Jonathan Bronshtein 208551374

Introduction:

When one thinks of computing a function it may seem a necessary thing for the input to be known, it is, however, not the case as has been shown by [4], [5], [6],[7] and others, which deal with the problem of secure computation, i.e., of players with inputs computing without any two players revealing their input to each other. Of these GMW and Yao’s Protocol are covered by the first and last overviews which are based off of [1],[3]. The second overview is given about Cleve’s article about the impossibility of two processors agreeing on a single bit when one of them can abort at any moment. I believe that this result is likewise surprising and somewhat disappointing as well, as it implies that stronger tools must be used for agreement on random numbers.

\* where is no. of rounds and

Overview of the GMW protocol [1] and its proof:

We have players who want to compute a function that is given in a form of a circuit consisting of and gates (any function can be represented using these gates) while the players’ input bits must remain secret. To share his input bit to the wire every player divides it into shares: and gives share . The shares for are bits chosen uniformly at random, whereas ( here denotes or addition modulo 2) it can clearly be seen that .

The gate with input wires and output wire is computed by each party by taking its shares of the wires and adding them privately, so each party has .

The computation of the gate (with input wires and output wire is more complicated. The computation of the AND gate is as follows:

To compute a share of the sum can compute by himself, but to compute a share of he requires help from player . Note that there are 4 options for for bits, , which are corresponding to . adds to each option a bit chosen uniformly at random and sends those options using 1 out of 4 oblivious transfer, then chooses the option corresponding to his bits, denote it by . Thanks to oblivious transfer doesn’t know bits and thanks to doesn’t know bits. Notice that .

Correctness:

XOR:

AND: Recall that party holds , andand that By combining their shares the players will get:

As required.

The correctness of the circuit can be proven using induction. Assume the wires are labeled in the order of the computation of their value, i.e., for a gate with input wires and output wire .

Basis: in the start of the algorithm the input for every wire is distributed into shares for which it holds that as shown above. Assume this holds for wire . Then we have by the proofs of correctness for the gates this holds for wire

Security:

receives from other players and are bits chosen uniformly at random, also does not reveal information because is the addition of bits chosen uniformly at random, so

For . Suppose this holds for . Then when adding another bit, we have by the induction hypothesis , so for :

Therefore cannot learn anything of from the shares he gets.

OT security: due to , so cannot learn anything from it. Due to OT can’t learn choice, so he doesn’t learn anything either.

Gate security: If it were possible to learn anything about the gate’s output it would allow one to learn about its input as well, which would be a contradiction to what is stated above.

By a similar argument of induction as that in the correctness part it can be shown that throughout the computation the shares leak no information about the secret.

Overview of “Limits on the Security of Coin Flips When Half the Processors are Faulty[2]”

We view the processors as a sequence of circuits where is the number of rounds and the last circuit outputs the bit selected by the processor. Define a 2-processor scheme to be if and define the bias of towards to be .

Theorem: If then there exists a faulty (or malicious) processor that can cause the bias of the other processor to be at least . We say that a processor quits at round if, from round onwards, all the messages that the processor sends are strings of zeroes. For , let be the default output of if quits at round and let be the output of if does not quit during the protocol. Define faulty processors here quits at round 1 and works as follows ( makes the first move in every round):

1. Run for rounds
2. Compute
3. If

simulate for round and then quit at round .

else:

quit at round

works similarly.

For the bias of (i.e. when the game is played by processors towards 0 is

For the bias of towards 0 is

Because quits at round if and it quits in round if .

Similarly, for the bias of towards 1 is

For the bias of towards 0 is:

Remember that makes the first move, that is why if quits in the round will respond in the round.

Finally, for the bias of towards 1 is:

Let be the average of the biases, then:

We get that because after summation:

for

1. due to consistency (as defined above) and because .

Because δ one of the biases must be

Overview of Yao’s Protocol for Two-Party Computation[3]

The purpose is to compute a function , where the inputs of players and are and , without the players revealing each other’s input.

The following is a description of the protocol. A circuit which computes is constructed. Let a gate in the circuit, has 2 input wires and and an output wire which can take on values 0 or 1. For every wire two values and are specified which represent the bits 0 and 1 respectively, are identically distributed, so they are indistinguishable. Garbling a gate is done by encrypting the input wires as described and the output wire is encrypted by the keys of the input in addition to being given keys for its bits, e.g., for an AND gate where encrypts with the key . Had this not been done it would have been possible to decipher the gate by inputting different values into it and seeing that, for example an AND gate gives 3 times and only once. Note that the gate is provided as a table and its entries are ordered randomly. A garbled circuit consists of such garbled gates, to generate one from a regular circuit we first generate the gates whose input is also the input to the circuit, we get their possible outputs and continue to the gates whose garbled inputs we have until we have garbled the entire circuit (this can be done by arranging the wires by the order of their computation). The circuit has a decryption table in which for every output wire we have Thus we can compute the entire circuit without knowing the input or any of the values of the intermediate wires. After generating the circuit sends it to . Next requires the input keys, can simply send his own keys, but he must not know the input of the receiver, so to get his key uses 1 out of 2 oblivious transfer, i.e., OT receives the input where is the bit wants to get and and are the encryptions of the bits 0 and 1. The result is that doesn’t know and gets without knowing . This concludes the description of the protocol.

Bibliography

[1] Seung Geol Choi, Kyung-Wook Hwang, Jonathan Katz, Tal Malkin, Dan Rubenstein. Secure Multi-Party Computation of Boolean Circuits with Applications to Privacy in On-Line Marketplaces, page 4.

[2] Richard Cleve. Limits on the Security of Coin Flips When Half the Processors are Faulty, Department of Computer Science University of Toronto Toronto, Canada, 1986.

[3] Yehuda Lindell and Benny Pinkas. A Proof of Security of Yao’s Protocol for Two-Party Computation, 2006.

[4] Oded Goldreich, Silvio Micali, Avi Wigderson. How to Play any Mental Game – A Completeness Theorem for Protocols with Honest Majority. In 19th STOC, pages 218–229, 1987.

[5] Donald Beaver, Silvio Micali, Phillip Rogaway. The Round Complexity of Secure Protocols, MIT Ph.D. Thesis, June 1991.

[6] Andrew Yao. How to Generate and Exchange Secrets. In 27th FOCS, pages 162–167, 1986.

[7] Michael Ben-Or, Shafi Goldwasser, Avi Wigdemon. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation, 20th Annual

ACM Symposium on Theory of Computing, pp. 1–10, 1988.