**Project 2 (CS6601) – Implement the Yao’s Millionaire problem using FastGC**

*Azharul Islam, Armita Abedijaberi, San Yeung*

*Date: 10-14-2015*

**(PROJECT REPORT)**

1. We are generating the circuit file for GCParser and the input file of each party from python code. The name of the python file is generator.py. Python file takes all the necessary inputs from a text file called input.txt.
2. input.txt file has 5 lines. Following table describes meaning of each of the lines

|  |  |  |
| --- | --- | --- |
| Line Number | Meaning | Data Type |
| 1 | Vector Size | Positive integer |
| 2 | Domain size | Positive integer |
| 3 | Threshold value | Positive integer |
| 4 | Alice’s input vector | Vector of integers |
| 5 | Bob’s input vector | Vector of integers |

1. After reading the input.txt file, python file generates 3 files:

|  |  |
| --- | --- |
| File Name | Function |
| multiplication.cir | Contains the circuit implementation for secure 2 party dot product computation for the given vector size and domain size. |
| server\_input | Contains Bob’s input which is meant to be secret to Bob only |
| client\_input | Contains Alice’s input which is meant to be secret to Alice only |

1. Once the circuit file is generated, it could be run from GCParser. Please note that input for Bob and Alice can be changed by editing server\_input and client\_input files respectively. Change in input file should follow the vector size and domain size mentioned in the input.txt file.
2. For instance, server\_input and client\_input files could contain following inputs for vector size 3 and domain size 4:

|  |  |  |
| --- | --- | --- |
| File Name | Whose Input | File Content |
| server\_input | Bob | b1 5  b2 3  b3 9 |
| client\_input | Alice | a1 2  a2 3  a3 6  threshold 10 |

In this case, dot product will be



Finally, the dot product will be checked with the threshold value:



Since the dot product is greater than the threshold value, output is 1.

Before we run the program, we need to establish a folder named **/results** in which output files will be stored. After running the program, 2 output files will be generated. These files contain the time that the client and server took to run.

|  |  |
| --- | --- |
| siclientout | siserverout |
| Time (ms) Starting program: 1444870246914 (0.0, 0.0)  1  Elapsed time (ms) on circuit preparation: 420 (0.0, 0.0)  Time (ms) right before NPOT public key generation: 1444870247420 (0.0, 0.0)  Elapsed time (ms) on NPOT public key generation: 55 (0.0, 0.0)  Elapsed time (ms) on OT preparation: 880 (0.0, 0.0)  Elapsed time (ms) on transfer inputs: 141 (0.0, 0.0)  Elapsed time (ms) on execution of circuit: 102 (0.0, 0.0)  output:  richer = 1 | Time (ms) Starting program: 1444870246104 (0.0, 0.0)  waiting for client to connect  client has connected  1  Elapsed time (ms) on circuit preparation: 1231 (0.009765625, 0.009765625)  Elapsed time (ms) on OT preparation: 1041 (13.1025390625, 8.97265625)  Elapsed time (ms) on transfer inputs: 156 (0.966796875, 7.203125)  Elapsed time (ms) on execution of circuit: 32 (1.0, 0.0)  output:  richer = 1 |

1. Implementation Details and Algorithms:
   1. Since GCParser does not have multiplication function, we have to implement it ourselves. We used python code to generate corresponding circuit file for multiplication. In order to determine dot product we just needed to repeat the multiplication vector size number of times and add all the results of individual multiplication. Then we compare the result with the threshold value.
   2. For example from point 5 we can see that a1 needs to be multiplied with b1. So, first we are representing both number to their binary value. Here, a1 is multiplicand and b1 is multiplier. Here, a1=2=0010 and b1=5=0101. We select each bit in b1 and construct a new number with the same domain size by repeating this bit. For example first bit of b1 is 1. So the new number will be 1111. There will be 4 new numbers for b1: {b11, b12, b13, b14 : 1111, 0000, 1111, 0000}. Then we do AND operation for each new number (eg. b11, b12…) with a1. For each of the AND results (0010, 0000, 0010, 0000), we perform shift operation by using the concatenation function of GCParser. In this example the shifted values are 00000010, 00000000, 00001000 and 00000000. Finally we add all the shifted results together to get the multiplication result (00001010). Here to make this add operation more efficient, we are doing add operation pairwise. Incase we have odd number of values to be added; we make it even by having the last value doing an OR operation with itself.
   3. We repeat step b domain size number of times
   4. Then we add all the multiplication results together followed by a comparison with the threshold value.