Homework Assignments for MPC Class

Dr. Adi Akavia, CS Department, University of Haifa

Abstract

This file contains the first homework assignments for the MPC course.
The timeline may be modified during the semester.

Assignments Overview and Background

In these homework assignments you will implement a secure computation protocol for the sign-activation function, using a variety of techniques to be learned throughout the course.

Where and when to submit. See a submission box for each assignment, together with the corresponding due date, in the Moodle course site.

Sign activation. The *sign activation function* $f_{\vec{a},b}\colon \mathbb{R}^d \to \{0,1\}$ is parameterized by a weights vectors $\vec{a} = (a_1, \dots, a_d) \in \mathbb{R}^d$ and a threshold $b \in \mathbb{R}$ and defined by

$$f_{\vec{a},b}(\vec{x}) = \begin{cases} 1 & \text{if } \sum_{i=1}^{d} a_i x_i \geq b \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Secure computation of the sign function. In the first assignments, for simplicity, we focus on sign activation functions of dimensions $d = 1$ and $d = 2$, with a fixed threshold set to be $b = 4$, and where the weights a_i and inputs x_i are restrict to accept values in $\{0,1,2,3\}$ (rather than $\mathbb{R}$). Namely, we focus on the following two functions. For $d = 1$, the function is $f_{a,4}\colon \{0,1,2,3\} \to \{0,1\}$ defined by

$$f_{a,4}(x) = \begin{cases} 1 & \text{if } ax \geq 4 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

For $d = 2$ the function is $f_{\vec{a},4}\colon \{0,1,2,3\}^2 \to \{0,1\}$ defined by

$$f_{\vec{a},4}(x_1, x_2) = \begin{cases} 1 & \text{if } a_1 x_1 + a_2 x_2 \geq 4 \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Background: Secure computation. Secure computation addresses situations where several parties wish to compute a function on the union of their private inputs, but without revealing any information on the inputs (other than what can be inferred from the output of the function). The computation to be securely computed is typically modeled by an arithmetic circuit or a Boolean circuit (see below). Other models of computations are also considered, such as Turing Machines and RAM computations. In this course we focus on the former, as they are simpler to address.

Background: Arithmetic circuits. Arithmetic circuits are a model of computation, where the inputs are variables $x_1, \dots, x_n$, and the computation is performed using the arithmetic operations of addition ($+$) and multiplication ($\times$) in a field $\mathbb{F}$ (possibly involving also constants from the field hard-wired in the circuits). The output of an arithmetic circuit is thus a polynomial (or a set of polynomials) in the input variables. The complexity measures associated with such circuits are *size* and *depth* that capture the number of operations (size) and the maximal distance between an input and an output (depth). Other important complexity measures are *number of multiplication* ($\#MULT$) that capture the number of multiplication operations, and the *multiplicative depth* ($\times$*-depth*) that captures the maximal distance between an input and an output when counting only multiplication gates while ignoring addition gates.

Formally, an arithmetic circuit $\mathcal{C}$ is a field $\mathbb{F}$ and variables $X = \{x_1, \dots, x_n\}$ is a directed acyclic graph satisfying the following. The vertices of $\mathcal{C}$ are called *gates*. Every gate in $\mathcal{C}$ of in-degree 0 is labeled by either a variable from X or a field element from $\mathbb{F}$. Every other gate in $\mathcal{C}$ is labeled by either $\times$ or $+$ and has in-degree 2. An arithmetic circuit is called a *formula* if every gate has out-degree 1 (so that the underlying graph is a directed tree). Every gate of in-degree 0 is called an *input gate* (even when the gate is labeled by a field element). Every gate of out-degree 0 is called an *output gate*. Every gate labeled by $\times$ is called a *product gate* (also called, multiplication gate), and every gate labeled by $+$ is called a *sum gate* (also called, addition gate). The *size* of $\mathcal{C}$, denoted $|\mathcal{C}|$, is the number of edges in $\mathcal{C}$. The *depth* of a gate v in $\mathcal{C}$, denoted $depth(v)$, is the length of the longest directed path reaching v. The *depth* of $\mathcal{C}$ is the maximal depth of a gate in $\mathcal{C}$. The edges of $\mathcal{C}$ are often called *wires*. The *fan-in* (respectively, *fan-out*) of a gate is its number of incoming wires (respectively, outgoing wires). Typically we address addition and multiplication gates with fan-in two and fan-out 1; sometimes we consider circuits with unbounded fan-in. For two gates u and v in $\mathcal{C}$, if (u, v) is an edge in $\mathcal{C}$, then u is called a *child* of v, and v is called a *parent* of u.

An arithmetic circuit computes a polynomial in a natural way: an input gate labeled by $\alpha \in \mathbb{F} \cup X$ computes the polynomial α. A product gate computes the product of the polynomials computed by its children. A sum gate computes the sum of the polynomials computed by its children.

For a gate v in $\mathcal{C}$, define $\mathcal{C}_v$ to be the sub-circuit of $\mathcal{C}$ rooted at v. Denote by X_v the set of variables that occur in the circuit $\mathcal{C}_v$. We usually denote by f_v the polynomial in $\mathbb{F}[X_v]$ computed by the gate v in $\mathcal{C}$. We sometimes abuse notation and denote by $\mathcal{C}_v$ the polynomial computed by v as well. Define the *degree* of a gate v, denoted $deg(v)$, to be the total degree of the polynomial f_v (*e.g.*, the total degree of $x_1^2 x_2 + x_1 + 1$ is three, whereas the individual degrees are at most two). The *degree* of $\mathcal{C}$ is the maximal degree of a gate in

$\mathcal{C}$.

Every polynomial $f \in \mathbb{F}[X]$ can be computed by an arithmetic circuit and by an arithmetic formula. The main question is what's the complexity of the circuit.

Fields $\mathbb{F}$ commonly considered are $GF(p)$, which is the field whose elements are the equivalent classes of modulo p computation, often represented by the integers $\{0, 1, \ldots, p-1\}$, and with operations of addition and multiplication modulo p, denoted $+_p$ (ADD_P), $\times_p$ ($MULT_p$). When p is clear from context we typically omit it and use the notation $+, \times$ or $ADD, MULT$. Other fields of interest are the complex numbers $\mathbb{C}$, real numbers $\mathbb{R}$, rationals $\mathbb{Q}$, and integers $\mathbb{Z}$.

In summary, an arithmetic circuit is a model of computation where the basic operations are addition and multiplication in a field $\mathbb{F}$, the circuit corresponds to a polynomial, the inputs are variables and constants, and the output is the evaluation of the polynomial on an assignment to the input variables.

Background: Boolean circuits. Boolean circuits are arithmetic circuits where the computation is in the field $GF(2)$ (also denoted $\mathbb{Z}_2$), where addition and multiplication are modulo 2. The gates of Boolean circuits are typically labeled by XOR and AND (since XOR is equivalent to summation, and AND is equivalent to a product modulo 2).

Assignment 1: Truth-table and circuits for sign activation

What to do.

1. Write the truth table for the function specified in Equation 2. The truth table specifies the function values for all possible assignments to $\vec{a}, \vec{x} \in 0, 1, 2, 3$.

2. Write a Boolean circuit computing the function specified in Equation 3, with XOR and AND gates. Here $\vec{a} = (a_1, a_2)$ and $\vec{x} = (x_1, x_2)$ are from $0, 1, 2, 3^2$ where each entry is specified in binary representation. Namely, $a_1 = (a_{11}, a_{10}) \in \{0, 1\}^2$, $a_2 = (a_{21}, a_{20}) \in \{0, 1\}^2$, $x_1 = (x_{11}, x_{10}) \in \{0, 1\}^2$, $x_2 = (x_{21}, x_{20}) \in \{0, 1\}^2$.

3. Write an arithmetic circuit computing the function specified in Equation 3, with addition and multiplication gates in the field $GF(11)$. That is, the gates correspond to the + mod 11" and "$\times$ mod 11" operations; and the inputs and outputs are represented as elements in $GF(11)$ (equivalently, integers in $0, 1, \ldots, 10$). Recall here again that we restrict a_1, a_2, x_1, x_2 to $\{0, 1, 2, 3\}$.

4. Analyze the complexity of your Boolean and arithmetic circuits in terms of all the following properties: (i) circuit depth, (ii) circuit size, (iii) $\times$-depth, (iv) $\#MULT$.

   Hint 1.

   Hint 2.

5. **(extra credit)** Generalized your solution for 2-4 to the case where:
   (i) The dimension is $d = 3$;
   (ii) The entries a_i, x_i are in $\{0, 1, 2, \ldots, 2^\ell - 1\}$ for $\ell = 3$;
   (iii) d, ℓ are parameters.

6. **(extra credit)** Can you improve the complexity of your solution? Suppose you may compute *approximate* sign function, that errs on some of its inputs. Can you give a lower complexity solution to the approximate sign function? Would your solution change if you consider different arithmetic, as in, Boolean vs. integer modulo a prime vs. floating point arithmetic on real number? If so, what would be your solutions in each setting? What's your trade-off between complexity and precision?

What to submit. Submit a full description of each circuit, both verbally (written in Latex) and pictorially. The verbal description must be written in Latex. The circuits drawing may be drawn using any drawing tool of your choice (for example, PowerPoint, or even with **clear** handwriting). Submit a single zip file names ps1-¡your name¿-¡your id¿ consisting of the following files: (1) drawings (e-copy), (2) latex source files, (3) a pdf file of the compiled latex text and drawings.

Most relevant class material. Lecture 1.

Assignment 2: One-Time Truth-Table (OTTT), passive

What to do. Implement a secure two-party protocol for computing the function specified in Equation 2 using the passively secure One-Time Truth-Table (OTTT) protocol. You should describe and implement both the dealer (offline) phase and the protocol online phase. Submit the report and files as specified in the Homework Guidelines.

In more details, the parties are Alice and Bob. At the onset of the protocol Alice has input $x \in \{0, 1, 2, 3\}$, and Bob has input $a \in \{0, 1, 2, 3\}$. Upon the termination of the protocol Alice should have obtained the value $f_{a,4}(x)$, Bob has no output. Security is required only against passive adversaries (also called semi-honest or honest-but-curious). The secure computation technique you should use is One-Time Truth-Table (OTTT). Recall you have written the truth-table for the function in Assignment 1, part 1.

Implementation notes: Since the goal of the exercise is to better understand the protocol (not to build a full functioning system), feel free to implement all parties on the same machine and without using network communication. For example, you could implement the dealer, Alice and Bob as three distinct classes in Python and then let them interact in the following way:

```
Dealer.Init();
Alice.Init(x, Dealer.RandA());
Bob.Init(y, Dealer.RandB());
Bob.Receive(Alice.Send());
Alice.Receive(Bob.Send());
z = Alice.Output();
```

What to submit. Submit the report and files as specified in the Homework Guidelines.

Most relevant class material. Lecture 3.

Assignment 3: BeDOZa protocol, passive

What to do. Implement the passively secure BeDOZa protocol for computing the function specified in Equation 3, where Alice holds $\vec{a}$ and Bob holds $\vec{x}$. You may implement the computation as either a Boolean or an arithmetic circuit, as you see fit (clearly state your choice in the report). You should describe and implement both the dealer (offline) phase and the protocol online phase. Submit the report and files as specified in the Homework Guidelines.

Since the goal of the exercise is to better understand the protocol (not to build a full functioning system), feel free to implement all parties on the same machine and without using network communication. For example, you could implement the dealer, Alice and Bob as three distinct classes in Python and then let them interact in the following way:

```
Dealer.Init();
Alice.Init(x, Dealer.RandA());
Bob.Init(y, Dealer.RandB());
while(Alice.hasOutput()==false)
{
    Bob.Receive(Alice.Send());
    Alice.Receive(Bob.Send());
}
z = Alice.Output();
```

Remark: the loop is not strictly necessary, since you are implementing the protocol for a specific function so you can calculate in advance how many rounds there will be.

What to submit. Submit the report and files as specified in the Homework Guidelines.

Most relevant class material. Lecture 4.

Assignment 4: BeDOZa protocol, active (malicious) adversary

What to do. Enhance your implementation of BeDOZa protocol (assignments 3) to achieve security against *malicious* adversaries (security with abort). Use MACs to authenticate secret shares, as shown in class.

What to submit. Submit the report and files as specified in the Homework Guidelines.

Most relevant class material. Lecture 7.

Assignment 5 (extra credit, not mandatory)

What to do. Can you extend your implementation of BeDOZa protocol (assignments 3-4) to general functionality? That is, the input to the protocol includes:

- a function specified by an arithmetic circuit
- Alice's input
- Bob's input.

Assignment 6: Oblivious Transfer to Replace Dealer

What to do. Implement a 1-out-of-2 OT protocol against passive adversary. Next, use your OT to eliminate the need for the dealer in BeDOZa protocol you implemented in Assignment . You should describe and implement both the dealer (offline) phase and the protocol online phase. Submit the report and files as specified in the Homework Guidelines.

More details on the OT implementation follow. Use ElGamal encryption as the underlying PKE scheme. Since the goal of the exercise is to better understand the protocol (not to build a full functioning system), feel free to implement all parties on the same machine and without using network communication.

What to submit. Submit the report and files as specified in the Homework Guidelines.

Most relevant class material. Lecture 8.

Assignment 7: Garbled Circuits

What to do. Implement Yao's protocol for the Boolean circuit evaluating Equation 2. You already have implemented OT in assignment 6. You can implement the PRF using SHA-256, and set the length of the wire labels to 128 bits. You may use the hashlib library in python https://docs.python.org/2/library/hashlib.

Since the goal of the exercise is to better understand the protocol (not to build a full functioning system), feel free to implement all parties on the same machine and without using network communication.

What to submit. Submit the report and files as specified in the Homework Guidelines.

Most relevant class material. Lecture 10.

Assignment 8: Homormorphic Encryption

What to do. In this assignment there is no programming; you are only required to write and submit in LaTex the details of a dry-run of homomorphic evaluation (i.e. running a program "on paper", rather than in a computer). Use as the homomorphic encryption the scheme over the integers that was presented in class.

The cleartext computation is as follows:

> Given tree bits $a, b, c \in \{0, 1\}$, output $AND(XOR(a, b), c)$.

The homorophic computation is as follows:

> Given tree bits $a, b, c \in \{0, 1\}$ and a security parameter λ, do:
>
> 1. Generate keys $sk \leftarrow \mathsf{Gen}(1^\lambda)$,
> 2. Encrypt the input $\mathsf{c}_a \leftarrow \mathsf{Enc}_{sk}(a)$, $\mathsf{c}_b \leftarrow \mathsf{Enc}_{sk}(b)$, $\mathsf{c}_c \leftarrow \mathsf{Enc}_{sk}(c)$,
> 3. Homomorphically evaluate the aforementioned cleartext computation to obtain a result ciphertext c_{res},
> 4. Decrypt to obtain a cleartext result $res \leftarrow \mathsf{Dec}_{sk}(\mathsf{c}_{res})$,
> 5. Verify that the result is correct, i.e., check that $res == AND(XOR(a, b), c)$, if yes out res else output Error.

Your task: choose input bits a, b, c and a security parameter λ, write on paper detailed execution of the above homomorphic computation. That is, for every step 1-5 write its input, the details of the computation and the resulting outcome of the step. Note: your solution should be (a) correct, (b) clear, easy to follows, and with sufficiently many details to make checking your work as easy as possible.

What to submit. Submit the solution (pdf + tex source files).

Most relevant class material. Lecture 11.