CPSC 467b: Cryptography and Computer Security

Michael J. Fischer

Lecture 21
April 11, 2013

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

A *bit-commitment* is an encryption of a bit using a cryptosystem with a special property.

1. The bit is hidden from anyone not knowing the secret key.
2. There is only one valid way of decrypting the ciphertext, no matter what key is used.

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

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

Bit commitment intuition

In other words,

- If Bob produces a commitment c to a bit b, then b cannot be recovered from c without knowing Bob's secret encoding key k.
- There is no key k' that Bob might release that would make it appear that c is a commitment of the bit $1 - b$.

Bit-commitments as cryptographic envelopes

More formally, 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.

Bit-commitment primitives

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.

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 $r' \cdot b' = D_k(c)$. Check $r' = r$. 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 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.

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 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 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.

Interactive Proof of Graph Non-Isomorphism

Other kinds of interactive proofs

Not all interactive proofs follow this simple (x, b, y) pattern.

Suppose Alice wants to prove to Bob that G_0 and G_1 are *non*-isomorphic graphs.

Even ignoring questions of Alice's privacy, there is no obvious data that she can send Bob that will allow him to easily verify that the two graphs are not isomorphic.

However, under a different set of assumptions, Alice can convince Bob that they can't be isomorphic, even though Bob can't do so by himself.

An all-powerful teacher

In this version of interactive proof, we assume that Alice is all-powerful and can compute intractable problems. In particular, given two graphs, she can determine whether or not they are isomorphic.

Bob on the other hand has no extraordinary powers and can just perform computation in the usual way.

Alice uses her computational powers to distinguish isomorphic copies of G_0 from isomorphic copies of G_1. If $G_0 \cong G_1$, there is no way she could do this, since any graph H isomorphic to one of them is also isomorphic to the other.

So by convincing Bob that she is able to reliably distinguish such graphs, she also convinces him that $G_0 \ncong G_1$.

Interactive proof of graph non-isomorphism

	Alice		Bob
1.			Choose random $b \in \{0,1\}$. Compute a random isomorphic copy H of G_b.
		$\xleftarrow{H}$	
2.	If $H \cong G_0$ let $b' = 0$.		
	If $H \cong G_1$ let $b' = 1$.	$\xrightarrow{b'}$	Check $b' = b$.

Graph non-isomorphism IP is not zero-knowledge

Alice performs a computation for Bob that he could not do himself.

Namely, Alice willingly tells Bob for any H of his choosing whether it is isomorphic to G_0 or to G_1.

(In any implementation of the protocol, she also probably tells him if H is not isomorphic to either one, perhaps by failing in step 2 when b' is undefined.)

Bit Commitment in Graph Non-Isomorphism IP

Non-isomorphism protocol viewed as bit commitment

In the non-isomorphism IP, H is a commitment of Bob's bit b.

Suppose Bob gives H to Carol (who doesn't have Alice's extraordinary computational powers).

Later Bob could convince Carol of his bit by telling her the isomorphism that proves $H \cong G_b$.

But there is nothing he could do to make her believe that his bit was really $1 - b$ since $H \ncong G_{1-b}$.

The actual protocol doesn't use the commitment in quite this way. Rather than having Bob later reveal his bit, Alice uses her special powers to discover the bit committed by H.

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 the beginning of today's lecture 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 $b_A \in \{0,1\}$. $c_A = \textbf{enclose}(s_A, k_B, b_A)$.	$\xleftrightarrow{c_A,\ c_B}$	Choose random bit $b_B \in \{0,1\}$. $c_B = \textbf{enclose}(s_B, k_A, b_B)$.
3. Send s_A.	$\xleftrightarrow{s_A,\ s_B}$	Send s_B.
4. $b_B = \textbf{reveal}(s_B, k_A, c_B)$. Coin outcome is $b = b_A \oplus b_B$.		$b_A = \textbf{reveal}(s_A, k_B, c_A)$. 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

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

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.

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 $\Pr[\mathrm{b_A} \neq \mathrm{b_B}] = 1/2$.

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 **reveal**$(s_A, h(k_B), c_A)$ and **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

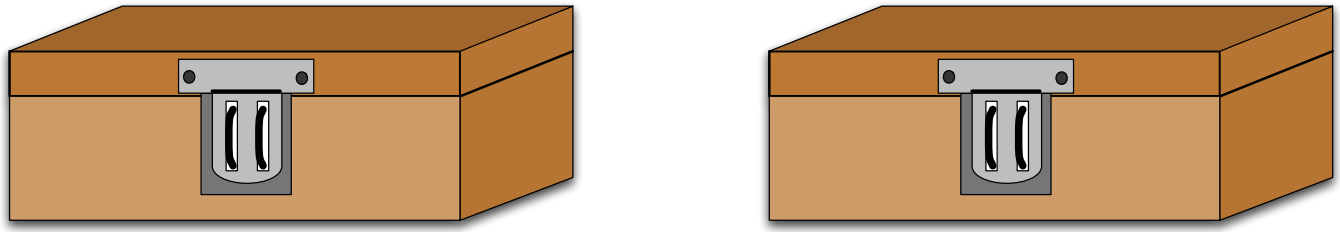
Protocols for coin flipping and for dealing a poker hand from a deck of cards can be based on the intuitive notion of locked boxes.

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.



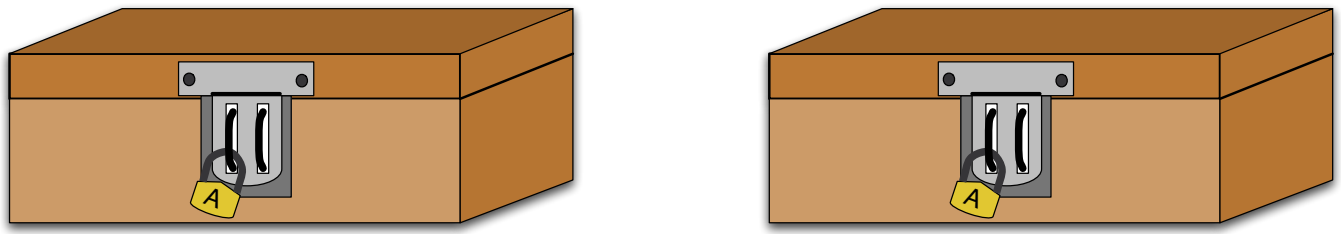
Alice writes "heads" on a slip of paper and "tails" on another.

"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.

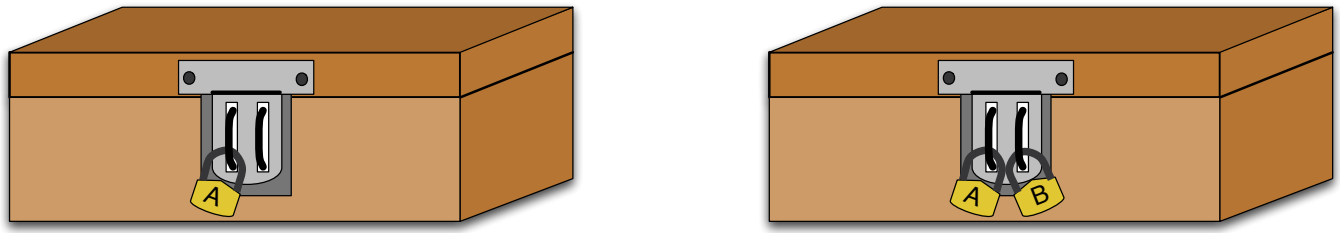


She then gives both locked boxes to Bob, in some random order.

Bob adds his lock

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

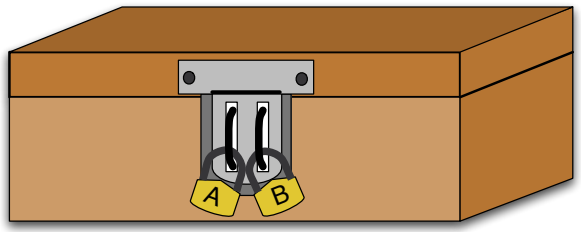
He chooses one of the boxes and locks it with his own padlock, for which he has the only key.



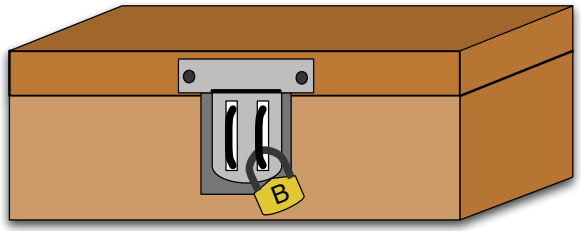
He gives the doubly-locked box back to Alice.

Alice removes her lock

Alice gets



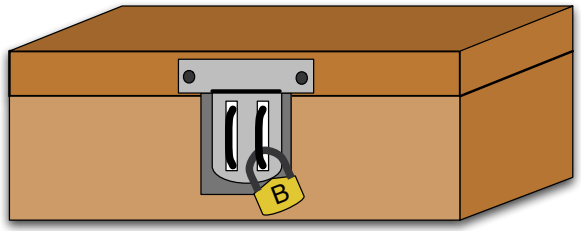
She removes her lock.



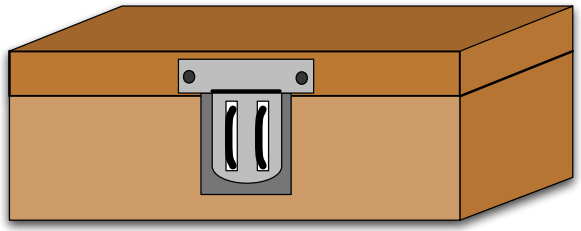
and returns the box to Bob.

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

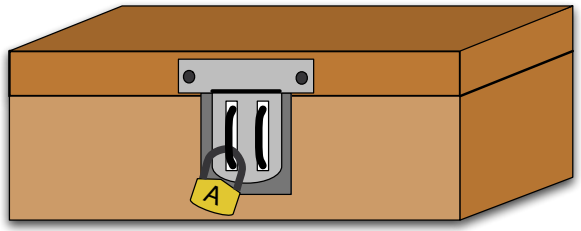
At this point, both Alice and Bob know the outcome of the coin toss.

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

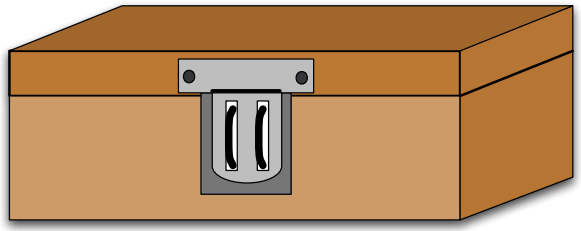
She sends her key to Bob.

Bob 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.

"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 12 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 10 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.			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".
		$\xleftarrow{B}$	

	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.