CPSC 367: Cryptography and Security

Michael J. Fischer

Lecture 22
April 16, 2019

Mutual Privacy-Preserving Protocols

Commitment Schemes

- Bit Commitment Using QR Cryptosystem
- Bit Commitment Using Hash Functions
- Bit Commitment Using Pseudorandom Sequence Generators

Coin-Flipping

Locked Boxes

- Locked Box Paradigm

Oblivious Transfer

Mutual Privacy-Preserving Protocols

Privacy

We have looked at several protocols that are intended to keep Alice's information secret, both from Bob and from a malicious adversary masquarding as Bob.

We now look at other protocols whose goal is to control the release of partial information about Alice's secret. Just enough information should be released to carry out the purpose of the protocol but no more.

We'll see several examples in the following sections.

Commitment Schemes

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.

Commitment scheme

A *commitment scheme* is an encryption c of a secret s using a cryptosystem with a special property.

1. s cannot be found from c by anyone not knowing the secret key.
2. All keys k' that decrypt c to a valid secret reveal the same secret.

Thus, if $c = E_k(s)$:

- It is hard to find s from c without knowning k.
- For every k', s', if $E_{k'}(s') = c$, then $s = s'$.

Commitment intuition

In other words,

- If Alice produces a commitment c to a secreta s, then s cannot be recovered from c without knowing Alice's secret encoding key k.
- There is no key k' that Alice might release that would make it appear that c is a commitment of some other secret $s' \neq s$.

Commitments as cryptographic envelopes

More formally, a *commitment* or *blob* or *cryptographic envelope* is an electronic analog of a sealed envelope.

Intuitively, a blob has two properties:

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

Commitment primitives

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

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

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

Requirements for secret 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 s after she runs **commit**(s), even if he cheats.
- Bob wants to be sure that all successful runs of **open**(c) reveal the same secret s', no matter what Alice does.

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

These ideas should become clearer in the protocols below.

Bit Commitment Using QR Cryptosystem

A simple (but inefficient) bit commitment scheme

Here's a simple way for Alice to commit to a secret that is a single bit b.

1. Create a Goldwasser-Micali public key $e = (n, y)$, where $n = pq$.
2. Choose random $r \in \mathbf{Z}_n^*$, and use it to produce an encryption c of b. (See lecture 21.)
3. Send the blob (e, c) to Bob.

To open (e, c), Alice sends b, p, q, r to Bob.

Bob checks that $n = pq$, p and q are distinct odd primes, $y \in Q_n^{00}$, and that c is the encryption of b based on r.

Security of QR bit commitment

Alice can't change her mind about b, since c either is or is not a quadratic residue.

Bob cannot determine the value of b before Alice opens c since that would amount to violating the quadratic residuosity assumption.

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

Cryptographically strong PRSG

Recall that a PseudoRandom Sequence Generator (PRSG) is a function G that maps short seeds to long pseudorandom output sequences.

G is *cryptographically strong* if the behavior of every probabilistic polynomial time algorithm J (the "judge") is essentially the same, whether J's source of random bits is a sequence of coin flips from a fair coin or from the bits of the sequence $G(s)$ that results from a truly random seed s.

We can build a bit-commitment scheme using such a G.

Bit commitment using a PRSG

Let $G_\rho(s)$ be the first ρ bits of $G(s)$. (ρ 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.

Why this protocol works

We argue informally why this is a proper bit-commitment protocol.

- If Bob is able to get an advantage at predicting b knowing only c and r, then he has an advantage at predicting the first ρ bits of $G(s)$. This contradicts the assumption that G is cryptographically strong.
- If Alice can open c to reveal either 0 or 1, then she has found two seeds s_0 and s_1 such that $G_\rho(s_0) \oplus G_\rho(s_1) = r$. This is not possible for most values of r if ρ is sufficiently large.

Details are in the appendix.

Coin-Flipping

Flipping a common coin

Alice and Bob are in the process of getting a divorce and are trying to decide who gets custody of their pet cat, Fluffy.

They both want the cat, so they agree to decide by flipping a coin: heads Alice wins; tails Bob wins.

Bob has already moved out and does not wish to be in the same room with Alice.

The feeling is mutual, so Alice proposes that she flip the coin and telephone Bob with the result.

This proposal of course is not acceptable to Bob since he has no way of knowing whether Alice is telling the truth when she says that the coin landed heads.

Making it fair

"Look Alice," he says, "to be fair, we both have to be involved in flipping the coin."

"We'll each flip a private coin and XOR our two coins together to determine who gets Fluffy."

"You should be happy with this arrangement since even if you don't trust me to flip fairly, your own fair coin is sufficient to ensure that the XOR is unbiased."

A proposed protocol

This sounds reasonable to Alice, so she lets him propose the protocol below, where 1 means "heads" and 0 means "tails".

	Alice		Bob
1.	Choose random bit $b_A \in \{0,1\}$	$\xrightarrow{b_A}$.	
2.		$\xleftarrow{b_B}$	Choose random bit $b_B \in \{0,1\}$.
3.	Coin outcome is $b = b_A \oplus b_B$.		Coin outcome is $b = b_A \oplus b_B$.

Alice considers this for awhile, then objects.

> *"This isn't fair. You get to see my coin before I see yours, so now you have complete control over the outcome."*

Alice's counter proposal

She suggests that she would be happy if the first two steps were reversed, so that Bob flips his coin first, but Bob balks at that suggestion.

They then both remember about blobs and decide to use blobs to prevent either party from controlling the outcome. They agree on the following protocol.

A mutually acceptable protocol

Alice		Bob
1. Choose random $k_A, s_A \in \mathcal{K}_A$.	$\xleftrightarrow{k_A,\ k_B}$	Choose random $k_B, s_B \in \mathcal{K}_B$.
2. Choose random bit $b_A \in \{0,1\}$.		Choose random bit $b_B \in \{0,1\}$.
$c_A = \textbf{enclose}(s_A, k_B, b_A)$.	$\xleftrightarrow{c_A,\ c_B}$	$c_B = \textbf{enclose}(s_B, k_A, b_B)$.
3. Send s_A.	$\xleftrightarrow{s_A,\ s_B}$	Send s_B.
4. $b_B = \textbf{reveal}(s_B, k_A, c_B)$.		$b_A = \textbf{reveal}(s_A, k_B, c_A)$.
Coin outcome is $b = b_A \oplus b_B$.		Coin outcome is $b = b_A \oplus b_B$.

At the completion of step 2, both Alice and Bob have each others' commitment (something they failed to achieve in the past, which is why they're in the middle of a divorce now), but neither knows the other's private bit.

They learn each other's bit at the completion of steps 3 and 4.

Remaining asymmetry

While this protocol appears to be completely symmetric, it really isn't quite, for one of the parties completes step 3 before the other one does.

Say Alice receives s_B before sending s_A.

At that point, she can compute b_B and hence know the coin outcome b.

If it turns out that she lost, she might decide to stop the protocol and refuse to complete her part of step 3.

Premature termination

What happens if one party quits in the middle or detects the other party cheating?

So far, we've only considered the possibility of undetected cheating.

But in any real situation, one party might feel that he or she stands to gain by cheating, *even if the cheating is detected*.

Responses to cheating

Detected cheating raises complicated questions as to what happens next.

- Does a third party Carol become involved?
- If so, can Bob prove to Carol that Alice cheated?
- What if Alice refuses to talk to Carol?

Think about Bob's recourse in similar real-life situations and consider the reasons why such situations rarely arise.

For example, what happens if someone

- fails to follow the provisions of a contract?
- ignores a summons to appear in court?

A copycat attack

There is a subtle problem with the previous coin-flipping protocol.

Suppose Bob sends his message before Alice sends hers in each of steps 1, 2, and 3.

Then Alice can choose $k_A = k_B$, $c_A = c_B$, and $s_A = s_B$ rather than following her proper protocol, so

$$\textbf{reveal}(s_A, k_B, c_A) = \textbf{reveal}(s_B, k_A, c_B).$$

In step 4, Bob will compute $b_A = b_B$ and won't detect that anything is wrong. The coin outcome is $b = b_A \oplus b_A = 0$. Hence, Alice can force outcome 0 simply by playing copycat.

Copycat attacks are not so easy to prevent.

Locked Boxes

Locked boxes

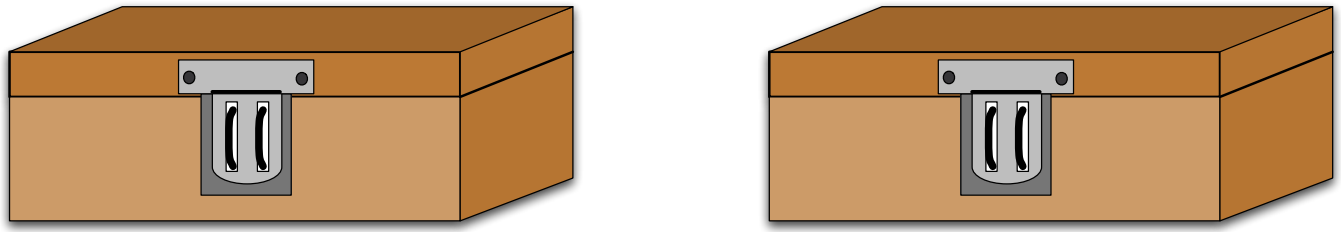
Protocols for coin flipping and for dealing a poker hand from a deck of cards can be based on the intuitive notion of locked boxes.

This idea in turn can be implemented using commutative-key cryptosystems.

We first present a coin-flipping protocol using locked boxes.

Preparing the boxes

Imagine two sturdy boxes with hinged lids that can be locked with a padlock.



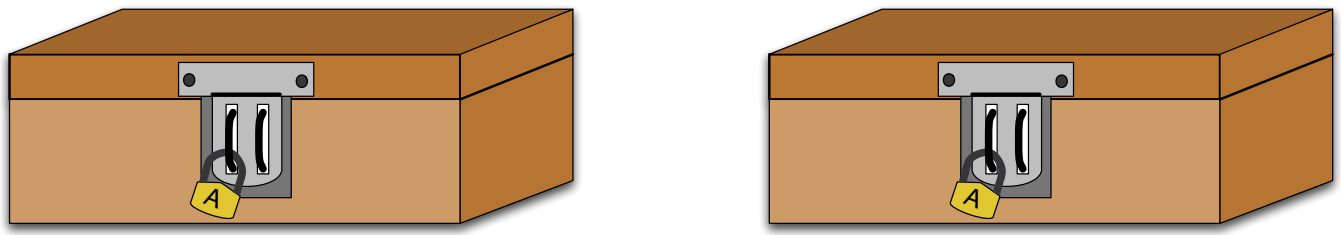
Alice writes "heads" on a slip of paper and "tails" on another.

"heads", signed Alice	"tails", signed Alice

She places one of these slips in each box.

Alice locks the boxes

Alice puts a padlock on each box for which she holds the only key.

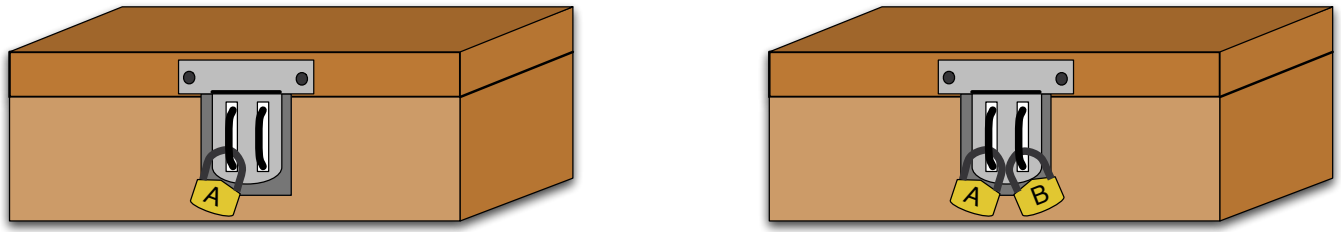


She then gives both locked boxes to Bob, in some random order.

Bob adds his lock

Bob cannot open the boxes and does not know which box contains "heads" and which contains "tails".

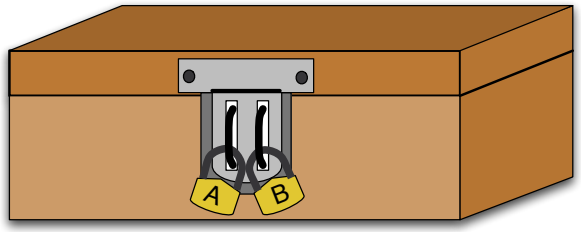
He chooses one of the boxes and locks it with his own padlock, for which he has the only key.



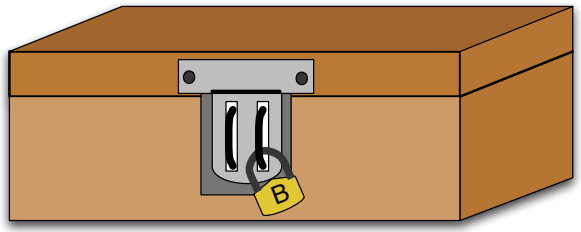
He gives the doubly-locked box back to Alice.

Alice removes her lock

Alice gets



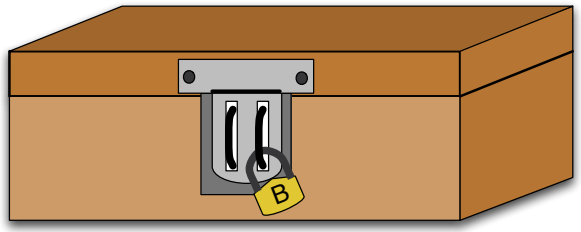
She removes her lock.



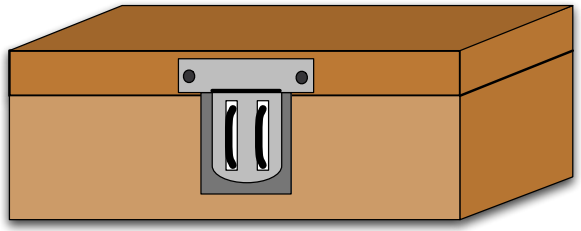
and returns the box to Bob.

Bob opens the box

Bob gets



He removes his lock



opens the box, and removes the slip of paper from inside.

"heads", signed Alice

He gives the slip to Alice.

Alice checks that Bob didn't cheat

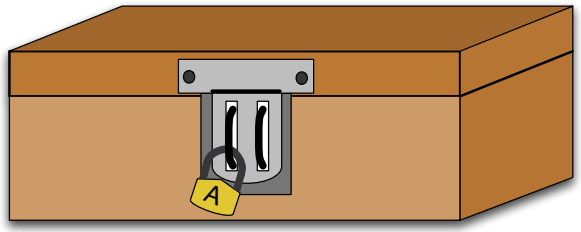
At this point, both Alice and Bob know the outcome of the coin toss.

Alice verifies that the slip of paper is one of the two that she prepared at the beginning, with her handwriting on it.

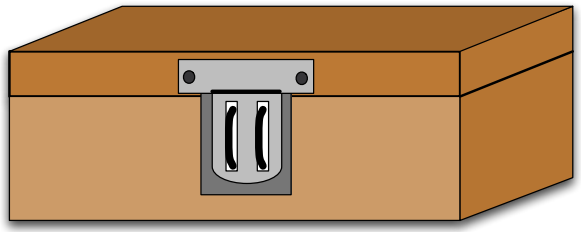
She sends her key to Bob.

Bob checks that Alice didn't cheat

Bob still has the other box.



He removes Alice's lock,



opens the box, and removes the slip of paper from inside.

"tails", signed Alice

He checks that it contains the other coin value.

Card dealing using locked boxes

The same locked box paradigm used for coin flipping can be used for dealing a 5-card poker hand from a deck of cards.

1. Alice takes a deck of cards, places each card in a separate box, and locks each box with her lock.
2. She arranges the boxes in random order and ships them off to Bob.
3. Bob picks five boxes, locks each with his lock, and send them back to Alice.
4. Alice removes her locks from those five boxes and returns them to Bob.
5. Bob unlocks them and obtains the five cards of his poker hand.

Further details are left to the reader.

Oblivious Transfer

Generalization of coin-flipping protocol

In the locked box coin-flipping protocol, Alice has two messages m_0 and m_1.

Bob gets one of them.

Alice doesn't know which (until Bob tells her).

Bob can't cheat to get both messages.

Alice can't cheat to learn which message Bob got.

The *oblivious transfer problem* abstracts these properties from particular applications such as coin flipping and card dealing.

One-of-two oblivious transfer

In *one-of-two oblivious transfer* (OT_1^2), Alice has two secrets, s_0 and s_1.

Bob gets exactly one of the secrets, each with probability $1/2$.

Alice does not know which one Bob gets.

The locked box protocol is one way to implement one-of-two oblivious transfer.

Strengthening one-of-two oblivious transfer

A slightly stronger version of OT_1^2, useful in secure multiparty computation, lets Bob choose which of Alice's secrets he gets, but his choice is hidden from Alice.

Upon completion, Bob has the secret he requested, but Alice has no idea which one he got.

This is just a brief overview of some of the more advanced techniques that are being used in advanced cryptographic protocols.

Appendix: Security of PRSG 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?

Assume Bob has advantage ϵ at guessing b when he can choose r 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 ρ 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^ρ 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 ρ 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_ρ 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.