CPSC 467: Cryptography and Computer Security

Michael J. Fischer

Lecture 23
November 29, 2017

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.

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 Alice produces a commitment c to a bit b, then b 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 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, the 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 she runs **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.

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 QR Cryptosystem

A simple (but inefficient) bit commitment scheme

Here's a simple way for Alice to commit to a 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 20, slide 29.)
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 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 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.

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 $b_A \in \{0,1\}$. $c_A = \textbf{enclose}(s_A, k_B, b_A)$.	$\xleftrightarrow{c_A,\ c_B}$	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 $c_A = c_B$.

A possible problem

The general properties for **reveal** given in the appendix do not seem strong enough to imply that **enclose**(k_A, k_B, b) has a strong enough dependence on k_B to prevent cheating by an Alice who knows a commitment c that she can open to reveal either 0 or 1.

For example, suppose **enclose**(k_A, k_B, b) does not depend on k_B.

If Alice knows s_0, k_0, s_1, k_1 such that **enclose**$(s_0, k_0, 0) =$ **enclose**$(s_1, k_1, 1) = c$, it follows that **enclose**$(s_0, k_B, 0) =$ **enclose**$(s_1, k_B, 1) = c$ for any k_B. Hence, **reveal**$(s_0, k_B, c) = 0$ and **reveal**$(s_1, k_B, c) = 1$.

The bit-commitment implementations using the QR cryptosystem and using pseudorandom sequence generators both do strongly depend on Bob's random value k_B.

Appendix: Formalization of Bit Commitment Schemes

Formalization of bit commitment schemes

The above bit commitment protocols all have the same form.

We abstract from them 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

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