Lecture 4

Gidon Rosalki
2026-05-13

Notice: If you find any mistakes, please open an issue in the github repository

1. Reminder and Introduction

We have discussed Zero Knowledge, that allows us to perform very specific shared computations of whether something is part of a computational language. We then moved on to another form of shared computation Oblivious Transfer, to allow us to create shared information between 2 parties, without sharing the foundational information. We used OT to create 2PC in the format of honest, but curious, and by combining this with ZK we can create 2PC with malicious actors.

We are not interest right now in formal proofs, more in the concepts. We did not prove 2PC with malicious actors, since it is a very technical multi page proof. It is within our capabilities to understand, and Ilan states that he would happily turn us in the right direction to find it, but it is not sufficiently interesting to be taught in the course.

Today we will discuss Multi Party secure Computation, where it is much like 2PC, but with n parties, rather than 2. Similar to 2PC, we can convert a proof for honest but curious to malicious through ZK, so today we will focus on the concepts for honest but curious.

2. Communication Model

This is slightly more nuanced than 2 actors, where for 2 actors we simply had a communication channel between them. For MPC, where we have n actors, it is a little more nuanced. We have the actors $p_1, ..., p_n$, and are left with the question of how they communicate. Perhaps there is a channel between each pair, or perhaps they communicate down the chain, or many other options.

Let us begin by considering 3 actors. We can have the model where there is a channel between each pair of actors, called peer to peer model (P2P). We also have the *broadcast* model, where every actor writes information on a shared board, where they can all see what is on the board.
We can also ask what the malicious actor can do. For example, in P2P, if our malicious actor is p_3, can he see the channel between p_1 and p_2? For the broadcast model, this is obvious, but needs to be defined for P2P. Additionally, we can consider if messages are authenticated, such that we know for certain from whom messages were sent? What about encrypting the P2P channel, such that p_3 can see that a message was sent, but not know its contents?

For most of this lecture, we will use the simplest broadcast model, with authenticated, unencrypted messages. This is also the accepted model in the literature, since it is also useful for abstraction. If we create a model for broadcast, then we can use essentially the same model for P2P without too many alterations.

3. Information Encoding

3.1. Secret Sharing

Let us suppose that we have a secret/message s. We want to split this message into parts $s_1, ..., s_n$, such that

$$1. \forall T \subseteq \{s_1, ..., s_n\} : |T| = t \text{ can be used to recover } s$$
$$2. \forall T \subseteq \{s_1, ..., s_n\} : |T| \leq t - 1 \text{ we know nothing about } s$$

Point 1 is rather similar to erasure codes, such that part of the data enables us to create the rest, but 2 is simply a security parameter. Let us consider a few examples:

- For $t = 1$ then we may set $s_i = s$
- For $t = n$ then if we assume that s is only one bit, then we will set $s_n = \bigoplus s_i$. This can also be extended to s being multiple bits.

- For $t = 2$ then $\forall i, j$ we will create 2 out of 2 secret sharing of s. That is to say, we perform secret sharing as though $n = 2$, by creating $r_{ij}, s \oplus r_{ij}$ to be our pairs for every pair. This requires $\approx n$ bits per party, and if we increase t then we will need n^{t-1} bits per party.

We also have a more general solution: **Shamir Secret Sharing**. This works for all values of t, so we will show it for $t = 2$, and leave it as an exercise to the reader to increase. We choose some random polynomial, of rank $t - 1$, so:

$$p(x) = a_1 x + s$$

So in our case, the only thing we are sampling is a_1, over the field $\mathbb{F}_q$, such that $q \geq n + 1$, and we will have q be some prime to make everything in the field equally likely (often, the first prime bigger than $n + 1$). We set

$$\begin{aligned} s_1 &= p(1) \\ s_2 &= p(2) \\ s_3 &= p(3) \\ &\vdots \\ s_n &= p(n) \end{aligned}$$

So, in our example for $t = 2$, p is a straight line, s is our y intercept, and a_1 is the gradient. If we are another actor that does not know this line, then any two points on this line will tell us the entire line, and thus we can compute the value of s, without the value of s ever being revealed (correctness). However, if we only know 1 point, since we know nothing about the gradient of the line, and so s is not revealed to the adversary (security). Since $q \approx n$, a point on the field will be $\log q$ bits, and so we now only need $\log n$ bits per party, which is much better.

This can be extended to $t = 3$, but now $p(x) = a_2 x^2 + a_1 x + s$. Once again, s is the y intercept, 1 point, and 2 points do not teach us enough about the parabola, but 3 points will tell us the entire parabola, and thus the y intercept. We will note that for this, for *every* value of t, we only need $\log n$ bits per party, it is **not** dependent on t.

3.2. Interpolation

Given $p(x_1) = (x_1, y_1), ..., p(x_t) = (x_t, y_t)$, we can recover the polynomial

$$p(x) = a_{t-1} x^{t-1} + ... + a_1 x^1 + a_0$$

through the *Lagrange Coefficient*:

$$\begin{aligned} p_i(x) &= \prod_{i \neq j} \frac{x - x_j}{x_i - x_j} \\ p(x) &= p_1(x) \cdot y_1 \\ &\quad + p_2(x) \cdot y_2 \\ &\quad \vdots \\ &\quad + p_t(x) \cdot y_t \end{aligned}$$

We will note that this may be done linearly, and it is all independent, so it is nice and fast.

Fun fact: We have made it down to $\log n$ bits per actor. Can we improve this further? Nope. When working $\forall t$, this is the best scheme that is possible (Proven by Ilan Komargodski, that name looks distinctly familiar, such a shame he cannot provide any intuition on the matter)

3.3. Complications

We may want to complicate the concept, for example, a boss on his own holds the key, between his 3 underlings, they need 2 of them to agree in order to recreate the key, and from their underlings, we need more people, and that they have all passed a security test, and so on. We can complicate this as much as we like, and still achieve the described security.

Let us consider performing for some group $A \subseteq \{0,1\}^n$. We may express each group similar to the following manner: $A = (p_1 \wedge p_3 \wedge p_5) \vee (p_5 \wedge p_4 \wedge p_1) \vee ....$ We know we can achieve this such that we just create groups of 3 according to what we know from Shamir Secret Sharing, but this is relatively inefficient. We can instead create a binary tree, where the left children are $\wedge$, and the right children $\vee$, and from here construct the required groups.

4. MPC

Now that we have done the necessary background work, welcome to Multi Party Computation. Here we have parites, with an adversary budget of t. We have two main types of security:

1. Statistic security: Requires $t < \frac{n}{2}$
2. Computational security: Requires $t = n - 1$

Let us consider we have two inputs, $x_1, ..., x_n, y_1, ..., y_n$, for parties x, y. We will demonstrate for $\oplus$ and $\wedge$, both operating on 1 bit of input (e.g. $x_1 \oplus y_5$), since with those two, we may compute everything. We want to use secret sharing, such that neither party may compute the entire input of the other, but between them, they may compute the result. To do this:

1. x will choose s_1, and send s_1 to y, and will compute for himself $x_1 \oplus s_1$
2. y will compute r_1, send it to x, and internally save $y_1 \oplus r_1$

We will do this for every entry wire. Let us consider a XOR gate. We will say that x has values r_1, r_2, and y has values s_1, s_2. x will compute $r_1 \oplus r_2$, and y will compute $s_1 \oplus s_2$. If we now compute the XORs of these values, then we can create the XOR of the inputs.

If we instead consider the AND gate, we will need to use OT, since we showed last lecture that we need OT for this type of communication. x has the values r_1, r_2, y has the values s_1, s_2. We may now construct the OT table:

s_1	s_2	s_3
0	0	$(r_1 \wedge r_2) \oplus r_3$
0	1	$(r_1 \wedge \neg r_2) \oplus r_3$
1	0	$(\neg r_1 \wedge r_2) \oplus r_3$
1	1	$(\neg r_1 \wedge \neg r_2) \oplus r_3$

x will choose r_3. So, x knows the values of the rs, and may compute the values for the s_3 column. By sharing between them, they may compute the correct row from the table. From here, we may compute:

$$r_1 \oplus s_1 = \gamma_1$$
$$r_2 \oplus s_2 = \gamma_2$$
$$\gamma_1 \wedge \gamma_2 = \gamma_3 = r_3 \oplus s_3$$

As required.

This may now be relatively simply extended to n parties. Let us begin with 3: We may fairly simply create a circuit between 3 parties, such as $(p_1 \oplus p_2) \oplus p_3$. p_1 and p_2 will now make use of secret sharing such that neither of them have the final value, but that computation may take place with p_3 in order to compute the results of this gate.

4.1. Statistical Security

Let us now assume that we have n actors, $p_1, ..., p_n$, and the adversary controls $t < \frac{n}{2}$. Every actor will divide his secret into n parts, and share the i-th part with p_i. This means that each player will have 1 part of each key, so:

$$p_1 \text{ has } q_1(1), q_2(1), q_3(1), ..., q_n(1)$$
$$p_2 \text{ has } q_1(2), q_2(2), q_3(2), ..., q_n(2)$$
$$p_3 \text{ has } q_1(3), q_2(3), q_3(3), ..., q_n(3)$$
$$\vdots$$

For now, we have nothing. Let us assume we want to compute $s_1 + s_2$. If now, every player p_i computes $q_i(1) + q_i(2)$, then the free variable will be $s_1 + s_2$. Since interpolation is a linear operation, our actors now hold a part of the shared secret for $s_1 + s_2$. We can do the same thing above multiplication, and achieve $s_1 \cdot s_2$, since the free variable will be $s_1 \cdot s_2$. However, the rank of the polynomial will change, and this is not in fact a random polynomial like may be created over addition, since it will definitely be decomposable into 2 polynomials.

To resolve this, we will perform another round of secret sharing, which will require $2t$ participants, and share the value 0 between everyone, such that we can add this on, and turn our new polynomial into a polynomial that is not necessarily decomposable.
To resolve the rank of the polynomial, we will simply ignore all the variables of rank $\geq t$, and continue as if they had never been there. We have shown this to be easy for addition, and similarly for multiplying by a scalar. This gives us all the linear functions (as long as $t < \frac{n}{2}$). So, given a linear function, we may securely compute this between n people. We now want to show that ignoring the variables above a rank is a linear function. We can do this since interpolation is a linear operation, where we project from $2t$ points to t points, and then interpolate to the polynomial. This holds since interpolation is essentially using Vandermonde matrices, where the standard Vandermonde V converts points to polynomials, and V^{-1} converts polynomials to points. So, our linear operation is now simply $V^{-1}PV$, where P is the projection.