1 Lecture 2: Introduction to Hard-Core Bits

1.1 Coin-flipping Protocol

Let's say two separate (distant) parties A and B want to settle a dispute using a coin flip. How can either ensure that the other is giving truthfully random results (i.e. not manipulating the data to win), or that any falsification cannot change the output to favor one side over the other in a predictable way? The solution to this issue is to use something called a **"commitment protocol"**.

The idea of a commitment protocol is to force the parties to be honest. Or, more realistically, it is to force the parties to make and finalize their decisions before it is possible for them to know anything about what the other party is doing or has done. A commitment protocol involves one party "committing" some information, like a bit, which can be revealed, or "opened", at some later time. Essentially, a commitment protocol is like locking something in an ordinary safe. It must have two key properties:

1. A **hiding property**: Suppose A commits some information using the protocol, and then passes the committed form of this information to B. There is no (reasonable/feasible/"easy") way for B to figure out what it is that A committed using only the committed form of the information.

   That is, the information is "hidden", as if A had locked it in a safe and gave the safe to B, without the combination.

2. A **binding property**: Once A commits some information and passes the committed form of this information to B, there is no way for A to manipulate the information in any way.

   That is, the information is "bound" to a value once it has been committed and sent, as if A had locked an ordinary safe, and B did not allow A to reopen the safe and change its contents.

Now, we can use a committment protocol to establish a fair way for A and B to flip a coin remotely:

1. A flips a coin and commits the value, b_1.
2. A sends this committed value, $com(b_1)$ to B. This is binding.
3. B flips a coin, b_2, and sends the value directly to A. Note that this step does not involve any sort of committment protocol in any way.
4. A "opens", or tells B how to "open" the committed value to reveal b_1
5. The two parties compute a shared coin, $b_1 \oplus b_2$.

So how do we know this is fair/safe?

Well, assuming that our committment protocol has the desired properties, suppose that A cheats. This means that A somehow based the final value of b_1 on the value of b_2. But due to the binding property, the value of b_1 could not have been changed after step 2, which is before b_2 has even been decided.

Similarly, if B cheats, then B somehow knew the value of b_1 before sending b_2. But due to the hiding property, B could not have possibly known b_1 before step 4, and b_2 was finalized in step 3.

Thus, neither party could have possibly cheated, so the coin flip works as desired due to our commitment protocol!

1.2 Building a Commitment Protocol

Now, of course, we have to find some way to actually build a commitment protocol of some sort. We will do this by assuming the existence of, and then using, one-way functions. In fact, we assume something a little bit stronger: we assume that there exist **one-way permutations**, which are one-way functions that are bijective (for our purposes, we assume $|x| = |g(x)|$). This allows us to avoid having to worry about lookup tables based on the range of our function, since it will certainly be the same size as the domain.

Now, before we go any further, remember the definition of a one-way function. We really can only guarantee that it is "hard to invert" for very large x. In fact, since we are working with permutations, we certainly have that they must not be hard to invert for $|x| = 1$, since there are only two possible permutations on $\{0, 1\}$, so even guessing can yield a probability of 1/2.

But this presents a new problem, especially if we want to make a commitment protocol that we can use for a coin flip. We may have to have both parties randomly generate and commit strings that are thousands of bits long, but we only want each party to be able to use one bit (i.e., so they don't have some way to "change their minds" later on).

A quick, natural idea to solve this might be to agree to use the first bit of the random values. However, we quickly run into an issue here. Namely, it's not hard to create a one-way function that quietly "leaks" the first bit by placing it at a certain index in the output where B can read it and use it to cheat. The same goes for the second bit, third bit, etc., as well as any agreed-upon logical combination of the bits in x. So how do we deal with this? It seems like the main issue here is that we are allowing the one-way function to be created after we decide which bit(s) to use.

This brings up the following question: **Is it possible, given a one-way function (permutation) g, to design a predicate B so that for any string x, the value of B is difficult to predict if we are given only $g(x)$?.**

The answer to this question is $\underline{\text{yes}}$.

1.3 Hard-Core Bits (HCB)

A **hard-core bit** b of a 1WF g is some predicate for which it is as difficult to predict $b(x)$ from $g(x)$ as it is to invert $g(x)$.

In other words, any (probabilistic) polynomial-time adversary can only gain at most a negligible advantage (over the base probability of 1/2) in predicting $b(x)$ from $g(x)$.

More formally, a predicate b is a **hard-core bit** of a one-way function g if:

$$\forall c \in \mathbb{N}, A \in \mathbf{PPT}, \exists N_c \in \mathbb{N} \text{ such that}$$
$$\forall x \text{ randomly generated with } |x| = n > N_c,$$
$$Pr[A(f(x)) = b(x)] < \frac{1}{2} + 1n^c.$$

Suppose B is a hard-core bit of g, and some ADV predicts B from $g(x)$ with probability $\frac{1}{2} + \varepsilon$ where ε is non-negligible. Then, ADV can be used (as a subroutine of some other algorithm) to invert g in polynomial time with non-negligible probability (probability $> 1/(|x|^c)$ for some $c \in \mathbb{N}$).

This is our first look at a reduction. What we are basically saying here is that given a one-way function, inverting it could be reduced to predicting some hard-core bit.

Now, we prove an important theorem about hard-core bits and one-way functions:

1.3.1 (Goldreich, Levin) For any 1WF f, there exists a hardcore bit b of the form

$$b(x) = \langle p, x \rangle := \bigoplus_{i=1}^{|x|} p_i \cdot x_i,$$

where p is some "random" string with $|p| = |x|$.

This theorem is the explanation of the "yes" that we gave above. As long as we pick the 1WF f first, we can always be sure that there is some "random" p such that $b(x) = \langle p, x \rangle$ is a HCB. So, as long as we pick the 1WF before doing anything else, A doesn't have to worry about it secretly "leaking" the secret bit to B before B chooses/generates their own string/bit.

1.3.2 Proof

As with most reductions, we prove this by contradiction/contrapositive. Namely, we do the following:

Suppose, towards a contradiction, that there is some 1WF g for which there does not exist a hard-core bit b of the form $\langle p, x \rangle$. Then, there is some adversary $ADV \in \mathbf{PPT}$ which predicts $b = \langle p, x \rangle$ from $g(x)$, for all p, with probability $Pr = 1/2 + \varepsilon$, where ε is not negligible. Then, we may show that ADV can be used to invert g with non-negligible probability, which would contradict the definition of a 1WF, meaning that our assumption must have been false.

We do not prove the theorem in its entirety, but instead do two simplified versions: one here, where we assume that ADV can predict b with probability 1, and another in the next section, where we assume that ADV can predict b with probability $3/4 + \varepsilon$, where ε is not negligible.

(1) $Pr = 1$.

This one is quite easy, as we have asserted that $ADV(p, g(x)) = \langle p, x\rangle$ 100% of the time. So, if $p_k = 0^{k-1}10^{|x|-k}$, then we know with complete certainty that $ADV(p_k, g(x)) = \langle p_k, x\rangle = x_k$, so we can use each of $p[1], p[2], \ldots, p[|x|]$ to completely determine x from $g(x)$.

1.4 Building Bit Commitment from HCB

For now, we assume the result from Goldreich and Levin, to demonstrate how we may use HCB's to build a commitment protocol. Our two parties A and B follow these steps:

0. We begin with the following shared information: a security parameter k, a one-way function g, and a random string p with $|p| = n$ (which determines a hard-core bit $b(x) = \langle x, p\rangle$ for g.
1. A generates a random string x of length k
2. A generates a random bit b_1
3. A computes $com(b_1) := b_1 \oplus b(x)$
4. A computes $y = g(x)$
5. A commits b_1 by sending y and $com(b_1)$ to B
6. A may open the committed value by sending x to B
7. At this point, B may verify that A did not cheat by checking that $g(x) = y$

Now, we verify the two properties of a commitment protocol:

(1) Hiding: Since $b(x)$ is a hardcore bit, B will not be able to predict it from $g(x)$ with any non-negligible advantage, so B cannot figure out b_1.

(2) Binding: Since $g(x)$ is a one-way function, A will not be able to find a $x' \neq x$ such that $g(x') = g(x)$, as this would require inverting the one-way function. Clearly, A cannot change the values of $com(b_1)$ and y after sending them, and if they cannot find a different x' to make the results work in their favor, there's nothing else they can do once the values have been committed. That is, once A picks x and sends $y, com(b_1)$, the values are bound.

So, we have ourselves a working commitment protocol!