Skoltech

Numerical Linear Algebra

Homomorphic encryption based on approximate eigenvector problem

Melnikov G., Kaziakhmedov E., Kireev K.

December 20, 2018

Homomorphic Encryption Overview

Encryption/Decryption

$$C_{1,2} = Enc(\mu_{1,2}, pk) \quad \mu_{1,2} = Dec(C_{1,2}, sk)$$

Addition

$$C_1 \boxplus C_2 \Leftrightarrow \mu_1 + \mu_2$$

Multiplication

$$C_1 \boxtimes C_2 \Leftrightarrow \mu_1 \times \mu_2$$

This problem was proposed in 1978
but the solution was found only in 2009 (Gentry)

Applications

1. Delegated computing
2. Zero-knowledge proof
3. Holy grail of cryptography

Eigenvector scheme

1. Public Key

$$P \in \mathbb{Z}_q^{n \times n}$$

2. Secret Key

$$s \in \mathbb{Z}_q^n : sP = 0$$

$$Enc(m) = PR + mI = C \in \mathbb{Z}_q^{n \times n}, \text{where } R \leftarrow \mathbb{Z}_q^n$$

$$Dec(C) = sC = s(PR + mI) = ms$$

It's not an Encryption Scheme since s is an exact eigenvector!

Tweaked eigenvector scheme

1. Secret Key

$$s = [s' \quad -1], \quad s' \leftarrow \mathbb{Z}_q^{n-1}, \quad sP \approx 0$$

2. Public Key

$$P = \begin{bmatrix} P' \\ s'P' + e \end{bmatrix}, \text{with } P' \leftarrow Z_q^{n-1 \times m},$$

e is a small error

$$Enc(\mu) = PR + \mu I = C \in \mathbb{Z}_q^{n \times m}, \text{where } R \leftarrow \{0,1\}^{n \times m}$$

$$Dec(C) = sC = s(PR + \mu I) = -eR + \mu s$$

if $\mu = 0$, $||sC||_\infty = ||-eR||_\infty$ is small, if $\mu = 1$, $||sC||_\infty \approx ||\mu s||_\infty$

Approximate eigenvector scheme

1. Secret Key

$$s = [s' \quad -1], \quad s' \leftarrow \mathbb{Z}_q^{n-1}, \quad sP \approx 0$$

2. Public Key

$$P = \begin{bmatrix} P' \\ s'P' + e \end{bmatrix}, \text{with } P' \leftarrow Z_q^{n-1 \times m},$$

e is a small error

$$Enc(\mu) = PR + \mu G = C \in \mathbb{Z}_q^{n \times m}, \text{where } R \leftarrow \{0, 1\}^{n \times m}$$

$$Dec(C) = sC = s(PR + \mu G) = -eR + \mu sG$$

if $\mu = 0$, $||sC||_\infty = ||eR||_\infty \leq m\|e\|_\infty$ is small
if $\mu = 1$, $||sC||_\infty \approx ||sG||_\infty$

Primitive matrix

$$G = I \otimes g, \quad g = \begin{bmatrix} 1 & 2 & \dots & 2^{[log\ q]} \end{bmatrix}$$

$$gv = a, a \in \mathbb{Z}_q \text{ - solution is bit decomposition of } a$$

$$Gv = a, a \in \mathbb{Z}_q^n \text{ where } v \in \mathbb{Z}^m$$

$$G^{-1} - \text{ expands } a \text{ into component-wise binary decomposition}$$

Errors accumulation

Addition

$$C_1 \boxplus C_2 \Leftrightarrow s(C_1 + C_2) = \underbrace{(e_1 + e_2)}_{\text{new error}} + \underbrace{(\mu_1 + \mu_2)}_{\text{result}} sG$$

Multiplication

$$C_1 \boxtimes C_2 \Leftrightarrow s(C_1 G^{-1}(C_2)) = \underbrace{(eG^{-1}(C_2) + \mu_1 e_2)}_{\text{new error}} + \underbrace{\mu_1 \mu_2}_{\text{result}} sG$$

Error bound for multiplication

$$\|e_{mult}\|_\infty \leq m\|e_1\|_\infty + \mu_1\|e_1\|_\infty$$

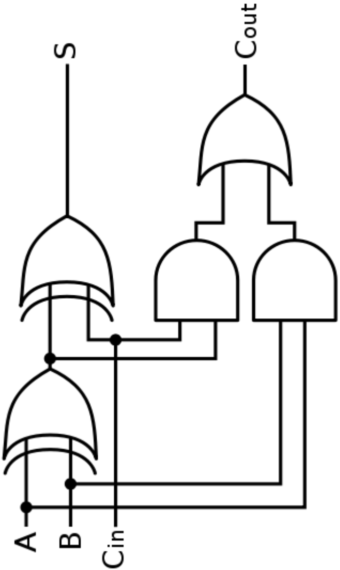
Circuits

NAND

$$1 - \mu_1\mu_2$$

XOR

$$\mu_1 + \mu_2 - 2\mu_1\mu_2$$



Error growth for circuits

NAND (Shiffer stroke)

$NAND(C_1, C_2) = (I - C_1 G^{-1} C_2^{-1})$ is fuctionally complete, so any boolean function can be done via NAND

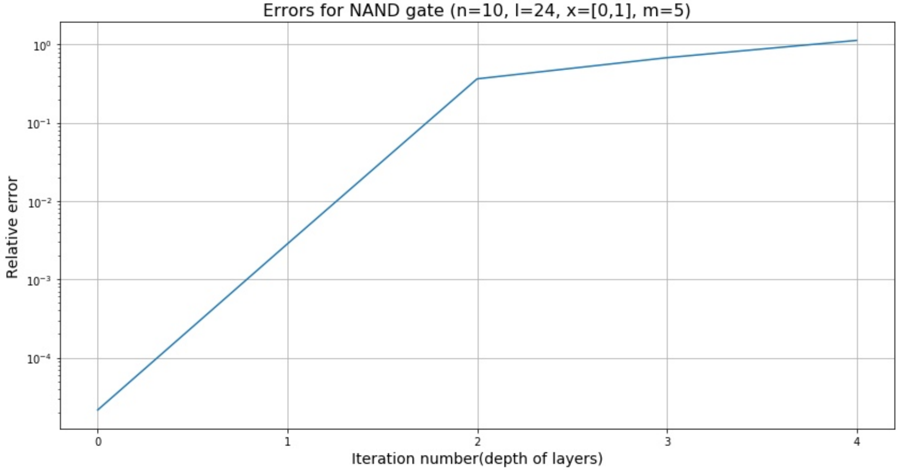
NAND Error

We can decode NAND curcuit with depth L when $\frac{q}{B} > 8(N+1)^L$
B - is bound for initial error

Bootstraping

In theory error can be reduced with technique called bootstrapping, but it is quite sophisticated

Error growth



Semi-live demo, one bit encryption

In [1065]:
```
print("1. Initialize with following parameters: n = %d, l = %d, x=[%d, %d], m=%d" % (10, 25, 0, 1, 5))
fhe = GSW_FHE(n=10, q=2**24, l=25, x=(0, 2), m=5)
print("2. Generate secret key")
t, v = fhe.SecretKeyGen()
print("3. Generate public key")
A = fhe.PublicKeyGen(t, v)
print("4. Obtain a ciphertext of 1")
C = fhe.Enc(A, 1)
print("5. Obtain a plaintext")
P = fhe.Dec(v, Cout)
print("Result: ", P)
```

```
1. Initialize with following parameters: n = 10, l = 25, x=[0, 1], m=5
m:  5
2. Generate secret key
3. Generate public key
Error norm:  1.0
q/4:  4194304.0
4. Obtain a ciphertext of 1
5. Obtain a plaintext
Result:  1
```

Semi-live demo, full-adder

```
In [1088]: print("1. Init with following parameters: n = %d, l = %d, x=[%d, %d], m=%d" % (10, 25, 0, 1, 5))
fhe = GSW_FHE(n=10, q=2**24, l=25, x=(0, 2), m=5)
print("2. Generate secret key")
t, v = fhe.SecretKeyGen()
print("3. Generate public key")
A = fhe.PublicKeyGen(t, v)
print("4. Obtain a ciphertext of 2")
C1 = num2ciphervec(A, 2)
print("5. Obtain a ciphertext of 3")
C2 = num2ciphervec(A, 3)
print("6. Sum ciphertexts")
C3 = add(C1, C2)
print("7. Decipher sum")
u = [fhe.Dec(v, _) for _ in C3]
print("Result: ", u)
```

```
1. Init with following parameters: n = 10, l = 25, x=[0, 1], m=5
m:  5
2. Generate secret key
3. Generate public key
Error norm:  1.0
q/4:  4194304.0
4. Obtain a ciphertext of 2
5. Obtain a ciphertext of 3
6. Sum ciphertexts
7. Decipher sum
Result:  [1, 0, 1, 0]
```

Conclusion

1. The described scheme was implemented
2. Matrix multiplication was optimized
3. Experiments were carried out
4. The scheme was proved to be inefficient now

References

Gentry, Craig, and Dan Boneh (2009)
A fully homomorphic encryption scheme.

Gentry, Sahai, Waters (2013)
Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based

Alperin-Sheriff, Chris Peikert (2014)
Faster Bootstrapping with Polynomial Error

Appendix A: Learning With Error

Theorem (search-*LWE* problem is computationally hard)

The search-LWE problem is to find the secret $\mathbf{s}$ *given access to* $\mathcal{O}_s^n$.

1. $\mathbf{a} \leftarrow \mathbb{Z}_q^n$, *is chosen freshly at random*
2. $\mathbf{s} \in \mathbb{Z}_q^n$, *is a secret (the same for every sample)*
3. $e \leftarrow \chi$, *is chosen freshly*

Oracle $\mathcal{O}_s^n$ *which outputs samples of the form* $(\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e)$