Lecture Notes for Cryptographic Computing
2. Trusted Dealer Model, Passive Security

Lecturers: Claudio Orlandi, Peter Scholl, Aarhus University

October 23, 2023

1 Before you start

Before reading this note, it is recommended to read the formal definition of security against semi-honest adversaries (or passive security) from [HL10, Sections 2.1-2.2].

Here are the references for the material covered in this note:

1. Examples of reductions between different functionalities (from [HL10, Sections 2.5.1-2.5.2]).
2. *One Time Truth Tables*: see also [IKMOP13, Section 3.2].
3. The BeDOZa protocol: see [BDOZ11, Section 3.1] (for the arithmetic case) and [NNOB12, Section 3] (for the Boolean case). In this note we only focus on the (simplified) version for passive security.
4. Most of the topics in this note are also covered in this video[1].

2 One-Time Truth Table – Passive Security

We start with (probably) the simplest protocol for secure two-party computation in the presence of a trusted dealer D. The protocol allows two parties to compute any function of their inputs. In more details we are going to compute a function

$$f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$$

(In this note we are going to treat $\{0,1\}^n$ and $[0,\ldots,2^n-1]$ interchangeably. Sometimes it will be useful to remember that the input is a string of bits, so that each bit can be referenced individually, while other times it will be useful to be able to add/subtract the inputs modulo 2^n). The function is represented by a truth table (i.e., a matrix) $T \in \{0,1\}^{2^n \times 2^n}$ where $T[i,j] = f(i,j)$ (that is, one of the inputs selects a row in the matrix, and the other selects a column in the matrix). Our goal (or the "ideal functionality") is the following: Alice inputs $x \in \{0,1\}^n$, Bob inputs $y \in \{0,1\}^n$. At the end of the protocol Alice learns $z = f(x,y)$ while Bob learns nothing. Here is how the protocol works:

The dealer: The dealer D performs the following operations:

1. Choose two shifts $r \in \{0,1\}^n$ and $s \in \{0,1\}^n$ uniformly at random;
2. Choose a matrix $M_B \in \{0,1\}^{2^n \times 2^n}$ uniformly at random;
3. Compute a matrix M_A such that

$$M_A[i,j] = M_B[i,j] \oplus T[i - r \bmod 2^n, j - s \bmod 2^n]$$

4. Output (r, M_A) to Alice and (s, M_B) to Bob;

[1] https://www.youtube.com/watch?v=jJ5d-EUq-DY

The protocol: Having received (r, M_A) and (s, M_B) from D, A and B with input x and y:

1. Alice computes $u = x + r \bmod 2^n$ and sends it to Bob;
2. Bob computes $v = y + s \bmod 2^n$ and $z_B = M_B[u, v]$ and sends (v, z_B) to Alice;
3. Alice outputs $z = M_A[u, v] \oplus z_B$;

It is clear that the protocol produces the right result. By construction:

$$z = M_A[u, v] \oplus z_B = M_A[u, v] \oplus M_B[u, v] = T[u - r, v - s] = T[x, y] = f(x, y)$$

We want to argue that the protocol is secure against passive adversaries. To do so, we need to construct a simulator S that, given the input/output of the functionality, produces a simulated view which is indistinguishable from the one in the "real world" (in the protocol execution). Remember that the *view* of a party contains the party's *input*, *internal randomness* (if any – note that in this protocol Alice and Bob do not need to make any random choices) and *all messages received during the protocol* (the messages that the party sent do not need to be included in the view, since they are a function of the other values above). So the views in this protocol are:

$$\mathsf{view}_A = \{x, (r, M_A), (v, z_B)\}$$

$$\mathsf{view}_B = \{y, (s, M_B), u\}$$

Since this is a deterministic functionality and we are only looking at semi-honest corruptions, we can look at the distribution of the output and the views separately (see also [HL10, page 21]).

The simulator for Alice works as follows (the simulator for Bob is easier and is left as exercise):

1. The simulator gets as input $x \in \{0, 1\}^n$ and $z \in \{0, 1\}$;
2. Sample uniformly a random $z_B \in \{0, 1\}$, a random $v \in \{0, 1\}^n$ and a random $r \in \{0, 1\}^n$;
3. Construct a matrix M_A in the following way: for all $(i, j) \neq (x+r, v)$, choose $M_A[i, j] \in \{0, 1\}$ uniformly at random. Finally define $M_A[x + r, v] = z \oplus z_B$;

It is easy to argue that the view of Alice in the real protocol and in the simulated execution has identical distribution: In both cases the values r, v and $M_A[i, j]$ for all $(i, j) \neq (u, v)$ are chosen uniformly at random; the pair $(M_A[u, v], z_B)$ is in both cases a pair of random bits such that $M_A[u, v] \oplus z_B = z$. This can be seen as a simple application of the "principle of deferred decision" (it does not matter in which order the random elements are sampled before they are revealed).

Pro and Cons of OTTT: The OTTT protocol has the following properties:

- ☺ Perfect (unconditional) security.
- ☺ Optimal round complexity (each party only sends one message).
- ☺ (Essentially) optimal communication complexity (total of $2n + 1$ bits. There is a protocol that does it with only $2n$ bits, and it can be proven that one cannot go below that for "interesting" functions, see [IKMOP13]).
- ☹ The main problem with the protocol is that the *storage complexity* (measured in the number of bits that parties need to receive from the dealer) is exponential in n[2], therefore the protocol can only be used for inputs of very small size.

[2]Note that one could generate M_B (or M_A) using a pseudorandom generator. In this case (at the price of going from unconditional to computational security) the storage complexity of one of the two parties can be made small, but the other party still needs exponential storage.

3 The BeDOZa protocol – Passive Security

In this section we describe the *BeDOZa protocol with passive security*[3] which is a protocol for secure two-party computation which allows to securely evaluate circuits (as opposed to truth tables) in the presence of passive adversaries and uses a trusted dealer.

Circuit Notation: We will consider a circuit $C : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$, where Alice inputs the first n bits and Bob inputs the second n bits. We consider circuits which are composed of four different kind of gates: two unary gates (XOR with constant, AND with constant), and two binary gates (XOR of two wires, AND of two wires). Gates have unbounded fanout (that is, the output of one gate can be fed as input to any number of gates) and we do not allow cycles (that is, gates are ordered in *layers* and a gate in layer i can only receive inputs from gates with layer $< i$).

Invariant: The BeDOZa protocol works on secret shared bits. For each wire in the circuit which carries a value $x \in \{0,1\}$, we write $[x]$ (x *in a box*) to say that the value x is shared between Alice and Bob, in such a way that neither Alice nor Bob has any information about x.[4] Formally when we write $[x]$ we mean that Alice knows a bit x_A and Bob knows a bit x_B such that (x_A, x_B) is a pair of uniform random bits under the constraint that $x_A \oplus x_B = x$.

Input Wires: For each of the n wires belonging to her, Alice samples a random bit $x_B \leftarrow \{0,1\}$, sets $x_A = x \oplus x_B$ and sends x_B to Bob. We write also $[x] \leftarrow \text{Share}(A, x)$. (The protocol for Bob's input wires is symmetric to this).[5]

Output Wires: If Alice (resp. Bob) is supposed to learn the value associated to some wire $[x]$, Bob sends x_B to Alice which outputs $x = x_A \oplus x_B$. We write $(x, \perp) \leftarrow \text{OpenTo}(A, [x])$. If both Alice and Bob are supposed to learn an output, we write $x \leftarrow \text{Open}([x])$ as a shortcut for $(x, \perp) \leftarrow \text{OpenTo}(A, [x])$ and $(\perp, x) \leftarrow \text{OpenTo}(B, [x])$.

XOR with Constant: We write $[z] = [x] \oplus c$ to denote the subprotocol for securely evaluating a unary gate which takes as input x, and outputs $z = x \oplus c$ for some constant bit $c \in \{0,1\}$[6]:

1. Alice defines $z_A = x_A \oplus c$;
2. Bob defines $z_B = x_B$;

AND with Constant: We write $[z] = c \cdot [x]$ to denote the subprotocol for securely evaluating a unary gate which takes as input x, and outputs $z = c \cdot x$ for some constant bit $c \in \{0,1\}$:

1. Alice defines $z_A = c \cdot x_A$;
2. Bob defines $z_B = c \cdot x_B$;

XOR of Two Wires: We write $[z] = [x] \oplus [y]$ to denote the subprotocol for securely evaluating a binary XOR gate which takes as input x, y and outputs $z = x \oplus y$, where both x, y are wires in the circuit and might therefore be unknown to both Alice and Bob:

[3] Many variants of this protocol for passive security have appeared in the literature before [BDOZ11], and in previous editions of this course the protocol was called the *GMW protocol* to credit the seminal work on multi-party computation of Goldreich, Micali and Widgerson. However, this generated confusion with a concept which will be introduced later in the course, namely the *GMW compiler*, thus the change of name.

[4] Note that we are abusing of notation, since when we write x we sometimes refer to the name of the wire/variable (the l-value of x) while sometimes we refer to the value associated to the wire/variable x (the r-value of x). When notation is unclear, we will use $n(x)$ to refer to the name of x and $v(x)$ for the value associated to x.

[5] Note that in the OTTT protocol the description of the protocol for Alice and Bob was completely deterministic (i.e., all their randomness was chosen by the trusted dealer), while in this protocol – as described here – parties sample some internal randomness as well. In the active secure version of BeDOZa instead we will let the trusted dealer sample the randomness used during the input phase instead.

[6] By "constant" here we mean that c is part of the description of the function, therefore c is known to both Alice and Bob – when $c = 1$ this is a NOT gate.

1. Alice defines $z_A = x_A \oplus y_A$;
2. Bob defines $z_B = x_B \oplus y_B$;

AND of Two Wires: We write $[z] \leftarrow \text{EvalAND}([x], [y])$ to denote the subprotocol for securely evaluating a binary AND gate which takes as input x, y and outputs $z = x \cdot y$, where both x, y are wires in the circuit and might therefore be unknown to both Alice and Bob. The protocol requires the presence of a trusted dealer D.

1. The dealer outputs a random triple $[u], [v], [w]$ with $w = u \cdot v$.[7]
2. Run subprotocol: $[d] = [x] \oplus [u]$;
3. Run subprotocol: $[e] = [y] \oplus [v]$;
4. Run subprotocol: $d \leftarrow \text{Open}([d])$;
5. Run subprotocol: $e \leftarrow \text{Open}([e])$;
6. Run subprotocol: $[z] = [w] \oplus e \cdot [x] \oplus d \cdot [y] \oplus e \cdot d$;[8]

Putting things together: We assume that the circuit has L wires, denoted by $x^1, \ldots, x^L$ and that there is only one output wire, namely x^L. The complete protocol for securely evaluating a circuit C works as follows: First Alice and Bob run the subprotocol Share for each of the $2n$ input wires in the circuit; Then, for each layer in the circuit $i = 1..d$ Alice and Bob securely evaluate all gates at that layer using the subprotocols for XOR and AND gates (since a gate can only get inputs from gates at lower levels, all gates at layer i can be evaluated in parallel); Once the value associated to the output wire is ready, run the $(x, \perp) \leftarrow \text{OpenTo}(A, [x^L])$ subprotocol;

Analysis: Since we are in the semi-honest case and we only considered deterministic functions, it is enough to prove that the output is *correct* and that the view of a corrupted party can be simulated. Correctness follows from inspection of the protocol: the evaluation of the unary gates and the XOR gates is trivially correct and the evaluation of the AND gates is correct since:

$$w \oplus e \cdot x \oplus d \cdot y \oplus e \cdot d = uv \oplus (xy \oplus vx) \oplus (xy \oplus uy) \oplus (xy \oplus vx \oplus uy \oplus uv) = xy$$

It is possible to simulate the view of a corrupted Alice having only access to her input/output in the following way. The simulator works as follows:

1. For each invocation of $[x^i] = \text{Share}(x^i, A)$, the simulator (behaving like a honest Alice) samples random x^i_B and sets $x^i_A = x^i \oplus x^i_B$;
2. For each invocation of $[x^i] = \text{Share}(x^i, B)$, the simulator includes in the view a message from Bob with a random bit $x^i_A \leftarrow \{0, 1\}$;
3. When $[x^k] = [x^i] \oplus [x^j]$ is invoked, the simulator (behaving like a honest Alice) computes $x^k_A = x^i_A \oplus x^j_A$; (Simulation for XOR with constant and AND with constant is done similarly);
4. When $[x^k] \leftarrow \text{EvalAND}([x^i], [x^j])$ is invoked, the simulator 1) adds three random bits u_A, v_A, w_A to the view of Alice; 2) simulates the XOR subprotocol as described above; 3) simulates $d \leftarrow \text{Open}([d])$ and $e \leftarrow \text{Open}([e])$ by sampling and adding random bits d_B, e_B to the view of Alice.

[7] That is, the dealer output (u_A, v_A, w_A) to Alice and (u_B, v_B, w_B) to Bob where $(u_A, u_B, v_A, v_B, w_A, w_B)$ are chosen uniformly at random in $\{0, 1\}$ under the constraint that $w_A \oplus w_B = (u_A \oplus u_B) \cdot (v_A \oplus v_B)$. Note that the dealer can send *all* of these tuples before the protocol starts, and he only needs to know how many AND gates there are in the circuit.

[8] **NB:** The exact descriptions of how to implement the three kind of operations run in step 6 (i.e., "$[x] \oplus [y]$", "$c \cdot [x]$", and "$[x] \oplus c$") were given above.

5. When $x^L \leftarrow \text{OpenTo}(A, [x^L])$ is invoked, the simulator adds to Alice's view the value $x_B^L = x^L \oplus x_A^L$ (the simulator knows x^L since in the passive case the simulator is given the input and output of the functionality; the simulator knows x_A^L because of the invariant and the simulation of the XOR/AND gates).

We argue that the view produced by the simulator is perfectly indistinguishable from the view of a corrupted party in a protocol execution. Note that Share and EvalXOR are simulated exactly as in the protocol. In EvalAND d_B, e_B are chosen at random instead of being the output of the computation of Bob – but in the protocol d_B, e_B are obtained as $d_B = x_B \oplus u_B$ and $e_B = y_B \oplus v_B$. Since u_B, v_B are uniformly random (in the protocol), the distribution does not change. Finally, x_B^L is distributed exactly as in the protocol since in the protocol $x_B^L \oplus x_A^L = x^L$, and in the simulation x^L is constructed as $x_B^L = x_A^L \oplus x^L$.

Pro and Cons of BeDOZa: The BeDOZa protocol has the following properties:

- 😐 Perfect (unconditional) security.
- 😐 Generic dealer: the dealer only needs to know an upper bound on the number of AND gates in the circuit, and nothing more about the computed function.
- 😐 (Essentially) optimal computational complexity (a small constant factor higher than computing the circuit in the clear).
- 😐 Round complexity proportional to the number of layers of the circuit (all gates belonging to the same layer can be evaluated in parallel).
- 😐 Communication complexity proportional to the size of the circuit (more exactly: 1 bit for each input/output wire and 4 bits for each AND gate).
- 😐 Storage complexity proportional to the size of the circuit (more exactly, each party receives 3 bits from the dealer for each AND gate).

4 Exercises

The exercises contain hints that have been "redacted" so they will not appear when you print this file. If you get stuck, you can use the hints. Can you figure out how? Congratulations! You are smarter than someone working at the Ministry of Defense of the UK (`https://www.information-age.com/nuclear-secrets-leaked-in-redacti`

Exercise 1. (Mandatory Assignment)

Implement a secure two-party protocol for the *blood type compatibility* function using the *one-time truth table* protocol. 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 Java 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();
```

Exercise 2. ("Secure" Computation?)

Can you come up with a "secure" protocol (against passive adversaries) for the bit XOR function, where only Alice learns the output, according to the definition in [HL10]?
Hint 1:
Hint 2:

Exercise 3. (Passive Security is Easy!)

Can you come up with a secure protocol for commitment, coin-flip and zero-knowledge proof of knowledge that only need to be secure against against passive adversaries according to the definition in [HL10]?
Hint 1: **Hint 2:**
Hint 2:

Exercise 4. (The dealer in BeDOZa)

In the BeDOZa protocol Alice and Bob need help from a dealer which outputs (u_A, v_A, w_A) to Alice and (u_B, v_B, w_B) to Bob where $(u_A, u_B, v_A, v_B, w_A, w_B)$ are chosen uniformly at random in $\{0,1\}$ under the constraint that $w_A \oplus w_B = (u_A \oplus u_B) \cdot (v_A \oplus v_B)$.
Suppose instead they have access to a dealer which provides outputs from a simpler distribution, namely it outputs random (r_A, s_A) to Alice and random (r_B, s_B) to Bob such that $r_A \oplus r_B = s_A \cdot s_B$.
Could Alice and Bob complete the EvalAND subprotocol using this dealer as well? (Perhaps using using multiple samples from the second distribution?)
Hint 1:

Exercise 5. (Arithmetic Circuits)

These notes contain the BeDOZa protocol for securely evaluating Boolean circuits against passive corruptions. It is sometimes better to compute over arithmetic circuits (that is circuits where each wire carries a value in some ring, say Z_m, and gates can either be additions or multiplications modulo m). Try to generalize the BeDOZa protocol from Z_2 to Z_m. Check that it works!
Hint 1:

Exercise 6. (Think Adversarially!)

How insecure is the OTTT protocol against malicious adversaries (that is, adversaries that do not follow the protocol and send arbitrary messages?). How many attacks can you find? Can you break any interesting security property? Are there properties which are still preserved, even against malicious adversaries?

Exercise 7. (Think Adversarially!)

How insecure is the BeDOZa protocol against malicious adversaries. How many attacks can you find? Can you break any interesting security property? Are there properties which are still preserved, even against malicious adversaries?

Hint 1:

References

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi and Sarah Zakarias.
Semi-Homomorphic Encryption and Multiparty Computation
EUROCRYPT 2011.
Available at `http://eprint.iacr.org/2010/514`

[HL10] Carmit Hazay, Yehuda Lindell
Efficient Secure Two-Party Protocols
`http://lib.myilibrary.com.ez.statsbiblioteket.dk:2048/Open.aspx?id=300348`

[IKMOP13] Yuval Ishai, Eyal Kushilevitz, Claudio Orlandi, Sigurd Meldgaard and Anat Paskin.
On the Power Of Correlated Randomness for Secure Computation.
TCC 2013.
Available at `http://cs.au.dk/~orlandi/tcc2013-draft.pdf`

[NNOB12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, Sai Sheshank.
A New Approach to Practical Active-Secure Two-Party Computation
CRYPTO 2012.
Available at `http://eprint.iacr.org/2011/091`