650.030 Introduction to Cybersecurity Summer Term 2022/23

Multi-party computation

Karol Wesołowski 12206888

Introduction

The goal of this assignment was to design and describe a simple multiparty computational scheme, that will be able to provide one type of calculation on integer numbers, and verify the results. In the first section, I will present the idea behind the multi-party computation and its most popular implementations. Then I will provide a quick overview of my implementation. Finally, I will talk about the possible use cases and social impact of these types of algorithms.

Multi-party Computation overview

The main idea behind secure multi-party computation is to allow two (or more) parties (people, organizations, companies) to gain some knowledge about the data each of them possesses, without reviling the data itself (Goldreich, 1998, p. 4). This is a great achievement in cryptography and computer science in general, since it allows many parties, that don't have to trust, or even know each other, to produce some meaningful and important information based on the data only they possess (Ben-David et al., 2008, p. 257).
To implement that idea, we use so-called protocols – synchronous models of communication between parties (Goldreich, 1998, p. 10). For this project, I will take into consideration only two-parties protocols, which constitute a subproblem of the broader multi-party computation.
There exists a lot of proposed implementations of this problem. Many researchers seek to improve works of other to generate a protocol that is faster (Wang et al., 2017), more efficient (Henecka et al., 2010) or user-friendly programs (Ben-David et al., 2008).
But those implementations are based on well acclaimed protocols (Ben-David et al., 2008, p. 261). To such we can amount protocols based on secure function notions of (among others): constant-round complexity security (Beaver et al., 1990), polynomial-time family of circuits (Goldreich et al., 1987), and the most popular - Yao garbled circuit (although not called that way in the paper, Yao, 1986).
Yao protocol or Yao garbled circuit is a secure communication protocol between two parties, that does not require existence of a third party (in contrast to for example Blum, 1983 (p.26)). Protocol is based on the existence of a one-way function (i.e. such that cannot be easily reversed), that can be generalized as (or described by) a digital circuit. Said circuit is then garbled (or randomized) and sent, alongside the output generated by one of the parties, to the other (in contrary to Beaver et al., 1990 (p. 509), where all parties cooperate in garbling). The second party pushes his own input through the circuit and thanks to the oblivious transfer (for example based on the description by Even et al., 1985) obtains a result of the function without knowing the real value, that the first party has sent.

Implementation

My implementation of the MPC problem uses Yao protocol. The task of my program is to calculate the maximal value of two set of numbers (each of the number is of four bits of length, meaning they are smaller than 16), held by two separate parties. Parties are named Alice and Bob, as proposed by Rivest et al., 1978 (p. 122).
The implementation is based on a python package developed by Roques & Risson, 2020. My version serves as an extension to the original code, provides the data input features, and allows verification.

Algorithm

Each party uses a separate terminal, to allow them to have private input data. Data may be imputed via console (by end user) or read from file. Execution of the program consist of multiple steps:

1. Read input data (alongside execution flags and circuit to perform operation; default settings allow proper execution of the program without the need of specifying anything in addition).
2. Calculate “true” input - maximal value of the private set of data.
3. Prepare circuit: produce probability bits and garble the table representing each gate, to assure secrecy. (This is done on Alice side)
4. Open communication in order to test the circuit and print the output for all possible inputs (i.e. push all combinations of two 4-bit integers through the circuit. This is done only when a proper flag is set; on default it is performed)
5. Alice opens communication and sends “true” input through the circuit. Bob responds with sending his input into circuit. Both parties receive the output - maximal value of both of their sets.
6. Verification

Most of the parts, including definition of Yao protocol, implementation of communication protocols and Oblivious Transfer are done by Roques & Risson, 2020. My task was to add input reading and specification for a given function – the maximal value across two sets.

Circuit

To achieve the calculation, Yao protocol requires a circuit, constructed from logic gates. To create a circuit that will produce as an output maximum of two values, I used an idea of full subtractor.
Subtractor is a simple logic circuit based on the implementation of adder. It takes three bits as an input: *A* and *B*, which are data bits, and a so called *Borrow-bit* S_{in} which represents the fact, that a previous operation “borrowed” from current values, and returns two bits of output: result of the operation (usually denoted as *D*) and *Borrow-bit* S_{out}.
In my implementation, the bit *D* is omitted, as it doesn’t serve a useful purpose. We are only interested in the final S_{out} bit, which holds the information, whether the number B^* is bigger than A^*.

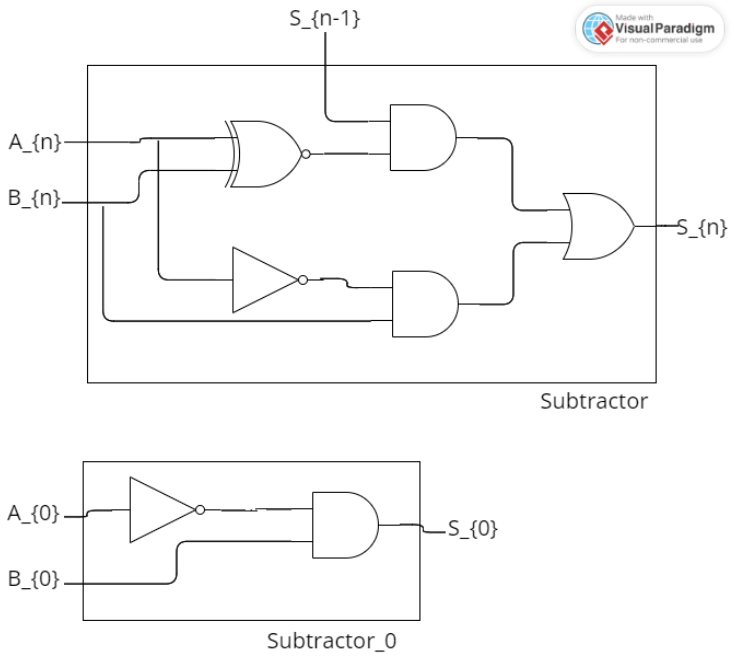

Figure 1a. Subtractor

Figure 1b. Simplified first subtractor

Each subtractor consist of five gates, connected in the following manner. Due to the fact, that *Borrow-bit* on the first subcircuit is always set to 0, it is possible to remove redundant gates and simplify the circuit as shown on the fig. 1b.

Subtractors are connected in chain, through borrow bits. Each subcircuit is fed with respective bits of input, starting from the least significant bit.

The output from the last subtractor is then passed to the set of multiplexers, each connected with respective bits of input. If the steering bit *S* is equal to 0, then on the output will appear the bit from input *A*, otherwise, the bit from input *B*.

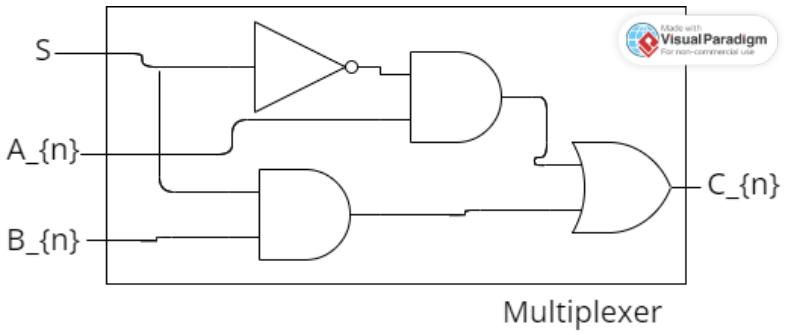

Figure 2. Multiplexer

Bits C^* create the full output.

Although for now the implementation accepts only four bits of input from each side, it is very easy to extend the size of input up to 16 bits, without any changes in code. The only part that would require a change is the description of the circuit, but due to its repeatable nature, it shouldn't require a lot of work.

Verification

The verification part, although necessary from the educational point of view, neglects the idea of the secret knowledge, since it requires both parties to share their "true" input. In my implementation it is Alice, who sends her value to Bob, and he performs the calculation once

more, this time on plain data. If the result of the calculation is the same as the one obtained through the use of MPC protocol, the value *True* is sent back to Alice.
The implementation utilizes the idea, that max function is homogenous. That means, that max performed on the whole set is equal to max calculated over respective *max*'s of subsets. To put it in mathematical terms:
$max(A \cup B) = max(max(A), max(B))$

Use cases and social impact

Secure multi-party computation protocols may be used in many different real life scenarios. Some of them were described by the creators of the protocols (or security notions). The protocol may be used to define which of the parties is the wealthiest (or in general, owns the biggest amount of something), without revealing the actual numbers (Yao, 1982, p. 160), to create a fair and foolproof coin flipping mechanism (or, in general, community based pseudo random number generator, that ensures, that both sides has exactly the same number) (Blum, 1983, p. 23), or to hold an online election (or any other voting) with the certainty, that the results will be correct, and no personal data will be revealed (Naidu et al., 2016, p. 3).
The secrecy of data transferred via some communication protocol is vital. There is a lot of personal information, generated by every user of a protocol (for example HTTPS protocol on the Internet), that is crucial to be kept secret. Among such may be names and addresses, which by itself are not very useful to malign party, or some more sensitive data, like passwords and credit card details. The secure communication protocols come then with a great help, for example during censuses, where it is important to obtain statistical data, without the necessity to store or process personal information (i.e. it is crucial to provide anonymity).
It is possible to find real-life use cases for my implementation. We can assume such situation: There are two noble families, each of great wealth. The heads of the houses compete with each other, and want to know, what is the most valuable possession in their collections. But they don't want to share the whole value of the collection, nor it's size and want to keep the information of their crown jewel secret as long as possible (because they don't want to risk the humiliation, when the other side sees the differences in value). So they share the value of the most valuable artifact to each other, via protocol, that will not inform the other side of the real value of the object, as long as it is not the most valuable across all the collections.
That was more of an anecdotal situation, but it is not that hard to find more down-to-earth examples. We can assume the scenario, in which there is a central exam, held among various schools. There is no fixed grading scheme, instead we assume, that it will be calculated afterward, based on the highest score among all students. Thus, we can implement this algorithm, to calculate this highest score, without revealing any other statistical data.
It can be also used in a situation, where we are interested in only *max*, instead of any other statistics of the sets, for example, what is the age of the oldest player among two (or more) teams.
In the first described situation, my implementation may not be very practical in two-users scenario. Since it returns the highest value, the parties will know, to whom that value belongs. But on the other hand, the rest of the data they possess will be kept to themselves. In the situation, where more users participate in the protocol (which modern implementations

of Yao protocol allow), even the bearer of the highest value will be kept secret to all (so we obtain the situation closer to the one described by Yao in his original paper (Yao, 1982), where other users, except for learning the highest value, can only check if their value is lower than obtained).
The implementation, although generally secure, is not foolproof. The main reason for that is the main idea of the protocol: secrecy of partial data. Even though the parties don't know the value, that the other side will provide, they may try to disturb the flow of information, by sending a value, that is not real (they don't have it in their input), and has a high chance of being the output of the protocol (for example sending a number that is close to the upper limit of the variable size is almost certain to be the result of max function, which was pointed out in Yao, 1986, p.196).
This means that for the secrecy, we are paying with the lack of ability to control and verify the realism of output (which was also shown in my implementation - verification required revealing data). But even if parties reveal the maximal value of their data, there is still a possibility, that it wasn't obtained from a reliable source (i.e. it was produced by the party, not obtained from input, pointed out in Goldreich et al., 1987, p. 226). Only by having a transparent and public way of collecting data, we can be certain, that no information was falsified.

Summary

As shown in this paper, multi-party computation protocols are very important for the future of the cryptography and computer science in general, since they allow to securely transfer data through internet and so allow the creation of distributed systems. They make possible secure exchange of information, required in many modern problems (Du & Atallah, 2001). Although they provide secrecy of data and security to a great extent, they are not foolproof, and are sensitive to actions of dishonest parties. My implementation solves a simple statistical task and provides a verification mechanism, to show that it is indeed working as intended.

References

Beaver, D., Micali, S., & Rogaway, P. (1990). The round complexity of secure protocols. In *Proceedings of the twenty-second annual ACM symposium on Theory of Computing (STOC '90)* (pp. 503–513). ssociation for Computing Machinery. https://doi.org/10.1145/100216.100287

Ben-David, A., Nisan, N., & Pinkas, B. (2008). FairplayMP: a system for secure multi-party computation. In *Proceedings of the 15th ACM conference on Computer and communications security (CCS '08)* (pp. 257–266). Association for Computing Machinery. https://doi.org/10.1145/1455770.1455804

Blum, M. (1983, Winter-Spring 1983). Coin flipping by telephone a protocol for solving impossible problems. *SIGACT News*, *15*(1), 23–27. https://doi.org/10.1145/1008908.1008911

Du, W., & Atallah, M. J. (2001). Secure multi-party computation problems and their applications: a review and open problems. In *Proceedings of the 2001 workshop on New security paradigms (NSPW '01)* (pp. 13-22). Association for Computing Machinery. https://doi.org/10.1145/508171.508174

Even, S., Goldreich, O., & Lempel, A. (1985). A Randomized Protocol for Signing Contracts. *Commun. ACM*, *28*, 637-647. 10.1007/978-1-4757-0602-4_19

Goldreich, O. (1998). Secure multi-party computation. *Manuscript*. dce0d462c182121f37279e3809d484624f3d3eba

Goldreich, O., Micali, S., & Wigderson, A. (1987). How to play ANY mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing (STOC '87)* (pp. 218–229). Association for Computing Machinery. https://doi.org/10.1145/28395.28420

Henecka, W., Kögl, S., Sadeghi, A.-R., Schneider, T., & Wehrenberg, I. (2010). TASTY: tool for automating secure two-party computations. In *Proceedings of the 17th ACM conference on Computer and communications security (CCS '10)* (pp. 451-462). Association for Computing Machinery. https://doi.org/10.1145/1866307.1866358

Naidu, P. S., Kharat, R., Tekade, R., Mendhe, P., & Magade, V. (2016). E-voting system using visual cryptography & secure multi-party computation. *2016 International Conference on Computing Communication Control and automation (ICCUBEA)*, 1-4. oi: 10.1109/ICCUBEA.2016.7860062

Rivest, R. L., Shamir, A., & Adleman, L. M. (1978, February 01). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, *21*(2), 120–126. https://doi.org/10.1145/359340.359342

Roques, O., & Risson, E. (2020, November 25). *ojroques/garbled-circuit: A two-party secure function evaluation using Yao's garbled circuit protocol*. GitHub. Retrieved April 29, 2023, from https://github.com/ojroques/garbled-circuit

Wang, X., Malozemoff, A. J., & Katz, J. (2017). Faster Secure Two-Party Computation in the Single-Execution Setting. In J. B. Nielsen & J.-S. Coron (Eds.), *Advances in Cryptology – EUROCRYPT 2017: 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 – May 4, 2017, Proceedings, Part III* (Vol. 10212, pp. 399–424). Springer International Publishing. https://doi.org/10.1007/978-3-319-56617-7_14

Yao, A. C.-C. (1982). Protocols for secure computations. *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (SFCS '82)*, 160–164. 10.1109/SFCS.1982.38

Yao, A. C.-C. (1986, October). How to generate and exchange secrets. *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, *1986*, 162-167. 10.1109/SFCS.1986.25