NYU

Implementation and Comparison of Two-party secure function evaluation

WenQiang Bao(wb854)
Min Kim(mjk884)

December 7, 2020

NYU

Contents

Contents

- Function F
- Design of 2 secure 2-party computation protocols for F
- Implementation
- Analyzing and comparing the performance
- Circuit Compiler(Obliv-C)

NYU

Function F

Yao's Millionaires' ("greater than" or "GT") Problem

- Determine who is richer between two parties
- No information about a party's amount of assets is leaked to the other party
- Yao gave the first protocol for solving the secure comparison problem

  (Exponential in time and space requirements)

NYU

Design of 2 Secure 2-Parties Computation Protocols for F

Blake-Kolesnikov Protocol(BT04)

- Two-round GT protocol based on an **additive** homomorphic (e.g. Paillier) cryptosystem
- Defined a primitive called **S-COT**, a stronger version of COT
  - Conditional OT: S has a secret s —>R, OT occurs iff a public predicate Q(x,y) = 1, x is S's input(can be not private), y is R's private input.
  - Stronger-COT: 1. x is private, 2. Not revealing Q(x,y) to R
- Exploit the structure of the GT predicate in a novel way
- More **efficient** and **flexible** than the best previously known in the semi-honest setting with the **unbounded receiver**

BT04 Protocol in Detail

1. Alice with private input $\mathrm{x}= x_n\, x_{n-1}\, .\,.\,.\, x_1$ does the following:
   - Runs key generation phase
   - Encrypts x bit-wise and sends $\mathrm{pk}, \mathrm{Enc}(x_n\ ), .\,.\,.\, ,\mathrm{Enc}(x_1)$ to Bob

2. Bob with private input $\mathrm{y} = y_n y_{n-1}.\,.\,.\, y_1$ does the following for each $\mathrm{i} = 1, .\,.\,.\,.\,, \mathrm{n}$:
   - Computes $\mathrm{Enc}(d_i) = \mathrm{Enc}(x_i - y_i)$
   - Computes $\mathrm{Enc}(f_i) = \mathrm{Enc}(x_i - 2x_i y_i + y_i\ )$
   - Computes $\mathrm{Enc}(\gamma_i) = \mathrm{Enc}(2\gamma_{i-1} + f_i)$ where $\gamma_0 = 0$
   - Computes $\mathrm{Enc}(\delta_i) = \mathrm{Enc}(\mathrm{d}_i + \mathrm{r}_i(\gamma \mathrm{i} - 1))$ where $\mathrm{r}_i \in_R Z_n$
   - Randomly permutates $\mathrm{Enc}(\delta_i)$ and sends to Alice

3. Alice obtains $\mathrm{Enc}(\delta_i)$ from Bob, then decrypts.
   - If there exists a value $\mathrm{v} \in \{+1,-1\}$ and output v

   →If $\mathrm{x} > \mathrm{y}$, the output value $\mathrm{v} = +1$; if $\mathrm{x} < \mathrm{y}$, $\mathrm{v} = -1$

- Two-round GT protocol based on **multiplicative**(e.g., El Gamal) **or additive**(e.g., Paillier) homomorphic cryptosystem
- Reduce GT problem to the intersection problem of two sets
- Multiplicative homomorphic encryption scheme is more efficient than an additive one

- Main Idea: Reduce the GT problem to the set intersection problem
- 0-encoding: S_s^0={$s_n s_{n-1} \ldots s_{i+1}1 \mid s_i$=0, 1 ≤ i ≤ n}
- 1-encoding: S_s^1={$s_n s_{n-1} \ldots s_i 1 \mid s_i$=1, 1 ≤ i ≤ n}
- Both S_s^0 and S_s^1 have at most n elements
- When encode x into its 1-encoding S_x^1 and y into its 0-encoding S_y^0 , x > y if and only

  if S_x^1 and S_y^0 has a common element

Example: x = 110 and y = 010, S_x^1 = {1,11} , S_y^0 = {1,011}

LT05 Protocol in Detail

1. Alice with private input x= $x_n\, x_{n-1}\, .\,.\,.\, x_1$ does the following:
   - Run G to choose a key pair (pk, sk) for (E,D)
   - Prepare a $2 \times$ n-table T[i, j], i $\in$ {0, 1}, $1 \le j \le n$, such that T[x_i, i] = E(1) and T[$\overline{x_i}$, i] = E(r_i) for some random $r_i \in G_q$
   - Send T to Bob

2. Bob with private input y = $y_n y_{n-1} .\,.\,.\, y_1$ does the following:
   - For each $t = t_n t_{n-1}\, .\,.\,.\, t_i \in S_y^0$, compute c_t = T[t_n, n] $\odot$ T[t_{n-1}, n-1] . . . T[t_i, i]
   - Prepare $l = n - |S_y^0|$ random encryptions $z_j = (a_j, b_j) \in G_q^2$, $1 \le j \le l$
   - Scalarize c_t's and permutate c_t's and z_j's randomly as $c_1, c_2, .\,.\,.\,, c_n$
   - Send $c_1, c_2, .\,.\,.\,, c_n$ to Alice

3. Alice decrypts D(c_i) = m_i, $1 \le i \le n$, and determine x > y if and only if some m_i = 1
   - When $x \le y$, negligible probability that GT protocol returns a wrong answer due to randomization
   - Using additive homomorphic encryption, replace E(1) by E(0) in setting up the table T

   → x > y if and only if some mi = 0

NYU

Analyzing and Comparing the Performance

- The receiver needs n encryptions and n decryptions
- The sender needs n modular multiplications in the 2a step,

  n modular multiplications and n inversions in the 2b step,

  2n modular multiplications in the 2c step,

  and $(2 + \log N)n$ modular multiplications in the second step
- Each inversion takes one modular multiplication
- Overall, the protocol needs 4n modular exponentiations (mod N^2),

  and 7n modular multiplications (mod N^2)
- The communication cost is n ciphertexts for the receiver and n ciphertexts for the sender

  → $4n \log N$ bits

Lin-Tzeng Protocol(LT05) Analyzing

- In Step 1, Alice encrypts n 1′s
- In Step 2, Bob computes c_t, $t \in S_y^0$, by reusing intermediate values

  → $(2n - 3)$ multiplications of ciphertexts, n scalar operations at most
- In Step 3, Alice decrypts n ciphertexts.
- Overall, GT protocol needs $5n \log p + 4n - 6$ modular multiplications

  → $5n \log p + 4n - 6 = n \times 2 \log p + 2 \times (2n-3) + n \times 2 \log p + n \times \log p)$
- Communication complexity : $6n \log p = 3n \times 2 \log p$ bits

Comparing BT04 VS LT05

	Computation of Alice	Computation of Bob	Total Computation	Communication
BK04	$12n \log N$	$4n \log N + 28n$	$16n \log N + 28n$	$4n \log N$
LT05	$3n \log p$	$2n \log p + 4n - 6$	$5n \log p + 4n - 6$	$6n \log p$

- Both constructions are secure in the semi-honest setting
- In the malicious setting, each round requires additional messages to assure legality of the sent messages
- The techniques are mostly based on non-interactive zero-knowledge proof of knowledge

```
$ python test.py
--- Testing LinTzeng Protocol ---
[1]Testing Paillier based...
Encryption Scheme: Paillier
Modulus Length: 256
Input Length: 100
---100/100 passed 8.53471922874 seconds ---

[2]Testing ElGamal based...
Encryption Scheme: ElGamal
Modulus Length: 256
Input Length: 100
---100/100 passed 7.4218378067 seconds ---

--- Testing Blake-Kolesnikov's Protocol ---
Encryption Scheme: Paillier
Modulus Length: 256
Input Length: 100
---100/100 passed 8.9105682373 seconds ---
```

- Result matches the analysis that **LT05** is faster than BT04
- Tested LT05 twice based on Paillier and El Gamal respectively where the one with El Gamal is faster

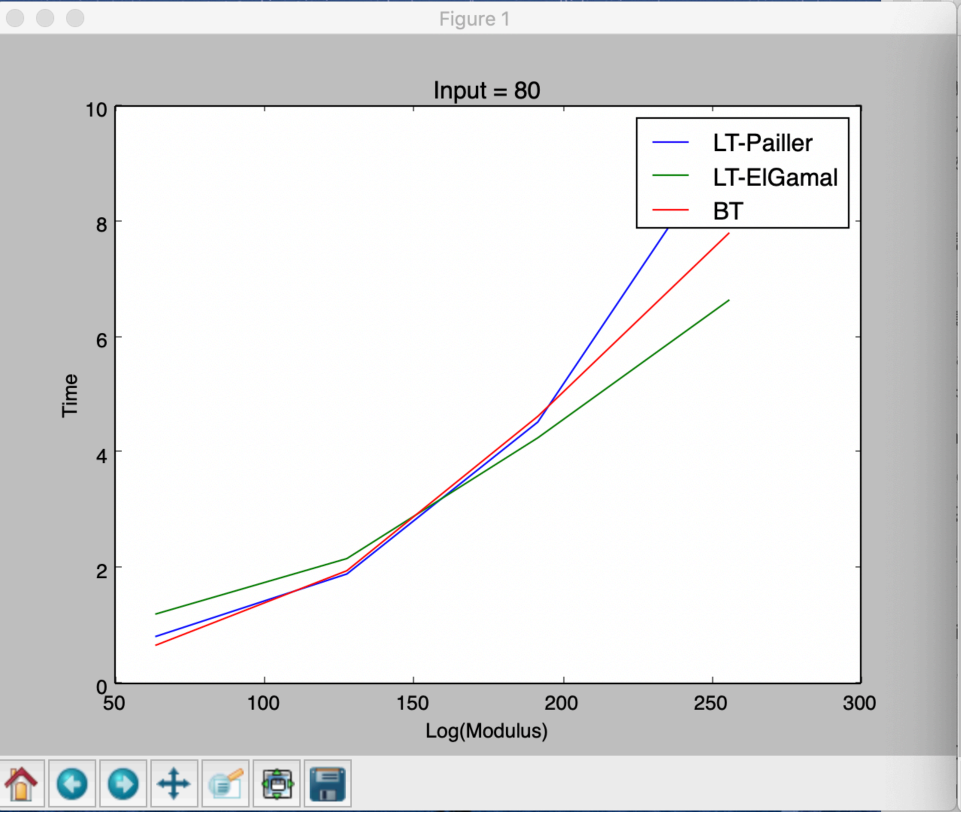
Figure 1
Input = 80
LT-Pailler
LT-ElGamal
BT
Time
Log(Modulus)
0
2
4
6
8
10
50
100
150
200
250
300

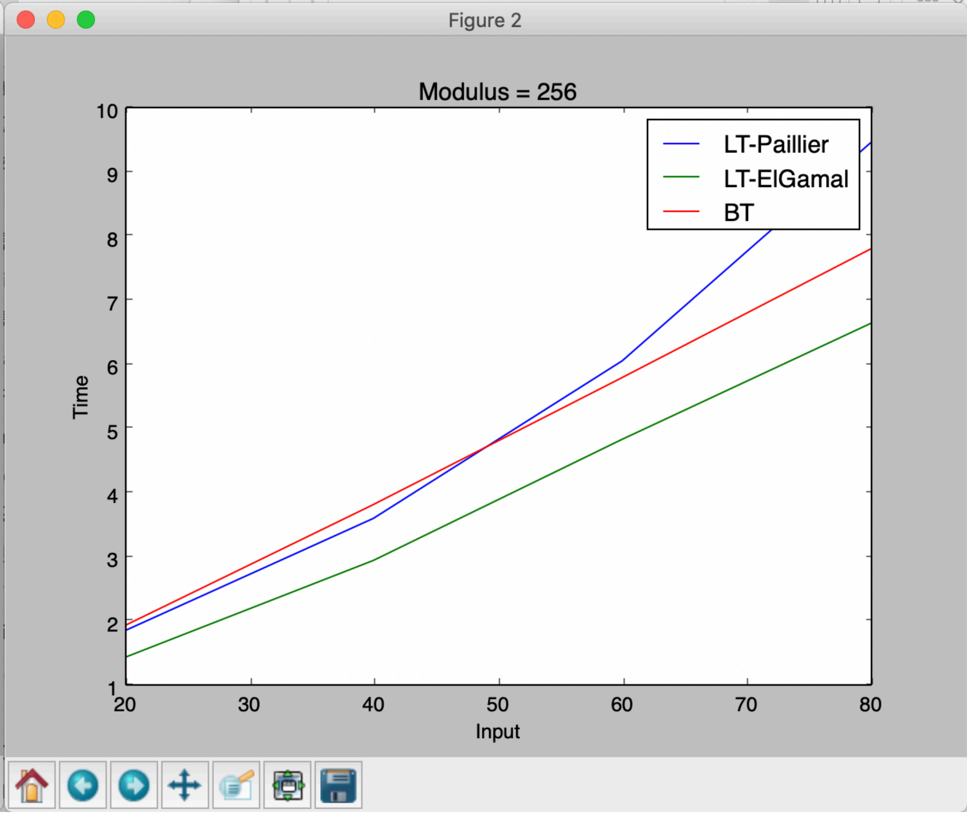
Figure 2
Modulus = 256
LT-Paillier
LT-ElGamal
BT
Time
Input
1
2
3
4
5
6
7
8
9
10
20
30
40
50
60
70
80

Obliv-C

https://github.com/samee/obliv-c

a simple GCC wrapper that makes it easy to embed secure computation protocols inside regular C programs.

write in Obliv-C(oc) langauge and compile/link it with the project.

General but slow than previous function-specific protocol

Project2 > gc > million > million.oc

```
#include<obliv.oh>
#include"million.h"

void millionaire(void* args)
{
    protocolIO *io=args;
    obliv int v1,v2;
    bool eq,lt;

    v1 = feedOblivInt(io->mywealth,1);
    v2 = feedOblivInt(io->mywealth,2);
    revealOblivBool(&eq,v1==v2,0);
    revealOblivBool(&lt,v1<v2,0);
    io->cmp = (!eq?lt?-1:1:0);
}
```

Reference

- Ian F Blake and Vladimir Kolesnikov. Strong conditional oblivious transfer and computing on intervals. In Advances in Cryptology-ASIACRYPT 2004, pages 515–529. Springer, 2004.
- Hsiao-Ying Lin and Wen-Guey Tzeng. An efficient solution to the millionaires ′problem based on homomorphic encryption. In Applied Cryptography and Network Security, pages 456–466. Springer, 2005.
- G. Di Crescenzo, R. Ostrovsky, and S. Rajagopalan. Conditional oblivious transfer and time-released encryption. In Proc. CRYPTO 99, pages 74–89. Springer-Verlag, 1999.
- Marc Fischlin. A cost-effective pay-per-multiplication comparison method for millionaires. In RSA Security 2001 Cryptographer’s Track, pages 457–471. SpringerVerlag, 2001.
- Marcella Hastings, Brett Henenway, Daniel Noble, and Steve Zdancewic. SoK: General Purpose Compilers for Secure Multi-Party Computation.
- https://github.com/samee/obliv-c