CPSC 467b: Cryptography and Computer Security

Michael J. Fischer

Lecture 20
April 2, 2012

Formalization of Bit Commitment Schemes

Formalization of bit commitment schemes

The three bit commitment protocols presented last time 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 ϕ 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. $\forall k_A \in \mathcal{K}_A, \forall k_B \in \mathcal{K}_B, \forall b \in \{0,1\}$,
   $\textbf{reveal}(k_A, k_B, \textbf{enclose}(k_A, k_B, b)) = b$;
2. $\forall k_B \in \mathcal{K}_B, \forall c \in \mathcal{B}, \exists b \in \{0,1\}, \forall k_A \in \mathcal{K}_A$,
   $\textbf{reveal}(k_A, k_B, c) \in \{b, \phi\}$.
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 $b = \textbf{reveal}(k_A, k_B, c)$.
			If $b = \phi$, then fail.
			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

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

Coin-Flipping

Flipping a common coin

Alice and Bob are in the process of getting divorced 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 Lecture 19 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 $b_A \in \{0,1\}$. $c_A = \mathbf{enclose}(s_A, k_B, b_A)$.	$\xleftrightarrow{c_A,\ c_B}$	Choose random bit $b_B \in \{0,1\}$. $c_B = \mathbf{enclose}(s_B, k_A, b_B)$.
3. Send s_A.	$\xleftrightarrow{s_A,\ s_B}$	Send s_B.
4. $b_B = \mathbf{reveal}(s_B, k_A, c_B)$. Coin outcome is $b = b_A \oplus b_B$.		$b_A = \mathbf{reveal}(s_A, k_B, c_A)$. 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

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

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.

Preventing a copycat attack

This problem is not so easy to overcome.

One possibility is for both Alice and Bob to check that $k_A \neq k_B$ after step 1.

That way, if Alice, say, chooses $c_A = c_B = c$ and $s_A = s_B = s$ on steps 2 and 3, there still might be a good chance that

$$b_A = \textbf{reveal}(s, k_B, c) \neq \textbf{reveal}(s, k_A, c) = b_B.$$

However, depending on the bit commitment scheme, a difference in only one bit in k_A and k_B might not be enough to ensure that different bits are revealed.

In any case, it's not enough that b_A and b_B sometimes differ. For the outcome to be unbiased, we need $\mathrm{Pr}[\mathrm{b_A} \neq \mathrm{b_B}] = 1/2$.

A better idea

A better idea might be to both check that $k_A \neq k_B$ after step 1 and then to use $h(k_A)$ and $h(k_B)$ in place of k_A and k_B, respectively, in the remainder of the protocol, where h is a hash function.

That way, even a single bit difference in k_A and k_B is likely to be magnified to a large difference in the strings $h(k_A)$ and $h(k_B)$.

This should lead to the bits $\textbf{reveal}(s_A, h(k_B), c_A)$ and $\textbf{reveal}(s_B, h(k_A), c_B)$ being uncorrelated, even if $s_A = s_B$ and $c_A = c_B$.

Locked Box Paradigm

Overview

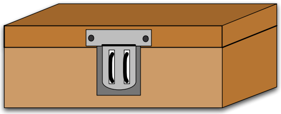
Protocols for coin flipping and for dealing a poker hand from a deck of cards can be based on the intuitive notion of locked boxes.

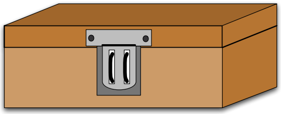
This idea in turn can be implemented using commutative-key cryptosystems.

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.





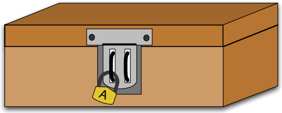
Alice writes "heads" on a slip of paper and "tails" on another.

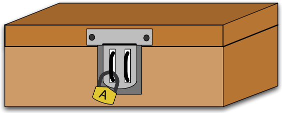
$\boxed{\text{“heads”, signed Alice}}$ $\boxed{\text{“tails”, signed Alice}}$

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.



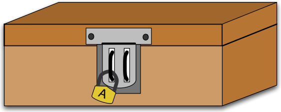


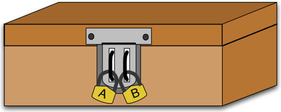
She then gives both locked boxes to Bob, in some random order.

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.

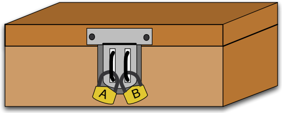




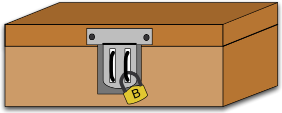
He gives the doubly-locked box back to Alice.

Alice removes her lock

Alice gets



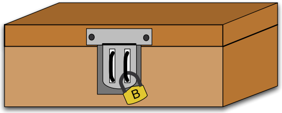
She removes her lock.



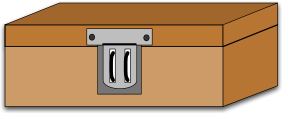
and returns the box to Bob.

Bob opens the box

Bob gets



He removes his lock



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

$\boxed{\text{“heads”, signed Alice}}$

He gives the slip to Alice.

Alice checks that Bob didn’t cheat

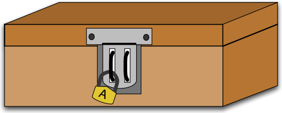
At this point, both Alice and Bob know the outcome of the coin toss.

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

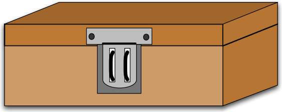
She sends her key to Bob.

Bob check 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.

$\boxed{\text{“tails”, signed Alice}}$

He checks that it contains the other coin value.

Implementation

Commutative-key cryptosystems

Alice and Bob can carry out this protocol electronically using any *commutative-key* cryptosystem, that is, one in which $E_A \circ E_B = E_B \circ E_A$.[1]

RSA is commutative for keys A and B with a common modulus n, so we can use RSA in an unconventional way.

Rather than making the encryption exponent public and keeping the factorization of n private, we turn things around.

[1] Recall the related notion of "commutative cryptosystem" of Lecture 14 in which the encryption and decryption functions for the *same* key commuted.

RSA as a commutative-key cryptosystem

Alice and Bob jointly chose primes p and q, and both compute $n = pq$.

Alice chooses an RSA key pair $A = ((e_A, n), (d_A, n))$, which she can do since she knows the factorization of n.

Similarly, Bob chooses an RSA key pair $B = ((e_B, n), (d_B, n))$ using the *same* n.

Alice and Bob both keep their key pairs private (until the end of the protocol, when they reveal them to each other to verify that there was no cheating).

Security remark

We note that this scheme may have completely different security properties from usual RSA.

In RSA, there are three different secrets involved with the key: the factorization of n, the encryption exponent e, and the decryption exponent d.

We have seen previously that knowing n and any two of these three pieces of information allows the third to be reconstructed.

Thus, knowing the factorization of n and e lets one compute d. We also showed in Lecture 9 how to factor n given both e and d.

The way RSA is usually used, only e is public, and it is believed to be hard to find the other two secrets.

A new use for RSA

Here we propose making the factorization of n public but keeping e and d private.

It may indeed be hard to find e and d, even knowing the factorization of n, but if it is, that fact is not going to follow from the difficulty of factoring n.

Of course, for security, we need more than just that it is hard to find e and d.

We also need it to be hard to find m given $c = m^e \bmod n$.

This is reminiscent of the discrete log problem, but of course n is not prime in this case.

Coin-flipping using commutative-key cryptosystems

We now implement the locked box protocol using RSA.

Here we assume that Alice and Bob initially know large primes p and q.

In step (2), Alice chooses a random number r such that $r < (n-1)/2$.

This ensures that m_0 and m_1 are both in $\mathbf{Z}_n$.

Note that i and r can be efficiently recovered from m_i since i is just the low-order bit of m_i and $r = (m_i - i)/2$.

	Alice		Bob
1.	Choose RSA key pair A with modulus $n = pq$.		Choose RSA key pair B with modulus $n = pq$.
2.	Choose random $r \in \mathbf{Z}_{(n-1)/2}$. Let $m_i = 2r + i$, for $i \in \{0, 1\}$. Let $c_i = E_A(m_i)$ for $i \in \{0, 1\}$. Let $C = \{c_0, c_1\}$.	$\xrightarrow{C}$	Choose $c_a \in C$.
3.		$\xleftarrow{c_{ab}}$	Let $c_{ab} = E_B(c_a)$.
4.	Let $c_b = D_A(c_{ab})$.	$\xrightarrow{c_b}$	
5.		$\xleftarrow{B}$	Let $m = D_B(c_b)$. Let $i = m \bmod 2$. Let $r = (m - i)/2$. If $i = 0$ then "tails". If $i = 1$ then "heads".

	Alice		Bob
6.	Let $m = D_B(c_b)$. Check $m \in \{m_0, m_1\}$. If $m = m_0$ then "tails". If $m = m_1$ then "heads".	$\xrightarrow{A}$	
7.			Let $c_a' = C - \{c_a\}$. Let $m' = D_A(c_a')$. Let $i' = m' \bmod 2$. Let $r' = (m' - i')/2$. Check $i' \neq i$ and $r' = r$.

Correctness when Alice and Bob are honest

When both Alice and Bob are honest, Bob computes $c_{ab} = E_B(E_A(m_j))$ for some $j \in \{0, 1\}$.

In step 4, Alice computes c_b.
By the commutativity of E_A and E_B,

$$c_b = D_A(E_B(E_A(m_j))) = E_B(m_j).$$

Hence, in step 5, $m = m_j$ is one of Alice's strings from step 2.

A dishonest Bob

A dishonest Bob can control the outcome of the coin toss if he can find two keys B and B' such that $E_B(c_a) = E_{B'}(c'_a)$, where $C = \{c_a, c'_a\}$ is the set received from Alice in step 2.

In this case, $c_{ab} = E_B(E_A(m_j)) = E_{B'}(E_A(m_{1-j}))$ for some j. Then in step 4, $c_b = D_A(c_{ab}) = E_B(m_j) = E_{B'}(m_{1-j})$.

Hence, $m_j = D_B(c_b)$ and $m_{1-j} = D_{B'}(c_b)$, so Bob can obtain both of Alice's messages and then send B or B' in step 5 to force the outcome to be as he pleases.

To find such B and B', Bob would need to solve the equation

$$c_a^e \equiv {c'_a}^{e'} \pmod{n}$$

for e and e'. Not clear how to do this, even knowing the factorization of n.

Card dealing using locked boxes

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

Alice takes a deck of cards, places each card in a separate box, and locks each box with her lock.

She arranges the boxes in random order and ships them off to Bob.

Bob picks five boxes, locks each with his lock, and send them back.

Alice removes her locks from those five boxes and returns them to Bob.

Bob unlocks them and obtains the five cards of his poker hand.

Further details are left to the reader.

Oblivious Transfer

Locked-box protocol revisited

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,

Oblivious Transfer of One Secret

Oblivious transfer of one secret

Alice has a secret s.

An oblivious transfer protocol has two equally-likely outcomes:

1. Bob learns s.
2. Bob learns nothing.

Afterwards, Alice doesn't know whether or not Bob learned s.

A cheating Bob can do nothing to increase his chance of getting s.

A cheating Alice can do nothing to learn whether or not Bob got her secret.

Rabin proposed an oblivious transfer protocol based on quadratic residuosity in the early 1980's.

Rabin's OT protocol

	Alice		Bob
1.	Secret s. $n = pq$, $p \neq q$ prime. RSA public key (e, n). Compute $c = E_{(e,n)}(s)$.	$\xrightarrow{(e,n,c)}$	
2.		$\xleftarrow{a}$	Choose random $x \in \mathbf{Z}_n^*$. Compute $a = x^2 \bmod n$.
3.	Check $a \in \mathrm{QR}_n$. Random $y \in \sqrt{a} \pmod{n}$.	$\xrightarrow{y}$	
4.			Check $y^2 \equiv a \pmod{n}$. If $y \not\equiv \pm x \pmod{n}$, use x, y to factor n and decrypt c to obtain s.

Analysis

Alice can can carry out step 3 since she knows the factorization of n and can find all four square roots of a.

However, Alice has no idea which x Bob used to generate a.

Hence, with probability $1/2$, $y \equiv \pm x \pmod{n}$ and with probability $1/2$, $y \not\equiv \pm x \pmod{n}$.

If $y \not\equiv \pm x \pmod{n}$, then the two factors of n are $\gcd(x - y, n)$ and $n/\gcd(x - y, n)$, so Bob factors n and decrypts c in step 4.

However, if $y \equiv \pm x \pmod{n}$, Bob learns nothing, and Alice's secret is as secure as RSA itself.

A potential problem

There is a potential problem with this protocol.

A cheating Bob in step 2 might send a number *a* which he generated by some means other than squaring a random *x*.

In this case, he always learns something new no matter which square root Alice sends him in step 3.

Perhaps that information, together with what he already learned in the course of generating *a*, is enough for him to factor *n*.

Is this a real problem?

We don't know of any method by which Bob can find a quadratic residue $a \pmod{n}$ without also knowing one of a's square roots.

We certainly don't know of any method that would produce a quadratic residue a together with some other information Ξ that, combined with a square root y, would allow Bob to factor n.

But we also cannot prove that no such method exists.

A modified protocol

We fix this problem by having Bob prove that he knows a square root of the number a that he sends Alice in step 2.

He does this using a zero knowledge proof of knowledge of a square root of a.

This is essentially what the simplified Feige-Fiat-Shamir protocol of Lecture 16 does, but with the roles of Alice and Bob reversed.

- Bob claims to know a square root x of the public number a.
- He wants to prove to Alice that he knows x, but he does not want Alice to get any information about x.
- If Alice were to learn x, then she could choose $y = x$ and eliminate Bob's chance of learning s while still appearing to play honestly.