Lecture 5

Gidon Rosalki
2026-06-03

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

1. Recap and Introduction

We have been discussing Multi Party Computation, which includes concepts such as ZK, 2PC, OT, and so on. We have for each of these created protocols, and sub parts to allow us to create more complex concepts. We established that there is a tradeoff between efficiency, and generalisability. The more general the concept, the slower the implementation, but if we have a specific target (eg, OT), we can create a very efficient implementation.

We are now going to discuss secured computation of databases. There are two main concepts that we will discuss:

1. PIR
2. ORAM

This is still under the umbrella of MPC.

The idea is that we have a very large database, sufficiently large that we cannot store it locally, so we instead store it remotely. Storing it remotely in the cloud is not necessarily the best solution, perhaps we do not want the cloud provider to know what we stored, so we may encrypt it. However, accessing is now a little more complicated. Even if we suppose that we have encrypted it in discrete blocks, say per picture, such that we need not download the entire thing at once to access a single thing, then even then, the provider knows where and when we have accessed data. Even if they do not know what the image is, they know that some data is stored at such a location, in a discrete chunk, and that I accessed it at this time.
There are a lot of statistics, that are useful to an adversary, that may be withdrawn from this. It has been demonstrated that an entire database may be reconstructed just from the metadata of the users accessing its parts.

To resolve this, we have the things like ORAM and PIR.

2. PIR - Private Information Retrieval

Let us suppose that we have a database of size n bits. Let's suppose that the DB is static, and we only want to read, not write, such that we read bit $i \in [n]$, in such a way that we hide from the DB what index we read.

This is possible for perfect secrecy, but at cost of $O(n)$ communication, where we simply read the entire database, and throw away all the data that is not the bit that we want.
Theorem: Every scheme for perfect secrecy requires $\Omega(n)$

This is essentially a form of 2PC, where we want to get the index i from the second party, without leaking the value of i.

2.1. k Databases

Let us suppose a different setup, where we have k different DBs. We give each copy to a different provider, and assume that they are not talking to each other, i.e. Google will not tell Amazon what bit I read. We still want to read index i, without leaking i.

Theorem: There exists a solution with communication cost $O(\sqrt[d]{n}) : k = 2^d$, so for $k = 4, O(\sqrt{n})$, or $k = 8, O(\sqrt[3]{n})$.
Let us suppose that we store each database as a square matrix of size $\sqrt{n} \times \sqrt{n}$, so now we want to read i^*, j^*. We will now

1. $S, T \leftarrow \{0,1\}^{\sqrt{n}}$

2. $S' = S \oplus \{i^*\}$
   $T' = T \oplus \{j^*\}$
3. Send the pairs $(S,T),(S',T),(S,T'),(S',T')$ to the appropriate databases (first to DB1, second to DB2, and so on).
4. Receive in response X_1, X_2, X_3, X_4 such that $X_1 = \bigoplus x_{i,j}$
5. Output $X_1 \oplus X_2 \oplus X_3 \oplus X_4$

Correctness: From the laws of xor, and the fact that the only thing that appears an odd number of times is the value for (i^*, j^*), and so is the only thing not cancelled out.
Security: Each database receives a pair of uniformly distributed strings of length $\sqrt{n}$. This does not leak the data, and so is private.

2.2. Improving the Scheme

We have shown that this works for 4 databases (and more), but we can in fact improve this scheme to 2 DBs, with $O(\sqrt[3]{n})$ communication.

Our current scheme has us sending some root of n (e.g., for $k = 8$, we send $3 \cdot \sqrt[3]{n}$), but only receive in response a single bit. Perhaps we can receive a little more in response (like a response of the same size).

1. $R, S, T \leftarrow \{0,1\}^{\sqrt[3]{n}}$, and R', S', T' as before
2. We will send R, S, T to DB1, and R', S', T' to DB2.
3. We want in response the bits $x_{000}, ..., x_{111}$. We have currently received x_{000}, x_{111}. In order to get the rest, we may send all the possibilities. For example, for x_{001}, we send to the first database all teh possibilities that could be in the T position, at a cost of $\sqrt[3]{n}$. We can do this again for x_{010}, and then for x_{011} we send to DB2, and so on.
4. We thus have simulated the earlier protocol, and have achieved our result in $k = 2$, and communication cost $O(\sqrt[3]{n})$

In the general case, for k databases, we think that the communication complexity is $O\left(n^{\frac{1}{\log k + \log\log k}}\right)$

2.3. 3 DBs, $O\left(n^{\frac{1}{4}}\right)$ Communication

This is a result from 1997. It is plausible that things have since improved.

We will once again begin by creating $s_1^0, ..., s_5^0 \leftarrow \{0,1\}^{\sqrt[5]{n}}$, and $s_1^1, ..., s_5^1 (= s_5^0 \oplus \{i_5^*\})$. Note, that the previous design essentially states that we may compute all the possibilities of hamming distance 1 from a given input, so the entire 3D cube is contained in the elements of distance 1 from 000, and the elements of distance 1 from 111.

Here, there are a total of 32 vectors (from 00000 to 11111). We send 00000 to DB1, and 11111 to DB3, and are left with 20. They may compute all of hamming distance one from them, leaving us now with 20. This is a good start, and we have not yet used DB2. Let us now create a virtual database, with $n^{\frac{2}{5}}$ options, and use the above scheme that only requires 2 DBs. The protocol is as follows:

1. The user begins by creating $S_1^0, ..., S_5^0 \leftarrow \{0,1\}^{\sqrt[5]{n}}$, and $S_1^1, ..., S_5^1 (= S_5^0 \oplus \{i_5^*\})$, sends S_*^0 to DBs 1 and 2, and S_*^1 to DB3
2. The user also sends to DB1 and DB2 20 queries for any 2-database PIR protocol with $O\left(n^{\frac{1}{3}}\right)$ communication complexity. These queries are determined based on the sets that the user generated in the previous step.
3. DB1 and DB3 send back X_{00000} and X_{11111}, respectively. Also, DB1 sends back $X_{00001}, ..., X_{10000}$ for all $n^{\frac{1}{5}}$ possible values for each of $S_1^1, ..., S_5^1$. Similarly, DB3 sends $X_{11110}, ..., X_{01111}$ for all $n^{\frac{1}{5}}$ possible values for each of $S_1^0, ..., S_5^0$.
4. (We use X_{11100} as an example, but the exact same computation is carried out for each of the remaining values) DB1 and DB2 both generate X_{111000} for all $n^{\frac{3}{5}}$ possibilities of S_1^1, S_2^1, S_3^1. This results in each of these databases holding identical copies of a string of length $n^{\frac{3}{5}}$. Using the appropriate query that was sent by the user, each database computes a response using the underlysing 2 database PIR protocol.
5. The user obtains X_{00000} and X_{11111} immediately, and can easily select the values for $X_{00001}, ..., X_{10000}$ and $X_{11110}, ..., X_{01111}$ from the data sent back by the databases. For the remaining

values, the user runs the underlying PIR protocol using the appropriate replies sent back by DB1, and DB2 to recover all remaining values. The desired bit of the original data is recovered as

$$\bigoplus_{b_1=0}^{1}\bigoplus_{b_2=0}^{1}\bigoplus_{b_3=0}^{1}\bigoplus_{b_4=0}^{1}\bigoplus_{b_5=0}^{1} X_{b_1b_2b_3b_4b_5}$$

3. Oblivious RAM (ORAM)

This feels similar to PIR, but it is in fact different. Where in PIR we had database(s), and a user that is reading. In ORAM, we have a single database, and a client, where the client has a secret key sk, and may both read and write.

Let us consider that a computer has a CPU, with registers, and RAM. We trust the CPU, and its registers, but do not trust the RAM. We therefore want to encrypt the RAM, such that one cannot observe it, and learn things. The problem is that we do not want to leak information when we access cells of RAM.

The initial naïve solution is once again to simply read all the RAM, and ignore all the values that do not correspond to the i-th cell. Problem is, a simple Read(i) costs $O(n)$ messages. This is horribly inefficient.

3.1. $\sqrt{n}$ Scheme

This was originally done to make it harder to decompile programs. For example, a method to hide how programs check license keys. This was proposed by an Oded Goldreich (Israeli) in 1987. In 1991, it was improved to $O(\log^3 n)$, and under a limited model, it was improved to $\Omega(\log n)$. In 2012, the general form was improved to $O(\log^2 n)$ (Tree ORAM, PATH ORAM). In 2018, a lower bound of $O(\log n)$, without limitations was proven, and in 2020 a scheme in $O(\log n)$ was created (Ilan's name is on that paper).

Let us begin with the original, $O(\sqrt{n})$. We will begin by taking our DB (RAM), and creating a random arrangement of it (π). This will be a secret rearrangement. So now, when we access i, we are in fact accessing $\pi(i)$, which is a random location. This does come with the drawback that π does not change, so if we access the same index twice, then our adversary is aware of this. We could refresh π every time, but this comes at a communication cost of n, and so is too expensive for our taste.
We will add more to our DB, of size $2\sqrt{n}$. $\sqrt{n}$ of dummy, and $\sqrt{n}$ of cache. Whenever we read a value from the database, we read the entire cache, and if it is not in the cache, we access it from $\pi(i)$. If it is in the cache, then we access a random variable from the dummy, such that we do not access a single point of the RAM twice. This does limit us to $O(\sqrt{n})$ accesses, since otherwise we run out of space to read in the dummy without repeating ourselves. To resolve this, after $\sqrt{n}$ accesses, we start again from the beginning, reset the cache, the dummy, and create a new π. This does cost $O(n)$, but this is only every $O(\sqrt{n})$ reads, so amortizes to $O(\sqrt{n})$.

3.2. $\log^3 n$

If instead of using the naïve implementation for reading the cache, we instead apply this concept recursively, accessing the cache with the more complicated version of ORAM, then we may do this with less, achieving (with many missing details) this reduced communication complexity.