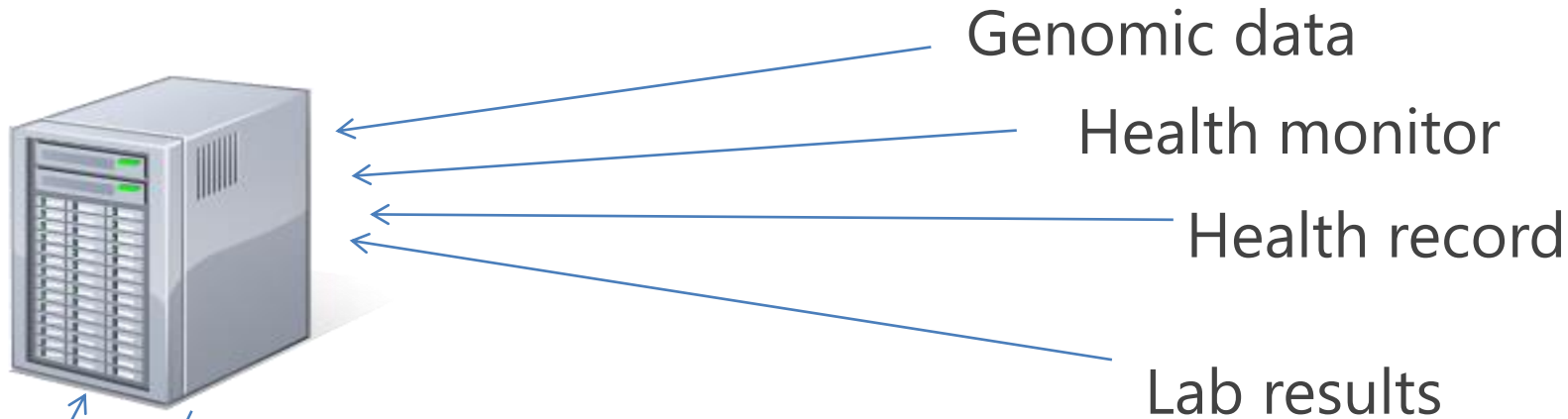
Private data , Public function



This picture is from duality.cloud

Data should be keep secret.
The function f can be public.

Disease prediction



private data , public function

- All data uploaded to the server encrypted under patient's public or private key
- Cloud operates on encrypted data and returns encrypted predictive results



Private data , Private function



This picture is from duality.cloud

Both data and the model should be keep secret.

Circuit privacy:

An additional requirement in many FHE applications is that the evaluated ciphertext should also hide the function f .



SEAL

Simple Encrypted Arithmetic Library

Quick Background

Homomorphic Encryption library from Microsoft Research

First version released in 2015; SEAL 3.1 just released

Developed in standard C++

Implements BFV and CKKS schemes

Simple and easy to use

Comes with detailed examples

Performance of SEAL

CryptoNets:

MIST handwritten digital picture recognition

60,000 predictions per hour

99% correct rate

16 encrypted pictures per second

Our experiment:

logistic regression prediction

10,000 pieces of data in 5 minutes

300 times slower than using sklearn directly on plaintext

Seems reasonable

HOW
DOES IT
WORK



Ring-LWE problem

Ring $R = \mathbb{Z}_q[x]/(x^{n+1})$

Given:

To make it simply to understand

You can think all elements here are integer

$$a_1, b_1 = a_1 \cdot s + e_1$$

$$a_2, b_2 = a_2 \cdot s + e_2$$

...

$$a_k, b_k = a_k \cdot s + e_k$$

Find: s

s is random in R

e_i are “small” (distribution symmetric around 0)

Decision Ring-LWE problem

Ring $R = \mathbb{Z}_q[x]/(x^{n+1})$

Given:

a_1, b_1

a_2, b_2

...

a_k, b_k

Question: Does there exist an s and “small”

e_1, \dots, e_k such that $b_i = a_i \cdot s + e_i$

or are all b_i uniformly random in R ?

BFV key pair

SecretKeyGen():

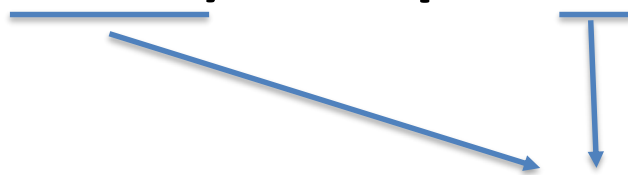
sample secret key $s \in \chi$

PublicKeyGen(s):

sample $a \in R_q, e \in \chi$

$pk_0 = -(a \cdot s + e)$

$pk_1 = a$



Ring-LWE pair

s cannot be recovered

Over simplified!

You can think all these are integer

BFV encryption

Encrypt(m): sample $u \in R_q$, $e_1, e_2 \in \chi$

$$c_0 = pk_0 \cdot u + e_1 + \Delta \cdot m, \quad c_1 = pk_1 \cdot u + e_2$$

Replace pk with $(-(a \cdot s + e), a)$

$$c_0 = -(a \cdot s + e) \cdot u + e_1 + \Delta \cdot m, \quad c_1 = a \cdot u + e_2$$

$$c_0 = -w \cdot s + e_1 + e \cdot u + \Delta \cdot m, \quad c_1 = w + e_2$$

Decision Ring-LWE pair (cannot be distinguished with random value)
Message m is encrypted with a random pad

Ciphertext can be considered as a polynomial:

$$f(x) = c_0 + c_1 \cdot x$$

BFV decryption

Decrypt(c):

$$f(x) = c_0 + c_1 \cdot x$$

substitute x with s

$$f(s) = c_0 + c_1 \cdot s$$

Replace c with $([-w \cdot s + e_1 + e \cdot u + \Delta \cdot m]_q, [w + e_2]_q)$

$$f(s) = -w \cdot s + e_1 + e \cdot u + \Delta \cdot m + (w + e_2) \cdot s$$

$$= \underline{e_1 + e \cdot u + e_2 \cdot s} + \Delta \cdot m$$

Much smaller than Δ
Then we can recover m

We can think that:

$$f(s) = v + \Delta \cdot m$$

where v is much smaller than Δ

Homomorphic addition

Homomorphic addition:

Ciphertext1: $f_1(x)$

Ciphertext2: $f_2(x)$

Compute: $f_3(x) = f_1(x) + f_2(x)$

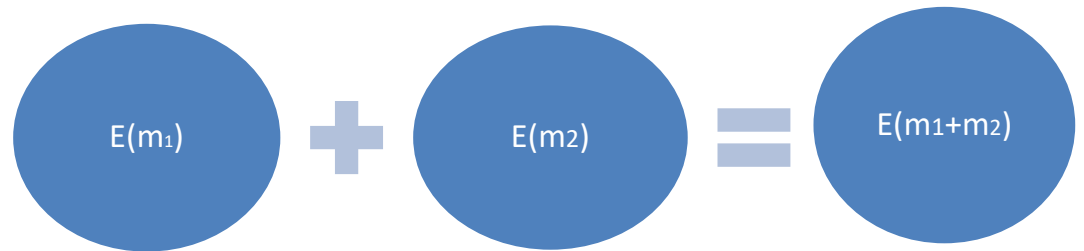
We have:

$$f_1(s) = v_1 + \Delta \cdot m_1$$

$$f_2(s) = v_2 + \Delta \cdot m_2$$

Then decrypt $f_3(x)$:

$$\begin{aligned} f_3(s) &= f_1(s) + f_2(s) = v_1 + v_2 + \Delta \cdot (m_1 + m_2) \\ &= v_3 + \Delta \cdot (m_1 + m_2) \end{aligned}$$



Homomorphic multiplication

Homomorphic multiplication:

Ciphertext1: $f_1(x)$

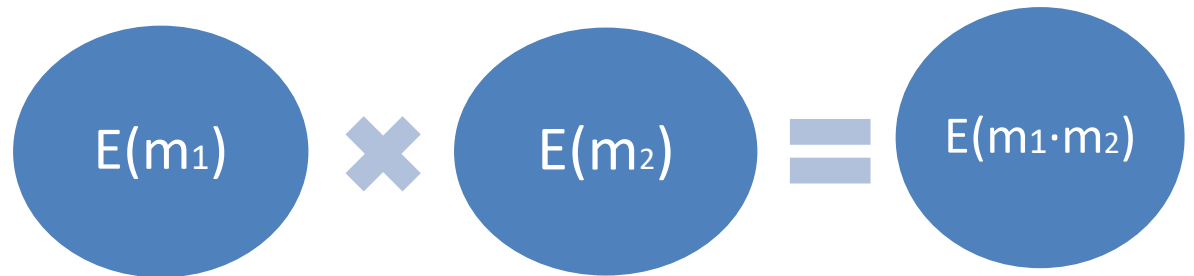
Ciphertext2: $f_2(x)$

Compute: $f_3(x) = f_1(x) * f_2(x)$

We have:

$$f_1(s) = v_1 + \Delta \cdot m_1$$

$$f_2(s) = v_2 + \Delta \cdot m_2$$



Then decrypt $f_3(x)$:

$$f_3(s) = f_1(s) * f_2(s) = v_1 \cdot v_2 + \Delta \cdot (v_1 \cdot m_2 + v_2 \cdot m_1) + \Delta \cdot m_1 \cdot m_2$$

Divide by Δ , we can get:

$$f_3(s) / \Delta = v_3 + \Delta \cdot (m_1 \cdot m_2)$$

More about BFV scheme

This is just a over simplified version of BFV

But enough to make you us understand the problems

For more details about BFV

Read the paper:

Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical gapsvp. In: CRYPTO 2012 - Volume 7417. pp. 868–886 (2012)

Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144 (2012)

You can play with BFV

I write a sage version of simplified BFV

<https://github.com/edwardz246003/danger-of-using-homomorphic-encryption/blob/master/BFV.py>

Security of BFV scheme

Encrypt message m to an polynomial $f(x)$

Decrypt by substitute x with s

get $f(s)=v+\Delta \cdot m$, can recover m easily

Message is “blind” by Ring-LWE pair + noise

Distinguish ciphertext \rightarrow distinguish Ring-LWE pair

Provable security: IND-CPA \rightarrow Ring-LWE

Chosen plaintext attack

If someone break IND-CPA, he can break Ring-LWE

Ring-LWE is suppose to be a hard math problem

IND-CCA?

chosen ciphertext attack

attacker is given access to a *decryption oracle*

BFV doesn't have IND-CCA security

All practical FHE cannot guarantee IND-CCA

homomorphic property seem conflict with CCA

Theoretical research on CCA FHE

Chosen-Ciphertext Secure Fully Homomorphic Encryption

All FHE implementation cannot guarantee security in the IND-CCA scenario

IND-CCA Scenario

Why need IND-CCA

attacker can always ask for a decryption in real scenario

IND-CCA is a standard requirement for encryption scheme

Scenarios that require HE often require IND-CCA

rich data flows between data-owner and cloud

multi-party's cooperation and data exchange

If certain decrypted data is leaked to the cloud

break the CPA model, need CCA security

One query attack

Suppose attacker can query decryption oracle 1 time

Realistic in many scenarios

Ask to decrypt a malicious ciphertext $f(x)$

$f(x) = c_0 + c_1x$ with $c_0 = 0$, $c_1 = \Delta$

Decryption substitutes x with s

We have : $f(s) = \Delta s$

then the decrypted message equal to s (private key)

Recover private key with only one query

extremely dangerous

other FHE face the same problem

Demo of one query attack

```
def recover_key():
    cc0=0
    cc1=delta
    print "Recover private key successfully:", (s).list()==decrypt([cc0, cc1]).list()
    print "Secret key:", Roundt(s).list()
    print "recovered key", decrypt([cc0, cc1]).list()

s, pk=gen()
rlk, E=gen_rlk(T)
recover_key()
```

Recover private key successfully: True

[illegible]

Default parameter of SEAL

https://github.com/edwardz246003/danger-of-using-homomorphic-encryption/blob/master/CCA_attack.py

Countermeasures

Do not use HE in any scenario that decrypted result may leak to evaluator.

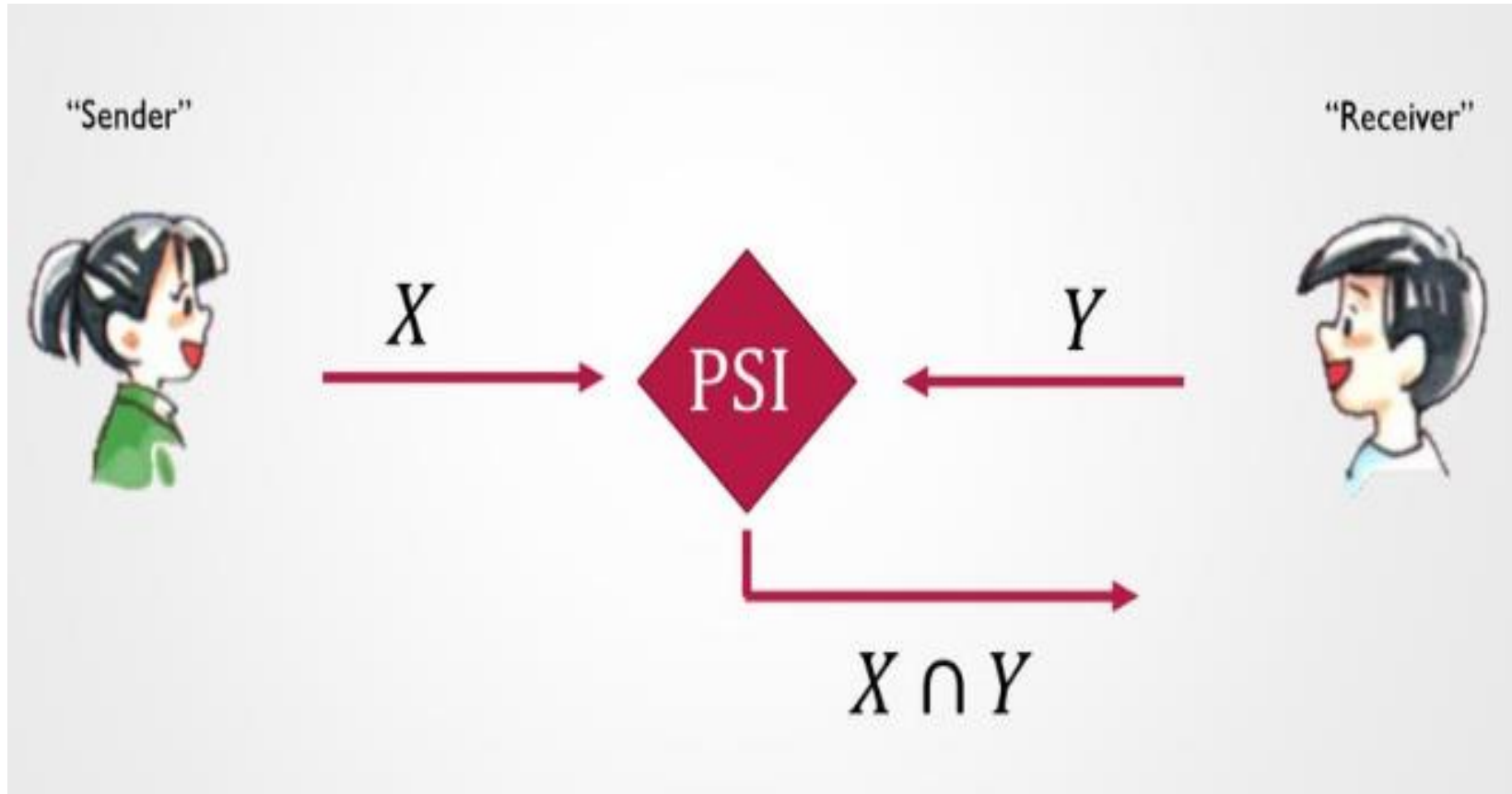
Otherwise, there is no encryption at all.

The decrypted result may leak to evaluator in many scenarios, with or without being noticed.

But how can we make sure there is no leakage?

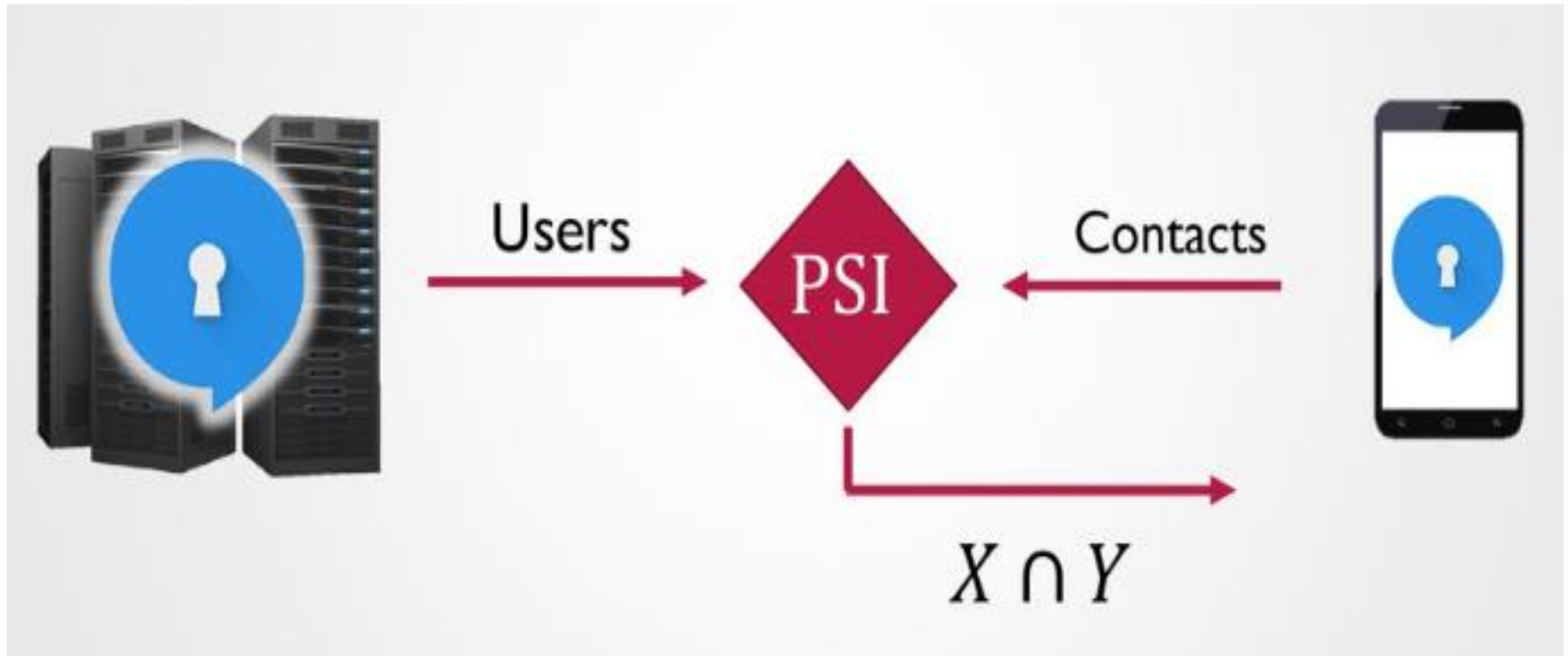
Currently, SEAL team is working on building mitigations on this problem. To detect malicious ciphertext before decrypt it.

Private Set Intersection (PSI)



Without leaking anything else

App: Contact discovery



private contact discovery on E2EE IM (signal...)

Using HE to build PSI FPSI in CCS17 (oversimplified)



local database Y



Encrypt(X) with HE

Send Encrypted X to Server



Compute with local database Y,
get the encrypted $X \cap Y$

Send Encrypted $X \cap Y$ to Client



Decrypt the result,
Get the $X \cap Y$

CCA attack on this scenario



After client get $X \cap Y$.

He found out that $X \cap Y$ are also using signal.

Then he add them as friends

$X \cap Y$ in plaintext



Information leakage

To server.

Server can launch CCA attack!

Lesson learned:

There are always unexpected data flows between data-owner and cloud.
Be extremely careful when using homomorphic encryption.

Hardness of this attack

Difference from previous CCA attack

User will check whether the decryption result is 0

So only 1 bit information leakage per query

We can reveal 1 bit of user's private key using this 1-bit information leakage.

Data owner do not need to leak the plaintext to the attack, any 1bit information leakage will lead to 1 bit key leakage

Security of BFV will drop exponentially

1 bit information leakage is Inevitable in real life application

It's very dangerous

Demo of CCA attack (1 bit infoleak)

```
def recover_key(i):  
    t1=[0 for _ in range(d)]  
    t1[i]=M  
    t2=M  
    cc0=pk[0]+R(t1)  
    cc1=pk[1]+R(t2)  
    return (decrypt([cc0, cc1]).list())[i]  
  
s, pk=gen()  
rlk, E=gen_rlk(T)  
M=delta//4+50  
Recoverd_key=[]  
for i in range(d):  
    Recoverd_key.append(recover_key(i))  
print "Recover private key successfully:", Recoverd_key==s.list()
```

Recover private key successfully: True

Any 1bit information leakage will lead to 1 bit key leakage

https://github.com/edwardz246003/danger-of-using-homomorphic-encryption/blob/master/CCA_attack.py

Another attack



local database Y



Encrypt(X) with HE

Send Encrypted X to Server



Compute with local database Y,
get the encrypted $X \cap Y$.

But most of the HE have no circuit privacy.
Other information except $X \cap Y$ may also leak.

Send Encrypted $X \cap Y$ to Client



Client decrypt the result,
Get the $X \cap Y$

Also get other information on Y

More details can be found on our whitepaper

```

ma=randint(0, t-1) # Alice's input
mb=randint(0, t-1) # Bob's input
print "Alice has input", ma
print "Bob has input", mb
#Alice encrypt her input
#Alice keep u, e1, e2
u=sample_2()
e1=sample_e()
e2=sample_e()
ca=(Roundq(pk[0]*u+e1+delta*ma), Roundq(pk[1]*u+e2))
#Then Alice send the ciphertext ca to bob

#Bob receive ca, and do some homomorphic computation.
cab=(Roundq(ca[0]-delta*mb), Roundq(ca[1])) # plus const mb
r=randint(0, t)
print "Bob choose a random factor, r:", r
cab=(Roundq(r*cab[0]), Roundq(r*cab[1])) # mul a random number r
#Bob respond cab back to Alice

result=decrypt(cab)
print "is mb==mb?", result==0

# A semi-honest Alice can recover the r by using
for i in range(t):
    if Roundq(i*ca[1])==Roundq(cab[1]):
        break
r_prime=i
print "Alice recover r' :", r_prime
print "Is r' equals to r:", r_prime==r

#Alice can use r to recover Bob's input mb
mb_prime=(ma-(int(result)*inverse_mod(r_prime, t)))%t
print "Alice recover mb' ", mb_prime
print "Is mb' equals to mb:", mb_prime==mb

```

Circuit Privacy of SEAL

SEAL doesn't provide circuit privacy on default

Addressed in SEAL handbook

Best practice is “noise flooding”

adding an encrypted 0 to the final result, with “enough” noise

But there is no standard interface of “noise flooding” in SEAL

normal software developer definitely can't play with the magic

Hardness of providing “noise flooding”

Need to know how much noise is needed, this is also some kind of information we need to protect. :(

All practical FHE lib seems have the circuit privacy problem

The crypto community need to solve this

Countermeasures

An improved PSI protocol is published in CCS18

<https://eprint.iacr.org/2018/787>

Secure against malicious party

As for circuit privacy of SEAL and HE

You need a crypto expert to review your implementation

You need professional knowledge on lattice-based crypto

SEAL team is working on provide a standard interface to this problem

Info leakage of Encoder

HE is working on a polynomial ring based on finite field

plaintext is integer, float or string

we need convert them to the ring

IntegerEncoder of SEAL

encode an integer to a polynomial

many to one mapping

Information leakage!

More details can be found on our [whitepaper](#).

I think you don't want the mathematical formula here.

Demo of Encoder

```
m1=IntegerEncoder(2)
m2=IntegerEncoder(2)
c1=encrypt(m1)
c2=encrypt(m2)
result=add(c1,c2)
print decrypt(result)
print IntegerDecoder(decrypt(result))
```

$2 \times X$
4

```
m1=IntegerEncoder(1)
m2=IntegerEncoder(3)
c1=encrypt(m1)
c2=encrypt(m2)
result=add(c1,c2)
print decrypt(result)
print IntegerDecoder(decrypt(result))
```

$X + 2$
4

Demo of information leakage of IntegerEncoder in SEAL

We have $m1$ and $m2$. We want:
 $result = m1 + m2$

For $m1=2$ and $m2=2$, we have
 $result = 4$

For $m1=1$ and $m2=3$, we have
 $result = 4$

But the encoded form are different!
 $2 \times X \neq X+2$

Attack on millionaire problem:

<https://github.com/edwardz246003/danger-of-using-homomorphic-encryption/blob/master/millionaire.py>

Countermeasures

Coding problem may also happen in other HE libs

Be careful when using encode functionality provided by HE lib

The crypto part have a security proof, but the encoding may not

Don't use IntegerEncoder, FloatEncoder in SEAL

They should considered primarily as a demonstrative tool

BatchEncoder is safe in SEAL

Other security issues

HE does not provide the security features as the commonly known encryption algorithms

- HE is not an Authenticated Encryption

- cannot guarantee the integrity of the data

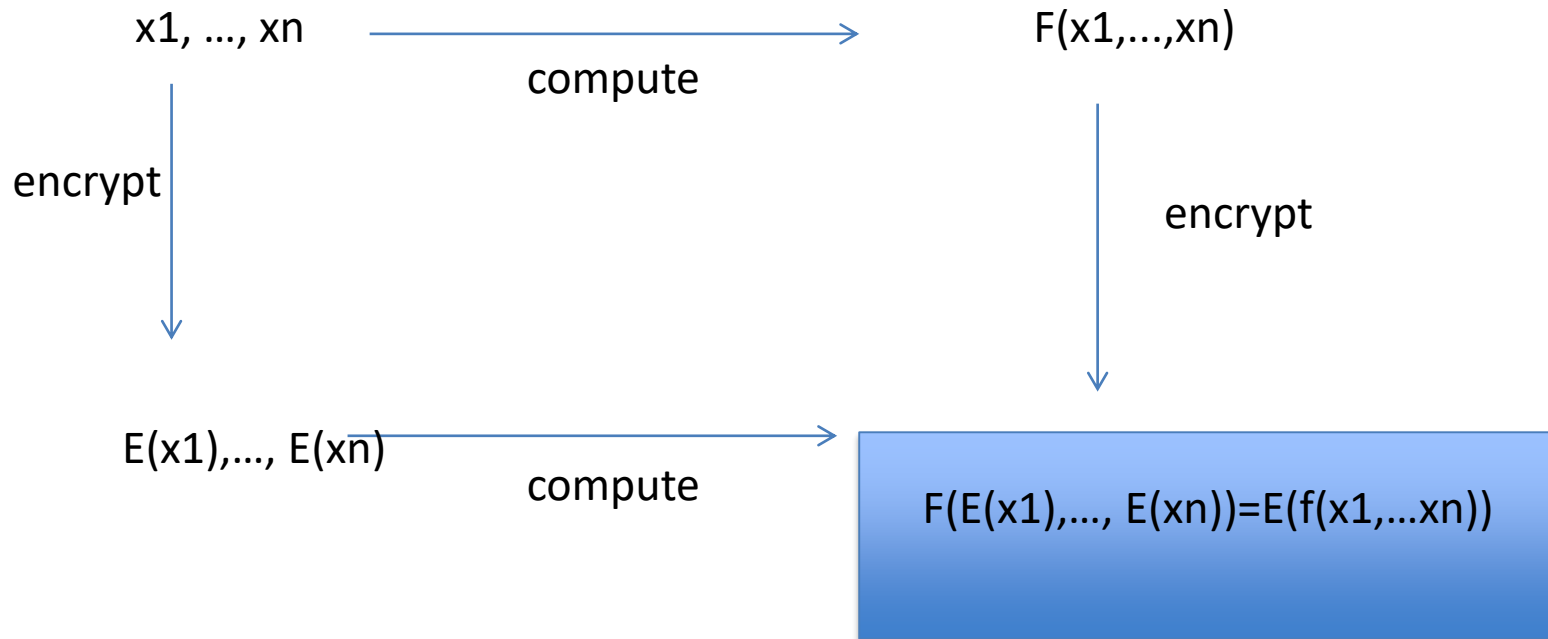
- attacker can use the homomorphic nature of HE to modify the ciphertext

- don't use HE for storage and data transmission directly

We need a standard documentation for HE

- Microsoft is currently leading the development of standard for HE

Is FHE really can compute arbitrary function?



Arbitrary function in FHE means arbitrary addition and multiplication here.

Arbitrary addition and multiplication does not mean you can run arbitrary program

You can't do comparison directly
(if branch is not support here)

Update the famous metaphor



1. Put your gold in the locked box
 2. Keep your key
 3. Let the worker work on it through a glove box **with eyeshade**
 4. Unlock the box and get the jeweler
- This box should be opaque!**

Conclusion

HE is a useful in many scenario

Its performance is improving and acceptable

HE is not omnipotent

It can not really run arbitrary program

HE has many security pitfalls

It's extremely dangerous to use HE without an crypto expert for now

Still need a long way to go

Acknowledgments

We would like to thank

Chen Hong of Alibaba Gemini Lab

Kim Laine of Microsoft Research

For their valuable comments and suggestions to the talk

Thanks



360
WWW.360.CN

SAFETY FIRST