Lecture 1

Gidon Rosalki
2026-04-28

1. Introduction

Last semester, in the introduction course, we mostly focused on communication between Alice and Bob, with Eve sitting in the middle, perhaps observing, perhaps as a Man in the Middle, and so on. We covered this for both public and private keys. This semester we will be extending beyond simple communication. For example, we will consider computation.

2. Communication

We may argue that communication is a type of computation. Consider if we have 2 people, Alice and Bob, with their own respective functions, and Alice wants to compute $f_{A(x,y)}$, where Bob wants to compute $f_B(x,y)$. If we consider the case $f_{A(x,y)} = 0$, and $f_B(x,y) = x$, then we may consider this to simply be Alice sending a message to Bob, since Alice has learnt nothing from her function, but Bob has learnt the sent value of x from Alice.

Let us instead consider $f_A = f_B = \max(x,y)$. This means that the protocol is to find the largest of two values. So the first person, that sent the largest value learns that they had the largest value, and the second person learns the other value.
If instead

$$f_A = \begin{cases} 0 \text{ if } x > y \\ 1 \text{ if } x \leq y \end{cases}$$

then both actors have learnt one bit of information. This sounds impossible, how can I check which of us has a larger value without finding out the second value?
This is a extension of communication, and we will discuss how to do this for every function.

2.1. Multiparty Communication

The next case is instead of only have 2 actors, we could have k actors. Each has their own secret input $x_1, ..., x_k$, and we want them to communicate with each other through our function

$$f(x_1, ..., x_n) = \text{Anything we like, such as do two people share a password}$$

This is called *multiparty computation*.

This seems a weird concept, but consider that I have my password manager X, and I want to know if it has been leaked, without leaking my password. Since the database of leaked passwords is too large for me to inspect personally, I want to use this sort of system to find $\exists x \in X : x \in Y$ or if $X \cap Y = \emptyset$.

2.2. Zero Knowledge Proofs

Consider if Ilan has written an exam, and he argues that the exam is solvable (let us call this theorem Π). The students disagree. Ilan could prove this by sending them the solution, but he must then write a new exam. Another potential solution is finding a function such that $f(\Pi) = 1$, such that we can see that the exam is solvable, without being shared the solution.
This may also be demonstrated with sudoku. Here one makes many requests as to the solution, which do not leak the solution, and then after enough requests, we are convinced that Ilan has the solution, without us having learnt anything about it. A practical example is with public-private keys, where we do not want to share the private key, but may want to prove to the world that it is in fact *my* private key, and nobody else's.

2.3. Advanced Encryption Methods

Last semester we discussed asymmetric encryption, where both sides have a public key, and a secret key. This is great, but slightly limited. In modern cryptography, we use more advanced methods. We will not

cover lots of it, since there is just too much, but we will hopefully reach a significant amount of information.

2.3.1. Homomorphic Encryption

Here, we have the operation

$$\begin{aligned} \mathrm{Enc}_{\mathrm{pk}}(m) &\to c \\ \mathrm{Eval}\ (f, c) &\to c' \\ \mathrm{Dec}_{\mathrm{sk}}(c) &\to m \end{aligned}$$

As we can see here, there is a new function *Eval*. Beyond that, *Enc* and *Dec* are exactly as we saw in asymmetric encryption. This *Eval* operation, if given an encrypted value of m, and the function f, can change the message m without the secret key, and without breaking the encryption.
Methods to do this were first found around 2009 (it had been a long open question, Diffie-Hellman raised it back in 1976).

Let us consider the following example in which it is useful. Alice has the secret key x, and creates the ciphertext c, which she sends to Bob. Bob then computes Eval (f, c), and sends it back to Alice. Alice then computes $\mathrm{Dec}_{sk}(c')$ and receives $f(m)$.

We will note that last semester we defined encryption schemes as malleable where we could perform a specific operation on the ciphertext, and receive an altered, but legal ciphertext that decrypts to an altered message, and this was *bad*. However, for homomorphic encryption, we are discussing for *any* function, and this is a feature instead.

This could be useful for a number of situations. Consider if you want to perform analysis of medical data. Here, you can send an encrypted form f the medical data to the analysis server, and it may perform the analysis without reading the private medical data, and return the analysis to you.

2.3.2. Functional Encryption

This is an encryption system as follows:

$$\begin{aligned} \mathrm{Enc}_{\mathrm{pk}}(m) &\to c \\ \mathrm{Dec}_{\mathrm{sk}}(c) &\to m \\ \mathrm{KG}_{\mathrm{sk}}(f) &\to \mathrm{sk}_f \\ \mathrm{Dec}_{\mathrm{sk}_f}(c) &\to f(m) \end{aligned}$$

Or in other words, we can create secret keys that cannot decrypt the entire message, but rather decrypt the ciphertext into some function of the message. For example, consider the Israeli ID number. You could given me an encrypted form of your ID number, and a decryption key that computes if the number is a valid number, such that I cannot see the value of the ID, but I can verify that it is a correct ID.

2.3.3. Obfuscation

(We probably will not reach this, but it is very neat).

Here, instead of us obscuring data as we have done in the previous two examples, we are obscuring *logic*. Consider I have written a function in some programming language in some course. Perhaps you feel ashamed of how you wrote your solution, so you do not want to submit the code, but want to prove that you have (or less contrived, you have a new algorithm for multiplying matrices, that you want to prove that you have, but not share with the world). So here, we obfuscate the code. We will note that there are many services that obfuscate your code, this is often done to JavaScript on the internet, but with enough time, we can reverse engineer it. What we are asking is if we have a method to obfuscate it such that it *cannot* be reverse engineered, much like with Diffie-Hellman. It has even been difficult to define what we want from our definitions here. In 2013, people successfully created definitions, and even a model that does this (beyond scope of this introduction, and probably of course. Ilan has said you may turn to him with questions if you want, and he can share relevant papers).

3. Beyond Secure Communication

(This is the actual beginning, before this was just an introduction).

3.1. Classical proofs

Cook-Levin introduced the idea of NP-completeness, where the class of NP proofs are proofs which are may be verified in polynomial time (they are not in P if we do not know of a way to find the solution in polynomial time).

Definition. $\mathcal{L}$ is an NP language if there is a polynomial time verifier V where:

- **Completeness: True theorems have short proofs.**
  For all $x \in \mathcal{L}$, there is a polynomial($|x|$)-long witness (proof) $w \in \{0,1\}^*$ such that $V(x,w) = 1$
- **Soundness: False theorems have no short proofs.**
  For all $x \notin \mathcal{L}$ there is no witness. That is, for all polynomially long $w \in \{0,1\}^*, V(x,w) = 0$

Examples are the theorem that N is the product of 2 prime numbers, or y is a quadratic residue mod N. These are both very easy to verify, but difficult to compute. An example we discussed last semester is *Graphs G_0, and G_1 are isomorphic*, the verifier will: Check $\forall i,j : (\pi(i), \pi(j)) \in E_1 \iff (i,j) \in E_0$.
We will note for all of these verifiers, not only does the verifier know that the theorem is true, but how (eg, not only that the two graphs are isomorphic, but also what is the isomorphism).

3.2. Zero Knowledge Proofs

Here, the idea is to demonstrate that we have the proof, without leaking information about how the proof works. To achieve this, we have two necessary new ingredients:

1. **Interaction**: Rather than passively reading the proof, the verifier may engage in conversation with the prover.
2. **Randomness**: The verifier is randomisecd, and may make mistakes with some (exponentially small) probability.

3.3. Intuition

Let us consider a Rubik's cube, that has been randomised. We will state a theorem "There is a solution that requires $\leq k$ moves to solve the cube". We could just share the solution, but we can also prove this with ZK:

1. Prover sends verifier a "random" configuration
2. The verifier sends the prover a challenge (0, or 1)
3. The prover will:
   1. If: $0 \Rightarrow$ Show from start to random in $\frac{k}{2}$ moves
   2. If: $1 \Rightarrow$ Show from random to solution in $\frac{k}{2}$ moves

If our prover may consistently do this, then it would seem that they can actually do what they claim.

Definition. $\mathcal{L}$ is an IP-Language (Interactive Proof) if there is an unbounded P, and PPT verifier V, where

- **Completeness**: If $x \in \mathcal{L} \Rightarrow V$ always accepts, or more mathematically

$$x \in \mathcal{L} \Longrightarrow \Pr[(P,V)(x) = \text{accept}] = 1$$

- **Soundness**: $x \notin \mathcal{L}$ regardless of the cheating prover strategy, V accepts with negligible probability
  Or more mathematically, for infinite $n \in \mathbb{N}$

$$x \notin \mathcal{L} \Longrightarrow \exists p(n) : \forall P^*$$

$$\Pr[(P^*,V)(x) = \text{accept}] = \frac{1}{p(n)}$$

We can also replace 1, and the negligible function with c and s (completeness, and soundness). These are equivalent as long as $c - s \geq \frac{1}{p(n)}$

3.4. Interactive Proof for QR

Let us define

$$\mathcal{L} = \{(N, y) : y \text{ is a quadratic residue } \bmod N\}$$

Let us construct the following interactive proof (left to right is prover to verifier, right to left is verifier to prover): Both know (N, y):

$$\begin{aligned}
&\longrightarrow s = r^2 (\bmod N) \\
&\longleftarrow b \leftarrow \{0, 1\} \\
&\longrightarrow \begin{cases} b = 0 \Longrightarrow z = r \\ b = 1 \Longrightarrow z = rx \end{cases} \\
&V \text{ checks } z^2 = sy^b (\bmod N)
\end{aligned}$$

Let's prove this:
Correctness If $(N, y) \in \mathcal{L}$, then the verifier accepts the proof with probability 1:
Proof:

$$\begin{aligned}
z^2 &= \left(rx^b\right)^2 \\
&= r^2 \left(x^2\right)^b \\
&= sy^b (\bmod N)
\end{aligned}$$

So the verifier's check passes, and it accepts.

Soundness: If $(N, y) \notin \mathcal{L}$, then for every cheating prover P^*, the verifier accepts with probability at most $\frac{1}{2}$:
Proof: We will begin by noting that we repeat the proof request n times, and so we may change the claim to accepting with probability at most $\left(\frac{1}{2}\right)^n$. The rest of the proof is left as an exercise for the reader. By following through what happens if P^* sends a random number, and whether or not it is a square number, we can see that it will succeed with probability $\frac{1}{2}$ every time, so when we repeat n times, it is a negligible function.

4. Defining Zero Knowledge

So, we can see above the concept of a Zero Knowledge Proof, but what does that actually *mean*? The setting of a ZKP is that we have a Prover (P), and a Verifier (V), with some sort of communication between them. In the previous example, we have the messages s, b, r from the prover, the verifier, and the prover. Let us consider what the verifier learns:

$$\text{View} = (s, b, r)$$

Let us consider, communication as $P(x, w) \Longleftrightarrow V(x)$ (so we want to hide w), if thanks to an algorithm we will call *Simulator* such that

$$\text{Simulator } (x) \longrightarrow (x, s', b', r')$$

If thanks to x, we may create (x, s', b', r'), such that $(s, b, r) \approx (s', b', r')$ (similar probabilities), then we're very happy. Why? We have gained information that we are not meant to have.

In a more general setting:
Definition. Consider a protocol between P and V, where $[P(x, w) \Longleftrightarrow V(x)]$, the protocol is **Zero Knowledge** if

$$S_{\text{PPT}}(x) \to \text{view}' \approx \text{view}$$

So, there exists a simulator that may create very similar results to the view. How similar? This gets to the concept of kinds of zero knowledge:

- Perfect ZK (the two distributions are *identical*)
- Statistical ZK (the two distributions are *statistically indistinguishable* (negligible difference))

- Computational ZK (the two distributions are *computationally indistinguishable* (negligible difference))

We have already discussed ZK for QR. We have proven completeness, and soundness. We will claim that this protocol is ZK. We will note that our View is the following triple (s, b, z). We will build the simulator Sim $(y) \Rightarrow (s', b', z')$ as follows:

1. Pick a random bit $b \leftarrow \{0, 1\}$
2. Pick a random $z \in \mathbb{Z}_N^*$
3. Compute $s = \frac{z^2}{y^b}$
4. Output (s, b, z)

We have created a simulator, and the simulated transcript is identically distributed to the real transcript, so this protocol is *Perfect Zero Knowledge*.

We can also consider what happens if V is not honest. We had the definition:
An Interactive Protocol is *honest-verifier* perfect zero knowledge for a language L if there exists a PPT simulator S such that for every $x \in L$, the following two distributions are identical:

$$\text{view}_V(P, V) \quad S\big(x, 1^\lambda\big)$$

We now create the real definition:
An Interactive Protocol is *perfect zero knowledge* for a language L if for every PPT V^*, there exists a (expected) PPT simulator S, such that for every $x \in L$, the following two distributions are identical:

$$\text{view}_{V^*}(P, V^*) \quad S\big(x, 1^\lambda\big)$$