1 Using Our Solution

One can use our implementation (an executable program) with ease by providing three files that contain configuration, actual data, and the binary circuit representing the output function f.

1.1 Compilation

In Windows environment, we provide a solution file for Visual Studio 2008/2010. In Linux environment, we provide a makefile,

1.2 Configuration

Figure 1 describes the configuration for two-party case. The configuration file for the provider (resp., the customer) is inputp.txt (resp., inputc.txt).

The meaning of the fields is straightforward but explained in the following to be complete:

- `num_parties`: It represents the number of parties participating in the protocol
- `pid`: It represents the identifier of the participating party. The id starts from 0, and the customer must have the last id (i.e., num_parties-1).
- `address-book`: It represents the name of the file that contains the addresses of the parties. Each address in the file is of a form (pid, ip-address, port).
- `circuit`: It represents the name of the file that contains the circuit of interest.
- `seed`: It represents a random seed necessary for SHA-1.
- `p`: It is a safe prime number to be used in the oblivious transfer. It must be the same among all the parties. We recommend p to be at least 512 bits long.
- `g`: It is a generator of the subgroup $QR(\mathbb{Z}_p^*)$. It must be the same among all the parties.

1.3 Providing input

The input is provided via the input file given in the configuration. The formation is simply a Boolean form. For example, the file `inputp.txt` (likewise `inputc.txt`) will look as follows:

```
1 0 1 0 1 1 1 ...
```

Of course, the actual data must be dependent on the circuit that is to be computed, description of which is described below.

In `configp.txt`:

```
num_parties  2
pid 0
address-book addresses.txt
circuit circ.bin
input inputp.txt
seed 11323434087201
p 893609795...799235731486130087
g 766091584...167705188057761352
```

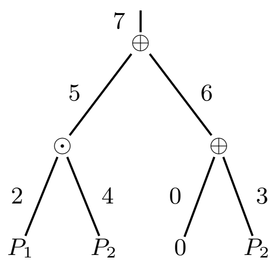
In `configc.txt`:

```
num_parties  2
pid 1
address-book addresses.txt
circuit circ.bin
input inputc.txt
seed 39347104378941
p 893609795...799235731486130087
g 766091584...167705188057761352
```

In `addresses.txt`:

```
0 192.168.1.4 7766
1 192.168.1.5 7767
```

Figure 1: Configuration Example



```
n 2
d 7 5 2
i 0 2 2
i 1 3 4
g 0 16 -1 -1 0
g 1 16 -1 -1 0
g 2 16 -1 -1 1 5
g 3 16 -1 -1 1 6
g 4 16 -1 -1 1 5
g 5 1 2 4 1 7
g 6 2 0 3 1 7
g 7 34 5 6 0
```

Figure 2: Circuit Example

1.4 Running the program

Using our implementation is simple. For GMW, following the example in the Figure 1, the provider will run:

```
>mpc.exe configp.txt
```

Similarly, the customer will run:

```
>mpc.exe configc.txt
```

For YAO, one can run 2pc.exe with a configuration file as above.

2 Providing Circuits

2.1 Circuit as a file

One can use our implementation with ease by providing three files that contain configuration, actual input, and the Boolean circuit representing the function f of interest. A circuit may be described as a file using the fields below. We show an example circuit in Figure 2.

- `n`: It represents the number of parties participating in the protocol.
- `d`: It represents the overall description of the circuit: the total number of wires in the circuit, the id of the first non-input gate, and the number of xor gates in the circuit.
- `i`: It describes the input part of a party in the circuit: party id, the wire id of the first input of the party, and of the last.
- `g`: It represents the gate. The id of each gate is the id of the output wire of the gate. The values to be specified are the gate id, the gate type, the left-input wire id, the right-input wire id, the number of parents (i.e., gates that take the output of this gate as input). The gate type can be either input (16), AND (1), XOR (2). If the gate is also an output gate, OUTPUT (32) is added to the original type. If the number of parents of the given gate is non-zero, the ids of the parents are listed. Note that the first two gates are reserved for constant 0 and 1; the circuit file should follow this rule.

Converting the text file into a binary one. To improve performance by avoiding parsing overhead, our MPC implementation recognizes only the binary file. Conversion can be done as follows:

```
>mpc.exe -c circ.txt circ.bin
```

2.2 Default circuit

The following default circuit is embedded in the source code. If the configuration doesn't specify the circuit file, the default circuit will be used.

Circuit description. The circuit considers the case where a single customer P_{n+1} interacts with a set of providers $\mathbf{P} = \{P_1, P_2, \ldots, P_n\}$. Each provider P_i holds (private) input x_i, and the customer holds (private) input x_{n+1}. The customer will compute $f(x_1, \ldots, x_n, x_{n+1})$. The input of each provider P_i is a vector of resource values $x_i = \{v_r\}_{r \in R_i}$, where $R_i = [(i-1)k+1, ik]$ (k is specified as `num_input` in the configuration file). The input of the customer is the vector $x_{n+1} = (b_1, \ldots, b_{nk})$ where $b_i \in \{0,1\}$ for $i \in [1, nk]$. Let $i_r = v_r b_r$. The output of the circuit is a resource r maximizing i_r.

Saving the default circuit into files. The default circuit can be save as a text file:

```
>mpc.exe config.txt -t circ.txt
```

Also, one can get a binary circuit file.

```
>mpc.exe config.txt -b circ.bin
```

2.3 Modifying the source code

If the size of a circuit becomes more than 10^6, loading the circuit from a file takes much longer than creating it directly within the program. To achieve higher performance, one can also customize the circuit with a source-code level with minimal modification; the MPC engine in our implementation refers to circuits via an abstract C++ interface.