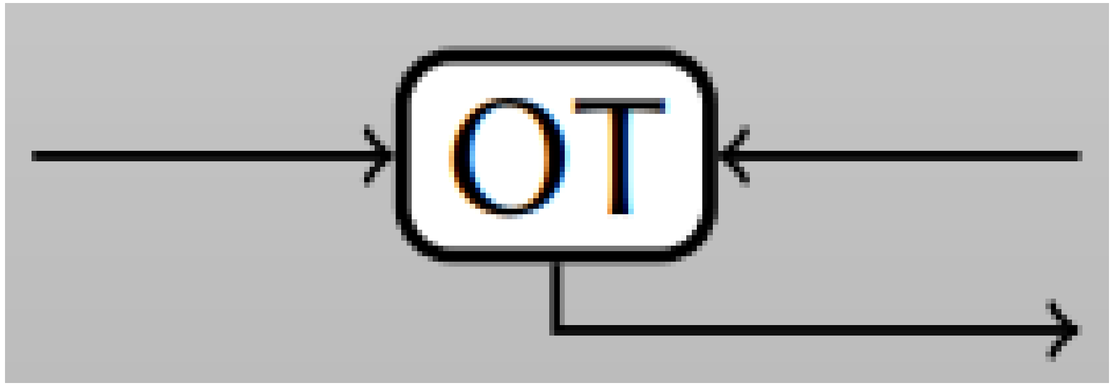
CSE 539: Applied Cryptography
Week 13: Homomorphic Encryption

Ni Trieu (ASU)

Reading: https://www.microsoft.com/en-us/research/project/microsoft-seal/

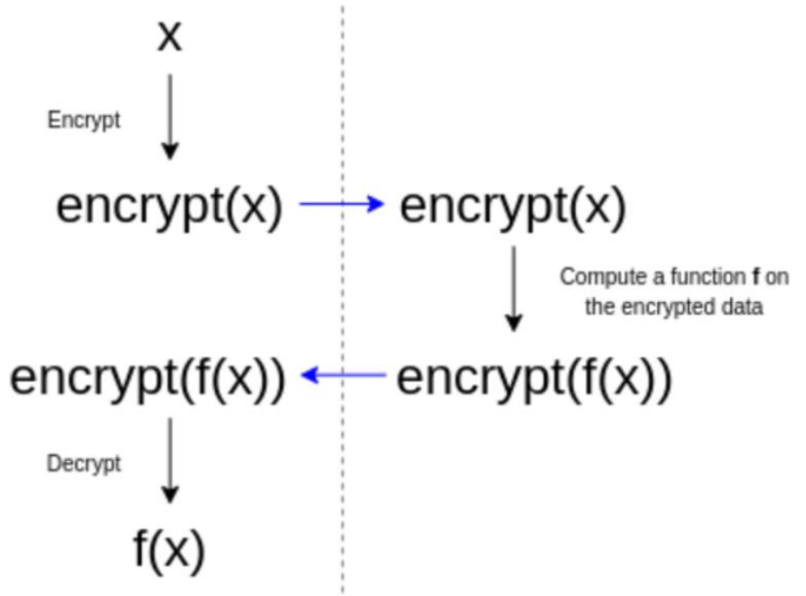
https://eprint.iacr.org/2019/939

Recap: Oblivious Transfer



Outline

- ~~What is Secure Computation?~~
- How does it work?
  - Yao's Protocol (Garbled Circuit)
    - Oblivious Transfer
  - Homomorphic Encryption



Homomorphic Encryption

- Public-key Encryption
  - Three procedures: KeyGen, Enc, Dec
    - (sk,pk) ← KeyGen($)
      - Generate random public/secret key-pair
    - c ← Enc(pk, m)
      - Encrypt a message with the public key
    - m ← Dec(sk, c)
      - Decrypt a ciphertext with the secret key
  - "Informal" Security: c reveals nothing about m (c looks random)

Homomorphic Encryption

- Homomorphic Encryption (HE)
  - Four procedures: KeyGen, Enc, Dec, Eval
    - (sk,pk) ← KeyGen($)
      - Generate random public/secret key-pair
    - c ← Enc(pk, m)
      - Encrypt a message with the public key
    - m ← Dec(sk, c)
      - Decrypt a ciphertext with the secret key
    - c’ ← Eval(pk, f, c)
      - Evaluate a function f on encrypted c.

Homomorphic Encryption

- Homomorphic Encryption (HE)
  - Five procedures: KeyGen, Enc, Dec, Eval (Add, Mul)
    - (sk,pk) ← KeyGen($)
      - Generate random public/secret key-pair
    - c ← Enc(pk, m)
      - Encrypt a message with the public key
    - m ← Dec(sk, c)
      - Decrypt a ciphertext with the secret key
    - c’ ← Eval(pk, f, c)
      - Evaluate a function f on encrypted c.

c_1 ← Enc(pk, m_1)
c_2 ← Enc(pk, m_2)
c ← Add(pk, c_1, c_2)
Add two ciphertexts using pk
$m_1 + m_2$ ← Dec(sk, c)

c ← Mul(pk, c_1, c_2)
Multiply two ciphertext using pk
$m_1 * m_2$ ← Dec(sk, c)

- If a HE scheme supports either Add() or Mul()
  => Partially HE
- If a HE scheme supports both Add() and Mul()
  => Fully HE (*arbitrary computation?)*

A Toy HE Scheme

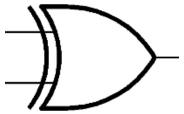
- Encryption: Double the plaintext. $x \rightarrow 2x$
- Decryption: Halve the ciphertext. $x \rightarrow x/2$

Homomorphic Encryption

- Why do ADD and MUL => *arbitrary computation*?

ADD

=

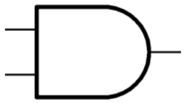


XOR

0 XOR 0	0
1 XOR 0	1
0 XOR 1	1
1 XOR 1	0

MUL

=



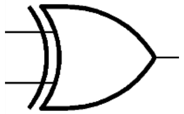
AND

0 AND 0	0
1 AND 0	0
0 AND 1	0
1 AND 1	1

Homomorphic Encryption

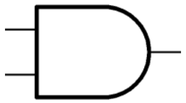
- Why do ADD and MUL => *arbitrary computation*?
- Because {XOR,AND} is Turing-complete

  => any function is a combination of XOR and AND gates



XOR

0 XOR 0	0
1 XOR 0	1
0 XOR 1	1
1 XOR 1	0



AND

0 AND 0	0
1 AND 0	0
0 AND 1	0
1 AND 1	1

Partially HE Cryptosystems

- ElGamal
  - Keygen: $h = g^x,\ sk = x)$ for a generator g
  - $\mathrm{Enc(h, m)} = (\mathrm{g}^r, mh^r)$
  - $\mathrm{Dec(x,} c_1, c_2) = c_2/c_1^x$
  - $\mathrm{Enc(h, m_1)} * \mathrm{Enc(h, m_2)} = \mathrm{Enc(h, m_1} + m_2)$

Partially HE Cryptosystems

- Paillier

KeyGen:

- Let p and q be distinct primes
- Let $N = pq$
- Let $\lambda = (p-1)(q-1)$

$\Rightarrow$ Public key: $pk = N$; Secret key: $sk = \lambda$

$\mathrm{Enc}_{pk}(m) = g^m r^N (mod\ N^2)$ where r is randomly chosen s.th $r \in Z_N^*, \gcd(r, N) = 1$ (e.g. r =0, .., N-1)

$\mathrm{Dec}_{sk}(c) = \frac{(c^\lambda\ mod\ N^2 - 1)}{N}\ \lambda^{-1}\ (mod\ N)$ where $\frac{a}{b}$ is the quotient of a divided by b

Fully HE Cryptosystems

- People tried do Fully HE for years and years with no success.
- Until, in October 2008, **Craig Gentry** came up with the **first** fully HE scheme

Homomorphic Encryption

Making cloud computing more secure

1 comment
ERICA NAONE
Tuesday, April 19, 2011

Ciphering: Gentry's system allows encrypted data to be analyzed in the cloud. In this example, we wish to add 1 and 2. The data is encrypted so that 1 becomes 33 and 2 becomes 54. The encrypted data is sent to the cloud and processed: the result (87) can be downloaded from the cloud and decrypted to provide the final answer (3).

Fully HE Cryptosystems

- Efficient (public-key) HE scheme (based on LWE assumption)

PARAMETERS: The key space χ, the plaintext space $R_p = \mathbb{Z}_p[X]/(X^N + 1)$, the ciphertext space $R_q = \mathbb{Z}_q[X]/(X^N + 1)$, and $\Delta = \lfloor q/p \rfloor$

HE.KeyGen() $\to (sk, pk)$

- Sample $s \leftarrow \chi$
- Sample $a \leftarrow R_q$ and $e \leftarrow \chi$
- Output (sk, pk) where $sk = s$ and $pk = ([-a \cdot s + e]_q, a)$

HE.Encrypt$(pk, m) \to ctx$

- Sample $u, e_1, e_2 \leftarrow \chi$
- Let $p_0 = pk[0]$, $p_1 = pk[1]$
- Compute $c_0 = [p_0 \cdot u + e_1 + \Delta \cdot m]_q$ and $c_1 = [p_1 \cdot u + e_2]_q$
- Output (c_0, c_1)

HE.Decrypt$(sk, ctx) \to m$

- Let $c_0 = ctx[0]$, $c_1 = ctx[1]$
- Compute $v = [c_0 + c_1 \cdot s]_q$
- Output $[\Delta \cdot v]_p$

Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144
https://eprint.iacr.org/2012/144

Fully HE Cryptosystems

- Eval(pk, f, c): If we can compute XOR and AND on encrypted bits, we can compute everything.