# CPSC 467b: Cryptography and Computer Security

## Lecture 23

Michael J. Fischer

Department of Computer Science
Yale University

April 14, 2010

1. Secret Splitting (continued)
   - Secret Splitting with Dishonest Parties

2. Bit Commitment Problem
   - Bit Commitment Using Symmetric Cryptography
   - Bit Commitment Using Hash Functions
   - Bit Commitment Using Pseudorandom Sequence Generators
   - Formalization of Bit Commitment Schemes

# Secret Splitting

# Secret splitting with semi-honest parties

Shamir's scheme is an example of a protocol that works assuming *semi-honest* parties.

These are players that follow the protocol but additionally may collude in an attempt to discover secret information.

We just saw that no coalition of fewer than $\tau$ players could learn anything about the dealer's secret, even if they pooled all of their shares.

# Secret splitting with dishonest dealer

In practice, either the dealer or some of the players (or both) may be dishonest and fail to follow the protocol. The honest players would like some guarantees even in such situations.

A dishonest dealer can always lie about the true value of her secret. Even so, the honest players want assurance that their shares do in fact encode a unique secret, that is, all sets of $\tau$ shares reconstruct the same secret $s$.

Shamir's $(\tau, k)$ threshold scheme assumes that all shares lie on a single polynomial of degree at most $k - 1$.

This might not hold if the dealer is dishonest and gives bad shares to some of the players.

The players have no way to discover that they have bad shares until later when they try to reconstruct $s$.

# Verifiable secret sharing

In *verifiable secret sharing*, the sharing phase is an active protocol involving the dealer and all of the players.

At the end of this phase, either the dealer is exposed as being dishonest, or all of the players end up with shares that are consistent with a single secret.

Needless to say, protocols for verifiable secret sharing are quite complicated.

# Dishonest players

*Dishonest players* present another kind of problem. These are players that fail to follow the protocol. During the reconstruction phase, they may fail to supply their share, or they may present a (possibly different) corrupted share to each other player.

With Shamir's scheme, a share that just disappears does not prevent the secret from being reconstructed, as long as enough valid shares remain.

But a player who lies about his share during the reconstruction phase can cause other players to reconstruct incorrect values for the secret.

# Fault-tolerance in secret sharing protocols

A *fault-tolerant secret sharing scheme* should allow the secret to be correctly reconstructed, even in the face of a certain number of corrupted shares.

Of course, it may be desirable to have schemes that can tolerate dishonesty in both dealer and a limited number of players.

The interested reader is encouraged to explore the extensive literature on this subject.

# Bit Commitment Problem

# Bit guessing game

Alice and Bob want to play a guessing game over the internet.

Alice says,

> *"I'm thinking of a bit. If you guess my bit correctly, I'll give you $10. If you guess wrong, you give me $10."*

Bob says,

> *"Ok, I guess zero."*

Alice replies,

> *"Sorry, you lose. I was thinking of one."*

# Preventing Alice from changing her mind

While this game may seem fair on the surface, there is nothing to prevent Alice from changing her mind after Bob makes his guess.

Even if Alice and Bob play the game face to face, they still must do something to *commit* Alice to her bit before Bob makes his guess.

For example, Alice might be required to write her bit down on a piece of paper and seal it in an envelope.

After Bob makes his guess, he opens the envelope to know whether he won or lost.

Writing down the bit commits Alice to that bit, even though Bob doesn't learn its value until later.

# Bit-commitment problem

A *bit commitment* or *blob* or *cryptographic envelope* is an electronic analog of a sealed envelope.

Intuitively, a blob has two properties:

1. The bit inside the blob remains hidden until the blob is opened.
2. The bit inside the blob cannot be changed, that is, blob cannot be opened in different ways to reveal different bits.

A blob is produced by a protocol **commit**($b$) between Alice and Bob. We assume initially that only Alice knows $b$.

At the end of the commit protocol, Bob has a blob $c$ containing Alice's bit $b$, but he should have no information about $b$'s value.

Later, Alice and Bob can run a protocol **open**($c$) to reveal the bit contained in $c$ to Bob.

# Requirements for bit commitment

Alice and Bob do not trust each other, so each wants protection from cheating by the other.

- Alice wants to be sure that Bob cannot learn $b$ after running **commit**($b$), even if he cheats.
- Bob wants to be sure that all successful runs of **open**($c$) reveal the same bit $b'$, no matter what Alice does.

# Requirements for bit commitment

Alice and Bob do not trust each other, so each wants protection from cheating by the other.

- Alice wants to be sure that Bob cannot learn $b$ after running **commit**($b$), even if he cheats.
- Bob wants to be sure that all successful runs of **open**($c$) reveal the same bit $b'$, no matter what Alice does.

Note that we do *not* require that Alice tell the truth about her private bit $b$. A dishonest Alice can always pretend her bit was $b' \neq b$ when producing $c$. But if she does, $c$ can only be opened to $b'$, not to $b$.

These ideas should become clearer in the protocols below.

# Bit Commitment Using Symmetric Cryptography

# A naïve approach to building a bit-commitment scheme

A naïve way to use a symmetric cryptosystem for bit commitment is for Alice to encrypt $b$ with a private key $k$ to get blob $c = E_k(b)$.

She opens it by releasing $k$. Anyone can compute $b = D_k(c)$.

Alice can easily cheat if she can find a *colliding triple* $(c, k_0, k_1)$ with the property that $D_{k_0}(c) = 0$ and $D_{k_1}(c) = 1$.

She "commits" by sending $c$ to Bob.

Later, she can choose to send Bob either $k_0$ or $k_1$.

This isn't just a hypothetical problem. Suppose Alice uses the most secure cryptosystem of all, a one-time pad, so $D_k(c) = c \oplus k$.

Then $(c, c \oplus 0, c \oplus 1)$ is a colliding triple.

# Another attempt

The protocol below tries to make it harder for Alice to cheat by making it possible for Bob to detect most bad keys.

| | Alice | | Bob |
|---|---|---|---|
| | To **commit**$(b)$: | | |
| 1. | | $\xleftarrow{r}$ | Choose random string $r$. |
| 2. | Choose random key $k$. | | |
| | Compute $c = E_k(r \cdot b)$. | $\xrightarrow{c}$ | $c$ is commitment. |
| | To **open**$(c)$: | | |
| 3. | Send $k$. | $\xrightarrow{k}$ | Let $r' \cdot b' = D_k(c)$.<br>Check $r' = r$.<br>$b'$ is revealed bit. |

# Security of second attempt

For many cryptosystems (e.g., DES), this protocol does indeed prevent Alice from cheating, for she will have difficulty finding any two keys $k_0$ and $k_1$ such that $E_{k_0}(r \cdot 0) = E_{k_1}(r \cdot 1)$, and $r$ is different for each run of the protocol.

However, for the one-time pad, she can cheat as before: She just takes $c$ to be random and lets $k_0 = c \oplus (r \cdot 0)$ and $k_1 = c \oplus (r \cdot 1)$.

Then $D_{k_b}(c) = r \cdot b$ for $b \in \{0, 1\}$, so the revealed bit is 0 or 1 depending on whether Alice sends $k_0$ or $k_1$ in step 3.

# Need for a different approach

We see that not all secure cryptosystems have the properties we need in order to make the protocol secure.

We need a property analogous to the strong collision-free property for hash functions (Lecture 15).

# Bit Commitment Using Hash Functions

# Bit commitment from a hash function

The analogy between bit commitment and hash functions described above suggests a bit commitment scheme based on hash functions.

| | Alice | | Bob |
|---|---|---|---|
| | To **commit**$(b)$: | | |
| 1. | | $\xleftarrow{r_1}$ | Choose random string $r_1$. |
| 2. | Choose random string $r_2$. | | |
| | Compute $c = H(r_1 r_2 b)$. | $\xrightarrow{c}$ | $c$ is commitment. |
| | To **open**$(c)$: | | |
| 3. | Send $r_2$. | $\xrightarrow{r_2}$ | Find $b' \in \{0,1\}$ such that $c = H(r_1 r_2 b')$. If no such $b'$, then fail. Otherwise, $b'$ is revealed bit. |

# Purpose of $r_2$

The purpose of $r_2$ is to protect Alice's secret bit $b$.

To find $b$ before Alice opens the commitment, Bob would have to find $r_2'$ and $b'$ such that $H(r_1 r_2' b') = c$.

This is akin to the problem of inverting $H$ and is likely to be hard, although the one-way property for $H$ is not strong enough to imply this.

On the one hand, if Bob succeeds in finding such $r_2'$ and $b'$, he has indeed inverted $H$, but he does so only with the help of $r_1$—information that is not generally available when attempting to invert $H$.

# Purpose of $r_1$

The purpose of $r_1$ is to strengthen the protection that Bob gets from the hash properties of $H$.

Even without $r_1$, the strong collision-free property of $H$ would imply that Alice cannot find $c$, $r_2$, and $r_2'$ such that $H(r_2 0) = c = H(r_2' 1)$.

But by using $r_1$, Alice would have to find a new colliding pair for each run of the protocol.

This protects Bob by preventing Alice from exploiting a few colliding pairs for $H$ that she might happen to discover.

# Bit Commitment Using Pseudorandom Sequence Generators

# Bit commitment using a PSRG

Let $G_\rho(s)$ be the first $\rho$ bits of $G(s)$. ($\rho$ is a security parameter.)

| | Alice | | Bob |
|---|---|---|---|
| | To **commit**$(b)$: | | |
| 1. | | $\xleftarrow{r}$ | Choose random $r \in \{0,1\}^\rho$. |
| 2. | Choose random seed $s$. | | |
| | Let $y = G_\rho(s)$. | | |
| | If $b = 0$ let $c = y$. | | |
| | If $b = 1$ let $c = y \oplus r$. | $\xrightarrow{c}$ | $c$ is commitment. |
| | To **open**$(c)$: | | |
| 3. | Send $s$. | $\xrightarrow{s}$ | Let $y = G_\rho(s)$. |
| | | | If $c = y$ then reveal 0. |
| | | | If $c = y \oplus r$ then reveal 1. |
| | | | Otherwise, fail. |

# Security of PSRG bit commitment

Assuming $G$ is cryptographically strong, then $c$ will look random to Bob, regardless of the value of $b$, so he will be unable to get any information about $b$.

# Security of PSRG bit commitment

Assuming $G$ is cryptographically strong, then $c$ will look random to Bob, regardless of the value of $b$, so he will be unable to get any information about $b$.

Why?

# Security of PSRG bit commitment

Assuming $G$ is cryptographically strong, then $c$ will look random to Bob, regardless of the value of $b$, so he will be unable to get any information about $b$.

Assume Bob has advantage $\epsilon$ at guessing $b$ when he can choose $x$ and is given $c$. Here's a judge $J$ for distinguishing $G(S)$ from $U$.

- Given input $y$, $J$ chooses random $b$ and simulates Bob's cheating algorithm. $J$ simulates Bob choosing $r$, computes $c = y \oplus r^b$, and continues Bob's algorithm to find a guess $\hat{b}$ for $b$.
- If $\hat{b} = b$, $J$ outputs 1.
- If $\hat{b} \neq b$, $J$ outputs 0.

# The judge's advantage

If $y$ is drawn at random from $U$, then $c$ is uniformly distributed and independent of $b$, so $J$ outputs 1 with probability $1/2$.

If $y$ comes from $G(S)$, then $J$ outputs 1 with the same probability that Bob can correctly guess $b$.

Assuming $G$ is cryptographically strong, then Bob has negligible advantage at guessing $b$.

# Purpose of $r$

The purpose of $r$ is to protect Bob against a cheating Alice.

Alice can cheat if she can find a triple $(c, s_0, s_1)$ such that $s_0$ opens $c$ to reveal 0 and $s_1$ opens $c$ to reveal 1.

Such a triple must satisfy the following pair of equations:

$$\left.\begin{array}{rcl} c & = & G_\rho(s_0) \\ c & = & G_\rho(s_1) \oplus r. \end{array}\right\}$$

It is sufficient for her to solve the equation

$$r = G_\rho(s_0) \oplus G_\rho(s_1)$$

for $s_0$ and $s_1$ and then choose $c = G_\rho(s_0)$.

# How big does $\rho$ need to be?

We now count the number of values of $r$ for which the equation

$$r = G_\rho(s_0) \oplus G_\rho(s_1)$$

has a solution.

Suppose $n$ is the seed length, so the number of seeds is $\leq 2^n$. Then the right side of the equation can assume at most $2^{2n}/2$ distinct values.

Among the $2^\rho$ possible values for $r$, only $2^{2n-1}$ of them have the possibility of a colliding triple, regardless of whether or not Alice can feasibly find it.

Hence, by choosing $\rho$ sufficiently much larger than $2n-1$, the probability of Alice cheating can be made arbitrarily small.

For example, if $\rho = 2n + 19$ then her probability of successful cheating is at most $2^{-20}$.

# Why does Bob need to choose $r$?

Why can't Alice choose $r$, or why can't $r$ be fixed to some constant?

If Alice chooses $r$, then she can easily solve $r = G_\rho(s_0) \oplus G_\rho(s_1)$ and cheat.

If $r$ is fixed to a constant, then if Alice ever finds a colliding triple $(c, s_0, s_1)$, she can fool Bob every time.

While finding such a pair would be difficult if $G_\rho$ were a truly random function, any specific PRSG might have special properties, at least for a few seeds, that would make this possible.

# Example

For example, suppose $r = 1^\rho$ and $G_\rho(\neg s_0) = \neg G_\rho(s_0)$ for some $s_0$.

Then taking $s_1 = \neg s_0$ gives
$G_\rho(s_0) \oplus G_\rho(s_1) = G_\rho(s_0) \oplus G_\rho(\neg s_0) = G_\rho(s_0) \oplus \neg G_\rho(s_0) = 1^\rho = r$.

By having Bob choose $r$ at random, $r$ will be different each time (with very high probability).

A successful cheating Alice would be forced to solve
$r = G_\rho(s_0) \oplus G_\rho(s_1)$ in general, not just for one special case.

# Formalization of Bit Commitment Schemes

# Formalization of bit commitment schemes

The three bit commitment protocols of the previous section all have the same form.

We abstract from these protocols a cryptographic primitive, called a *bit commitment scheme*, which consists of a pair of *key spaces* $\mathcal{K}_\mathcal{A}$ and $\mathcal{K}_\mathcal{B}$, a *blob space* $\mathcal{B}$, a *commitment* function

$$\textbf{enclose} : \mathcal{K}_\mathcal{A} \times \mathcal{K}_\mathcal{B} \times \{0,1\} \to \mathcal{B},$$

and an *opening* function

$$\textbf{reveal} : \mathcal{K}_\mathcal{A} \times \mathcal{K}_\mathcal{B} \times \mathcal{B} \to \{0,1,\phi\},$$

where $\phi$ means "failure".

We say that a blob $c \in \mathcal{B}$ *contains* $b \in \{0,1\}$ if $\textbf{reveal}(k_A, k_B, c) = b$ for some $k_A \in \mathcal{K}_A$ and $k_B \in \mathcal{K}_B$.

# Desired properties

These functions have three properties:

1. $\forall k_A \in \mathcal{K}_A, \forall k_B \in \mathcal{K}_B, \forall b \in \{0,1\}$, **reveal**$(k_A, k_B,$ **enclose**$(k_A, k_B, b)) = b$;
2. $\forall k_B \in \mathcal{K}_B, \forall c \in \mathcal{B}, \exists b \in \{0,1\}, \forall k_A \in \mathcal{K}_A$, **reveal**$(k_A, k_B, c) \in \{b, \phi\}$.
3. No feasible probabilistic algorithm that attempts to distinguish blobs containing 0 from those containing 1, given $k_B$ and $c$, is correct with probability significantly greater than $1/2$.

# Intuition

The intention is that $k_A$ is chosen by Alice and $k_B$ by Bob. Intuitively, these conditions say:

1. Any bit $b$ can be committed using any key pair $k_A, k_B$, and the same key pair will open the blob to reveal $b$.
2. For each $k_B$, all $k_A$ that successfully open $c$ reveal the same bit.
3. Without knowing $k_A$, the blob does not reveal any significant amount of information about the bit it contains, even when $k_B$ is known.

# Comparison with symmetric cryptosystem

A bit commitment scheme looks a lot like a symmetric cryptosystem, with **enclose**$(k_A, k_B, b)$ playing the role of the encryption function and **reveal**$(k_A, k_B, c)$ the role of the decryption function.

However, they differ both in their properties and in the environments in which they are used.

Conventional cryptosystems do not require uniqueness condition 2, nor do they necessarily satisfy it.

# Comparison with symmetric cryptosystem (cont.)

In a conventional cryptosystem, we assume that Alice and Bob trust each other and both share a secret key $k$.

The cryptosystem is designed to protect Alice's secret message from a passive eavesdropper Eve.

In a bit commitment scheme, Alice and Bob cooperate in the protocol but do not trust each other to choose the key.

Rather, the key is split into two pieces, $k_A$ and $k_B$, with each participant controlling one piece.

# A bit-commitment protocol from a bit-commitment scheme

A bit commitment scheme can be turned into a bit commitment protocol by plugging it into the *generic protocol*:

| | Alice | | Bob |
|---|---|---|---|
| | To **commit**$(b)$: | | |
| 1. | | $\xleftarrow{k_B}$ | Choose random $k_B \in \mathcal{K}_B$. |
| 2. | Choose random $k_A \in \mathcal{K}_A$. | | |
| | $c = \textbf{enclose}(k_A, k_B, b)$. | $\xrightarrow{c}$ | $c$ is commitment. |
| | To **open**$(c)$: | | |
| 3. | Send $k_A$. | $\xrightarrow{k_A}$ | Compute $b = \textbf{reveal}(k_A, k_B, c)$.<br>If $b = \phi$, then fail.<br>If $b \neq \phi$, then $b$ is revealed bit. |

The previous bit commitment protocols we have presented can all be regarded as instances of the generic protocol.

For example, we get the second protocol based on symmetric cryptography by taking

$$\textbf{enclose}(k_A, k_B, b) = E_{k_A}(k_B \cdot b),$$

and

$$\textbf{reveal}(k_A, k_B, c) = \begin{cases} b & \text{if } k_B \cdot b = D_{k_A}(c) \\ \phi & \text{otherwise.} \end{cases}$$