ECS 227 – Modern Cryptography – Fall 96
Lecture 8
Bit Commitment

22 October 1996

Introduction

A bit commitment protocol provides a way for one party to select a bit, transmit that bit to a second party so that the second party has possession but not knowledge of the bit, and finally reveal the bit. The protocol should give the second party a way to verify that the revealed bit is the same as the committed bit.

The following is a bit commitment protocol for the physical world. Party **A** ("the committer") selects a bit, writes it on a piece of paper, and gives the bit to party **B** ("the receiver") in a sealed envelope. Later, the envelope can be opened, and the bit can be revealed. (Note: for the rest of these notes, I'll refer to "the committer" as **A**, and to "the receiver" as **B**.)

$$A \xrightarrow{\boxed{\text{envelope}}} B$$

Inherently, bit commitment involves two steps: a **commit** step, and a **reveal step**. Intuitively, we associate **A** with the commit step, and **B** with the reveal step. In actuality, though, both parties can participate in both steps. For example, either step may involve the exchange of several messages between **A** and **B**.

An Example Application

There are many applications for a good bit commitment protocol. For example, bit commitment can be used to solve the "coin-flipping" problem, which can be described as follows. **A** and **B** are having a heated discussion over the phone. They finally decide to flip a coin to resolve the matter. An unfair way to do this would be for **A** to flip the coin, ask **B** to call it in the air, and then tell **B** if she was right.

A and **B** can better resolve this problem by using bit commitment. First, **B** selects 0 or 1 as the final outcome. Then both **A** and **B** flip a fair coin to select a bit: heads = 0, tails = 1. **A** commits her bit to **A**, and **B** commits her bit to **A**. Then, both parties reveal their bits to each other, and **B** calculates the *exclusive-or* of the 2 bits. If the result matches her initial selection, then she is the winner.

This protocol can be optimized. After **A** commits her bit to **B**, **B** can send her bit to **A** in the clear. Then **A**'s bit can be revealed.

Coin-flipping is considered to be a tough problem, so this demonstrates the power of bit commitment.

3 Key Properties

A good bit commitment protocol should satisfy the following three properties (stated informally):

1) *Validity Condition* - If **A** and **B** are both honest, and if **A** runs the commit protocol for a bit b, and the reveal protocol produces the bit b', then $b = b'$.
2) **B** should not have any idea about what the committed bit is until it is revealed.
3) **A** should not be able to change her commitment after the commit step.

Computational Power

We will consider the case in which **A** has unbounded computational power, and **B** has only bounded computational power.

A First Attempt at a Protocol

Consider the following protocol, pictured below:

$$\mathrm{A} \xrightarrow[\textit{commit}]{\varepsilon_{key}(b \parallel key)} \mathrm{B}$$

$$\mathrm{A} \xrightarrow[\textit{reveal}]{\text{key}} \mathrm{B}$$

Commit Step: **A** encrypts a string that consists of the committed bit b concatenated with the encryption key, and sends the result to **B**.

Reveal Step: **A** sends the key to **B**.

The problem with this protocol is that **A** might be able to find two keys **key** and **key**$'$ such that **key** $\neq$ **key**$'$ and yet $\varepsilon_{key}(b \parallel key) = \varepsilon_{key'}(b' \parallel key')$.

(Note: this may occur with a perfectly good encryption scheme.) Thus, **A** can cheat. In the reveal step, **A** can choose which key to send, thereby determining which bit is revealed. Clearly, this violates one of the properties described earlier, so this protocol is not acceptable.

Pseudorandom Generators (PRG's)

A good bit commitment protocol can be developed using a pseudorandom generator (PRG). Let's first take a quick look at PRG's in general.

Recall the definition of breaking a PRG. Consider a PRG $G : \Sigma^k \rightarrow \Sigma^n$ that generates a string of n bits, given a string of k bits, where $n \geq k$. An Adversary A (t, ϵ)-*breaks* G if:

$Pr[s \leftarrow \Sigma^k; A(G(s) = 1] - Pr[r \leftarrow \Sigma^n; A(r) = 1] \geq \epsilon$; and

A runs in $\leq t$ steps.

The definition formalizes the notion that a PRG is good if an adversary cannot distinguish a pseudorandom string from a random string very well.

How to Make a PRG out of a PRF

A PRG G can be constructed using a PRF such as DES. We can define G as follows:

$G(s) = DES_s(0) \parallel DES_s(1) \parallel ... \parallel DES_s(2^{64} - 1)$

So G is calculated by applying the PRF to fixed strings and then concatenating the results together.

Using a PRG for Bit Commitment: A First Attempt

Consider the bit commitment protocol pictured below. G is a PRG, s is a random seed of k bits. The notation $G^{(i)}(s)$ refers to the ith bit of $G(s)$, and $G^{(\leq m)}(s)$ refers to the first m bits of $G(s)$.

$$\mathrm{A} \xrightarrow[\textit{commit}]{\quad G^{(\leq m)}(s) \parallel (b \oplus G^{(m+1)}(s)) \quad} \mathrm{B}$$

$$\mathrm{A} \xrightarrow[\textit{reveal}]{\qquad\qquad \mathrm{s} \qquad\qquad} \mathrm{B}$$

After the reveal step, **B** first calculates $G^{(\leq m)}(s)$ and checks to make sure that it is the same as the first m bits of **A**'s commit message. (If not then **A** has cheated.) The last bit of **A**'s commit message is $(b \oplus G^{(m+1)}(s))$. Let's call that δ^*. **B** can recover the bit b by calculating: $b = \delta^* \oplus G^{(m+1)}(s)$.

The problem with this protocol is similar to the problem with the protocol discussed before. **A** can cheat by finding an s and an s' such that:

- $s \neq s'$;
- $G^{(\leq m)}(s) = G^{(\leq m)}(s')$; and
- $G^{(m+1)}(s)$ is the complement of $G^{(m+1)}(s')$.

If **A** can find such a pair, then **A** can choose to send either s or s' to **B** in the reveal step, and thereby determine which bit is finally revealed.

Naor to the Rescue!

Moni Naor proposed the following protocol:

commit step:

$$\mathrm{A} \xleftarrow{\quad r_1 ... r_{3k} \quad} \mathrm{B}$$

$$\mathrm{A} \xrightarrow{\quad d_1 ... d_{3k} \quad} \mathrm{B}$$

reveal step:

$$\mathrm{A} \xrightarrow{\qquad s \qquad} \mathrm{B}$$

Commit Step: **B** sends a random string r of $3k$ bits to **A**. **A** selects a random string s of k bits.

Then **A** uses r, s, and a PRG $G : \Sigma^k \rightarrow \Sigma^{3k}$ to construct a string d of $3k$ bits. **A** sends d to **B**. The construction rule for d is:

For each i in $[1...3k]$,
$d_i = G^{(i)}(s)$ if $r_i = 0$; and
$d_i = G^{(i)}(s) \oplus b$ if $r_i = 1$.

Reveal Step: **A** sends s to **B**. **B** should then compute $G(s)$, and verify that the bits of $G(s)$ that correspond to 0's in r match the bits of d that correspond to 0's in r. Also, **B** should verify that the bits of d that correspond to 1's in r are consistent with $G(s)$ and a particular choice of b.

Proof that the Naor Protocol Works

Claim 1: **B** learns nothing computationally about b from the message it receives from **A** in the commit step.

For a proof, please see the Naor paper. The gist of the argument is that if **B** *can* learn anything about b, then B can distinguish between a pseudorandom string and a random string.

Claim 2: **A** cannot cheat.

To see this, note that **A** can cheat only if there exist an s and an s' such that:

- $G^{(i)}(s) = G^{(i)}(s')$ for all i in $[1..3k]$ such that $r_i = 0$; and
- $G^{(i)}(s) \neq G^{(i)}(s')$ for all i in $[1..3k]$ such that $r_i = 1$.

Then **A** can cheat in the same way as described before: she can send either s or s' in the reveal step, and thereby determine which bit is revealed.

We say that such an (s, s') pair *fools* the string r that **B** first sends to **A**. Observe that each (s, s') pair fools exactly one r.

Since both s and s' have k bits, there are 2^k unique s's, and 2^k unique s's, so there are $2^k \times 2^k = 2^{2k}$ different (s, s') pairs. Actually, there are less than 2^{2k} such pairs, because in this calculation we have counted some of the pairs twice. Since each unique (s, s') pair fools exactly one r, it follows that there are less than 2^{2k} r's which can be fooled.

But, since r has $3k$ bits, there are 2^{3k} different r's. Thus, the probability that a random r is fooled by an (s, s') pair is $< \frac{2^{2k}}{2^{3k}} = 2^{-k}$. So 2^{-k} is an upper bound on the probability that **A** will be able to cheat, and unless k is small, that probability is negligible.