CS 4501-6501 Topics in Cryptography 5 April 2018

Lecture 11, Secure Computation

Lecturer: Mahmoody *Scribe: Ben Hughes, Juan Palacio Duque, Matthew Bielskas*

1 Introduction

Last time, we introduced zero knowledge proofs and within the context of the three-colorability problem demonstrated a protocol which would allow Alice, having knowledge of a 3-coloring for a particular graph G, to prove to Bob that they know this three coloring without revealing what it actually was. In this first section, we will review and define zero knowledge proofs. In the next section, we will introduce a more general set of problems under Secure Multi-party Computation that include zero-knowledge proofs as a special case. Towards obtaining secure MPC protocols we explain oblivious transfer, which is a specific, yet "complete" task for MPC. We will then describe how to use OT together with Yao's garbled circuit technique for obtaining 2 party computation.

2 Zero Knowledge Proofs

We go back to our intuitive picture of a zero knowledge proof. Bob, the verifier, wants to know whether Alice, the prover, knows the magic word to the door in the back of the cave, whereas Alice wants to keep the word secret. The two paths of the cave, A and B, are divided by the door and the door is out of sight. Alice and Bob agree upon a protocol where 1. Bob will wait outside the cave, 2. Alice will go in and randomly choose a path, and 3. After a specified duration of time, Bob will enter the cave and will request at random a path he wishes Alice to emerge from. If Alice emerges from the correct path, Bob has more certainty she is telling the truth, whereas he knows she is lying if she fails to pass this test. There are several elements of this exchange which we would like to capture: (a) the verifier is guaranteed up to some negligible probability that the protocol reveals whether or not the prover knows the information and (b) the prover is assured that no information is leaked in the exchange. From our cave example, it is clear that if Alice chose a path at random and did not know the word, there would be a 50% probability Bob would choose the same path and she would not be caught. This means multiple iterations of the exchange will be required, so that by the n^{th} iteration Alice only has a $\frac{1}{2^n}$ chance of fooling Bob if she did not know the word. It was also necessary that Bob wait outside the cave to remain out of earshot- from Bob's perspective, he could have easily "simulated" what he observed by entering by himself, choosing a random direction and then coming back. Both of these aspects are captured in the following definition.

2.1 Definition of Zero Knowledge Proofs

Recall that a language $L \in \mathbf{NP}$ if there exists a deterministic verification algorithm $V(\cdot, \cdot)$ such that:

1. $x \in L$ if and only if there exists some input $w \in \{0,1\}^*$ such that $V(x, w) = 1$
2. V runs in polynomial time.

We refer to w as the "witness", since it is sufficient to know w to prove that x has membership in the language.

Examples (1) Let L = $\{G = (V, E)|$ G is 3-colorable$\}$. Then given G = (V,E) $\in L$, there exists c: V $\to \{1, 2, 3\}$ such that for any i, j $\in V$ if (i, j) $\in E$ then c(i) $\neq$ c(j) (i.e. there exists a three-coloring of the graph).
(2) Let L = $\{G = (V, E)|$ G is Hamiltonian$\}$. Then given G = (V,E) $\in L$, there exists a bijection $\pi : V \to V$ such that π is a $|V|$-cycle (i.e. there exists a Hamiltonian path).

With this in mind, we say an interactive protocol between a *prover* P and *verifier* V given an NP language L is a Zero Knowledge Proof if:

1. **Completeness:** P knows $x \in L$ and P and V do not deviate from the protocol then Pr[V(x) = 1] $\approx$ 1. This is to say, when P and V are "honest" the protocol works.
2. **Soundness:** For all *malicious* P^* where $x \notin L$ Pr[V(x) = 1] $\leq$ negl(n)
3. **Zero Knowledge:** For all (even *malicious*) polynomial-time V^* there exists a polynomial-time *simulator* S such that S(x) is indistinguishable from $\mathsf{View}\langle V \rangle$ where $\mathsf{View}\langle V \rangle$ is the union of all the information visible to V in the interaction.

Here, we introduced some new terms which will reappear in other contexts (such as multi-party computation) as well.

- A **malicious** party deviates from protocol either to gain secret information or fool the other parties.
- A **semi-honest** party follows the protocol, but will try to use a log of the protocol to obtain secret information.
- A **simulator** would receive the relevant inputs (and sometimes the final output) of some of the parties, perform some efficient computation based off of these inputs, and construct a view which is indistinguishable from the real view.

Why ZK proofs for NP complete problems is crucial. Last lecture, we saw such a protocol for a 3-colorable graph. We know 3-coloring is NP-Complete, so given any L in NP with verifier V$(\cdot,\cdot)$ there exists a polynomial-time algorithms T_L and S_L, where $x \in L$ if and only if $T_L(x)$ is 3-colorable and V(x, w)= 1 if and only if $S_L(w)$ is a 3-coloring for $T_L(x)$. Using this reduction, we suddenly have a Zero Knowledge Proof for all problems in NP! The reason is as follows. If prover wants to convince the verifier that $x \in L$ and he has a witness w in hand. They both transform x into $T_L(x)$ which is a graph G and the prover tries to prove that G is 3 coloring. Note that the prover (and not the verifier) would also run the transformation $S(w)$ to turn it into a "3 coloring" for the graph G and uses it in its interactive proof of the 3 colorability of G.

3 Secure Computation

The idea of secure multi-party computation is to enable for two (or more) parties to jointly compute a function over their inputs while keeping those inputs and all other information private. The basics of this problem are illustrated by Yao's billionaire problem, in which we want to know which party has the largest amount of money without actually disclosing their information. In an ideal (but not realistic) scenario, we have two parties that are interacting with a trusted third party (TTP). This intermediary will perform the verification by receiving the information independently from the prover and the verifier. The prover will send x, w and the verifier will send x to the TTP, which runs $V_L(x, w)$ and sends the answer back to the prover and the verifier. We do not allow that these inputs be altered after they have been sent. Either the prover provides a correct witness to the TTP, to which TTP outputs one, or the prover provides a witness that will not pass the verification and the TTP outputs zero. In this ideal scenario the prover cannot hide if w is not valid and reveals that the prover does not know if x is part of the language. This is because x is a member of the language if and only if there is a w that gives a correct output for the function V_L.

In a real world model, we want to simulate the role of this third party such that multiple parties can achieve the same objectives as obtained in the ideal model.

3.1 Definition of Multi-Party Security

We define security for Party one (P1) and (P2) as the following:

- There exists some simulation $\mathsf{Sim}_2()$ that takes the view of P2 in the ideal world and simulates $\mathsf{View}\langle P_2\rangle$ of the real world.
- There exists some simulation $\mathsf{Sim}_1()$ that takes the view of P1 in the ideal world and simulates $\mathsf{View}\langle P_1\rangle$ of the real world.

Note that the view of the parties in the ideal work only consists of their inputs and outputs, and does not say anything more about other parties' inputs and outputs (unless that

information could directly be derived from the parties' own input and output). Therefore, the definition above basically says that nothing has leaked in the real world beyond what is inherently going to leak even in the ideal world.

We will design secure computation protocols in steps. We first focus on a "simpler" task for a specific problem, and then we will use it to build protocols for general functions.

4 Oblivious Transfer and Trapdoor Permutations

We want to design one protocol for some problem such that after solving it, we can use the result as a building block for solving two-party computation for a general function that is polynomial-time computable. So we will use a reductionist approach again. Before we assumed the the output for two parties was the same, but with OT we now assume they could be different.

4.1 Definition of Oblivious Transfer

Suppose Alice and Bob are engaging in business with the following inputs and outputs:

- Alice has pieces of information, say x_1 and x_0, that form input x.
- Bob has a bit of input b (that forms input we usually call y).
- Alice is supposed to get no extra output at the end.
- Bob is supposed to obtain x_b. Namely, if Bob "chooses" $b = 0$, then Bob is going to get x_0, and if $b = 1$, then he gets x_1 instead.

Then it holds that Alice has security in that Bob only reads x_1 or x_0, while Bob has security in that Alice does not know the value of b. Achieving one of these is trivial, but both provides a challenge. This is analogous to earlier with proving soundness and zero-knowledge (**2.1**). Alternatives to this definition include bit b being a full string (construct bit by bit), or Bob not having a choice in b so that he gets a random piece of information (random OT). Note that they all are still equivalent, though we don't prove this in class.

IMPORTANT:

- The above definition of OT is not formal. However, such a formal definition is the same as the one for general two-party security that we discussed before. That definition directly captures the requirement that Alie is not able to learn anything about b, because in the ideal world, Alice does not get any output, and the simulator for Alice's view will simulate what Alie sees in the *real* world, only based on her own input. On the other hand, Bob shall not be able to read both of Alice's inputs, because in the ideal world, Bob can only submit *one* input to the trusted third party, and so the simulator would simulate his view based on that particular input and the output obtained by

Bob (i.e., x_b). To be concise, what is learned in the 'real world' is simulatable based on information in the ideal world. Alice outputs nothing so she is simulated just based on input x. Meanwhile the simulator for Bob takes b and x_b and generates a fake transcript indistinguishable from the real one.

4.2 Semi-Honest Oblivious Transfer from Trapdoor Permutations

So now it must be asked how we can achieve security for OT. Let's focus on the Semi-Honest case where parties follow the protocol honestly and the simulator only generates the view of the honest party. Note that proving security in this case is a stepping stone toward malicious security, yet it can still be useful on its own. In particular we will use semi-honest secure OT to get semi-honest secure general 2 party computation (i.e. Yao's idea) and then that protocol can be made also secure against malicious parties by other tricks.

In addition, semi-honest security can be enough as the security model when two big agencies have an image to keep up and thus cannot afford to deviate from the protocol. However they can still log everything they send to each other; security here would imply that this action is useless to them later.

1st try (not secure):

1. Sender has bits $x_1, x_0 \in \{0,1\}$, receiver has bit b.
2. The receiver gives two encryption keys k_1, k_0 to the sender, who then delivers $c_1 = \text{Enc}_{k_1}(x_1)$ and $c_0 = \text{Enc}_{k_0}(x_0)$ to the receiver who now performs $\text{Dec}_{k_b}(c_b)$.

A problem is that if the receiver gives two keys, then the sender has both ciphertexts and both keys. So, even though it looks like a "semi-honest" player would leave the ciphertext and not decrypt it, the information that the receiver observes already contains more than it should, as extracting x_1, x_0 both from the view of the receiver is possible. So after running the protocol honestly they can read the information of x_1 and x_0 via the transcript (view of the receiver). (Think of the view being logged, and later trying to extract information from it.)

Why it is close to a good definition:

- The sender cannot cheat because they just encrypt x_1 and x_0 without knowledge of what the receiver will do with the ciphertexts.
- It is not hard to modify this definition so that we have a good definition. What we need is a way to make sure one of the keys that the sender sends is useless!

Say one of the keys the receiver sends is unable to be decrypted. This would rely on an encryption scheme where there are correct and incorrect ways to generate keys, and the real ones are indistinguishable from the fake. Through this the receiver limits their own potential to cheat. For such an encryption encryption scheme, trapdoor permutations can

be used. More formally, we will have a protocol in which the sender also helps the receiver in generating the two keys where one of them is useful and one of them is not!

Trapdoor Permutations:

1. The sender generates a pair of a permutation π and its inverse (aka trapdoor) π^{-1} gives permutation $\pi : \{0, ..., N-1\} = \{0,1\}^n$ to the receiver and keeps π^{-1} which could invert it.

   Discussion Denote $\alpha = \{0, ..., N-1\}$ and $\beta = \{0, ..., N-1\}$ so that $\pi : \alpha \to \beta$ (they are technically same sets, but better to see them like this for clarity). Suppose the goal is to pick "keys" k_1, k_0 from the set α and send them to the sender and use the same protocol that we discussed above. Instead of keys k_1 and k_0 picked in α like one might expect, let the receiver pick k_1' and k_0' in β, and send them to the receiver, and the agreement between them is that the receiver has to pick the "corresponding keys" k_1, k_0 such that $\pi(k_1) = k_1', \pi(k_0) = k_0'$.

   But the receiver wants to pick these two keys in a way that she does not actually know one of them! So, what she can do is to pick one of the k_1', k_0' directly at random, and choose the other one by picking the corresponding key in set α.

2. What she will actually do is very simple. She will pick: choose $k_b \in \{0,1\}^n$ and calculate $k_b' = \pi(k_b)$ while randomly allowing $k_{1-b}' \in \{0,1\}^n$.

3. The sender will use the trapdoor permutation π^{-1} to get the "actual keys" $k_1 = \pi^{-1}(k_1')$ and $k_0 = \pi^{-1}(k_0')$. (She does not know which one is "fake" and which one is not, as they are both uniformly sampled from $\{0, \ldots, N-1\}$. The sender provides the receiver with $\text{Enc}_{k_1}(x_1) = c_1$ and $\text{Enc}_{k_0}(x_0) = c_0$.

4. The receiver knows only one of the keys which is k_b, using which she decrypts c_b and obtains x_b.

The receiver is honest in this protocol because they use π to get k_1' but not k_0'. This relies on π and π^{-1} being asymmetrical. When constructing a full proof for this usage of trapdoor permutations, we need to construct simulators and we will skip doing so here.

5 Yao's Solution for 2 party computation using OT: Garbling of Circuits

Now assume there is one function f, and both Alice and Bob want to compute $f(x, y)$ where x, y are their local inputs that they try to keep as private as possible. Let's use OT (an asymmetric function) to solve the symmetric problem where Alice and Bob want to compute f. Ignoring the full details, we recall that OT allows someone to read exactly one piece of

information from your computer (among two possible pieces) without you knowing which one is read. On the other hand, the "reader" cannot read both of the pieces of information. Using OT, a solution to this problem (of computing f securely and privately) is known as Yao's Garbled Circuit. It uses the fact that if $f(x, y)$ is polynomial-time computable, then it can be represented as a circuit.

Setting Things Up

- Together Alice and Bob can write $f(x, y)$ as circuit $C(x, y)$
- Specific groups of wires x and y are inputs from Alice and Bob into the circuit. As the wires lead to logic gates (which in turn lead to logic gates), we get an output $f(x, y) \in \{0, 1\}$. In this setting we are assuming that f is simply Boolean, but if there were more input bits, we can write down several functions and compute all of them securely (so the Boolean case is enough).
- We will only use NAND gates (for simplicity) and reduce the problem from security of the circuit to security of an NAND gate.
- From computations, the two parties will not learn anything besides the output bit.

5.1 Garbled Circuits with OT and SKE

Yao's intuition is to ask Alice to encode the circuit so it is obfuscated yet usable. Bob could still plug in inputs, but he would need Alice's help to compute.

1. Alice writes f as a circuit C, with both of their inputs x and y required.
2. Alice converts C into a garbled version G ("different language") which hides the computation and only reveals the output. Thus Bob needs Alice's help to traverse the circuit from his input.

   High level description of the "new language" for the garbled circuit. For every wire w in this circuit (including the input wires as well as the output wire) there are two random "keys" k_{w0}, k_{w1} that would represent whether that wire is carrying zero or one. We might call these "keys" also "lables". The flow of computation is to start from the right wires for the inputs, then compute the right wires for the intermediate wires, and find the right wire for the output, and then ask Alice about what that wire means. In other words: Alice would send the garbled circuit (the description of "gates" in the new language) to Bob. Then Bob will find the wires for the inputs of himself and Alice's inputs, and then Bob will do the computation to find the right wire for the output. Then he sends it to Alice so that both of them would know what the output is.

3. Alice sends G and its related keys (on a gate-to-gate level) to Bob. Note that although she does not send wires, she sends the garbled version of the truth table for each NAND gate.

4. Alice also sends the right wires for her own input to Bob.

5. Bob can get the right keys for his own inputs using OT protocol. With this he doesn't ask what the labels for 0 or 1 are, but rather the labels of his own inputs without Alice knowing what they are.

To recap, going from gate-to-gate Bob would have the ability to obtain the output if he knew the values of every label. The OT in step 4 ultimately allows him to do this without revealing his input or learning about the circuit. Also he must put in Alice's garbled input for every gate, though there is no harm in her revealing this to him (Bob only learns via OT what label his input corresponds to). This garbled circuit being secure implies that Bob could not get an output if he put in all inputs x and y.

Garbling Truth Table:

Now we have to describe the core idea: how to garble a NAND gate so that: if Bob has the right labels for the input wires, he can compute the right wire for the output wire *without* talking to Alice and *without* knowing what actual inputs the input lables are representing. This is the core part of the Yao's protocol.

First note that a NAND $(x, y \to z)$ can be represented as truth table $(0,0 \to 1), (0,1 \to 1), (1,0 \to 1), (1,1 \to 0)$. So, if someone knows the inputs, they can use this table to find the output. We want the garbled truth table to be filled with "keys" and "encrypted information" so that only having the right labels for the.

First idea: Say someone knows k_{0y} and k_{1y}, they should be able to obtain k_{1z}. What we can do is to give 4 ciphertexts that together form the garbled version of this particular NAND gate. (Note that we have to garble every single NAND gate separately.) These 4 ciphertexts are generated using the logic of the NAND gate. They are based on the idea that we use the labels of the input wires as key to "double encrypt" the label of the output. So someone who has the right keys can decrtypt and find the output wire's label.

$c_1 = \text{Enc}_{0y}(\text{Enc}_{0x}(k_{1z})), c_2 = \text{Enc}_{0y}(\text{Enc}_{1x}(k_{1z})), c_3 = \text{Enc}_{1y}(\text{Enc}_{0x}(k_{1z})), c_4 = \text{Enc}_{1y}(\text{Enc}_{1x}(k_{0z}))$.

Note that having these 4 ciphertexts, and having the right keys, one can find the right output. But, there are 2 issues here:

- (1) How can we (as Bob) choose the "right" ciphertext and decrypt it. After all, all 4 of them might be "decryptable" while 3 of them might give us junk information.

Fixing the first issue We can fix the first issue by using an encryption algorithm that does return $\perp$ if we try to decrypt using wrong keys. For example, we can use a special message $m = 00000 \cdot 0$ of sufficiently long length, and put it inside the plaintext (next to labels), so that when we try to decrypt using wrong keys, we notice that m is not zero anymore. Another idea is to use the CCA encryption that we constructed in class (the one that also uses internal authentication). This way, when we try to decrypt using the wrong keys, we would actually receive $\perp$ as response.

But now, a second issue comes up:

- (2) If Bob knows which one of the 4 ciphertexts is decryptable, then he can actually find out the inputs and outputs! For example, if he succeeds in decrypting c_3, then he knows that $x = 0, y = 1, z = 1$.

Fixing the second issue We can fix the 2nd issue as well, by randomly permuting the ciphertexts that are sent to Bob. This way, Bob would not know the exact index if the ciphertext that he can decrypt.