YALE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

CPSC 467a: Cryptography and Computer Security *Notes 21 (rev. 1)*
Professor M. J. Fischer *November 28, 2006*

Lecture Notes 21

105 Bit-Commitment Using Pseudorandom Sequence Generators

A pseudorandom sequence generator (PRSG) maps a "short" random seed to a "long" pseudorandom bit string. For a PRSG to be cryptographically strong, it must be difficult to correctly predict any generated bit, even knowing all of the other bits of the output sequence. In particular, it must also be difficult to find the seed given the output sequence, since if one knows the seed, then the whole sequence can be generated. Thus, a PRSG is a one-way function and more. While a hash function might generate hash values of the form yy and still be strongly collision-free, such a function could not be a PRSG since it would be possible to predict the second half of the output knowing the first half.

I am being intentionally vague at this stage about what "short" and "long" mean, but intuitively, "short" is a length like we use for cryptographic keys—long enough to prevent brute-force attacks, but generally much shorter than the data we want to deal with. Think of "short"=128 or =256 and you'll be in the right ballpark. By "long", we mean much larger sizes, perhaps thousands or even millions of bits. In practice, we usually thing of the output length as being variable, so that we can request as many output bits from the generator as we like and it will deliver them. Also, in practice, the bits are generally delivered a block at a time rather than all at once, so we don't even need to announce in advance how many bits we want but can go back as needed to get more.

There are many ways to use a PRSG G for bit commitment. One such way is shown in Figure 1. Here, ρ is a security parameter that controls the probability that a cheating Alice can fool Bob. We let $G_\rho(s)$ denote the first ρ bits of $G(s)$.

	Alice		Bob
	To **commit**(b):		
1.		$\xleftarrow{r}$	Choose random string $r \in \{0,1\}^\rho$.
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.

Figure 1: Bit commitment using PRSG.

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.

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:

$$\left.\begin{array}{rcl} c & = & G_\rho(s_0) \\ c & = & G_\rho(s_1) \oplus r. \end{array}\right\} \tag{1}$$

It is sufficient for her to solve the equation

$$r = G_\rho(s_0) \oplus G_\rho(s_1) \tag{2}$$

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

One might ask why Bob needs 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 (2) and cheat. If r is fixed to a constant, then if Alice ever finds a triple (c, s_0, s_1) satisfying (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. For example, suppose $r = 1^\rho$ and $G_\rho(\neg s_0) = \neg G_\rho(s_0)$ for some s_0. Then (2) could be solved by taking $s_1 = \neg s_0$. By having Bob choose r at random, r will be different each time (with very high probability), and a successful cheating Alice would be forced to solve (1) in general, not just for one special case.

106 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* $\mathcal{K_A}$ and $\mathcal{K_B}$, a *blob space* $\mathcal{B}$, a *commitment* function

$$\mathbf{enclose} : \mathcal{K_A} \times \mathcal{K_B} \times \{0,1\} \to \mathcal{B},$$

and an *opening* function

$$\mathbf{reveal} : \mathcal{K_A} \times \mathcal{K_B} \times \mathcal{B} \to \{0, 1, \phi\},$$

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

These functions have three properties:

1. $\forall k_A \in \mathcal{K}_A, \forall k_B \in \mathcal{K}_B, \forall b \in \{0,1\}, \mathbf{reveal}(k_A, k_B, \mathbf{enclose}(k_A, k_B, b)) = b$;

2. $\forall k_B \in \mathcal{K}_B, \forall c \in \mathcal{B}, \exists b \in \{0,1\}, \forall k_A \in \mathcal{K}_A, \mathbf{reveal}(k_A, k_B, c) \in \{b, \phi\}$.

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.

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.

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

Figure 2: A generic bit commitment protocol.

A bit-commitment scheme looks a lot like a symmetric cryptosystem, with $\mathbf{enclose}(k_A, k_B, b)$ playing the role of the encryption function and $\mathbf{reveal}(k_A, k_B, c)$ 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 condition 2, nor do they necessarily satisfy it. In a conventional cryptosystem, it is assumed 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 scheme can be turned into a bit-commitment protocol by plugging it into the generic protocol given in Figure 2. Each of the bit-commitment protocols of sections 103 and 104 (lecture notes 20), and section 105 above can be regarded as an instance of the generic protocol. For example, we get the protocol of Figure 1 in section 103 (lecture notes 20) by taking $\mathbf{enclose}(k_A, k_B, b) = E_{k_A}(k_B \cdot b)$, and $\mathbf{reveal}(k_A, k_B, c) = \begin{cases} b & \text{if } k_B \cdot b = D_{k_A}(c) \\ \phi & \text{otherwise.} \end{cases}$

107 Coin-Flipping

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. "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." This sounds reasonable to Alice, so she lets him go on to propose the protocol of Figure 3. In this protocol, 1 means "heads" and 0 means "tails".

After Alice considers Figure 3 for awhile, she objects. "This isn't fair. You get to see my coin flip before I see yours, so now you have complete control over the value of b." She suggests that she would be happy if the first two steps were reversed, so that Bob flipped his coin first, but Bob balks

	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 $b = b_A \oplus b_B$.

Figure 3: Distributed coin flip protocol requiring honest parties.

	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 $b_B \in \{0,1\}$.
	$c_A = \mathbf{enclose}(s_A, k_B, b_A)$.	$\xleftrightarrow{c_A,\ c_B}$	$c_B = \mathbf{enclose}(s_B, k_A, b_B)$.
3.	Send s_A.	$\xleftrightarrow{s_A,\ s_B}$	Send s_B.
4.	$b_B = \mathbf{reveal}(s_B, k_A, c_B)$.		$b_A = \mathbf{reveal}(s_A, k_B, c_A)$.
	Coin outcome is $b = b_A \oplus b_B$.		Coin outcome is $b = b_A \oplus b_B$.

Figure 4: Distributed coin flip protocol using blobs.

at that suggestion.

They then both remember last week's lecture and decide to use blobs to prevent either party from controlling the outcome. They agree on the protocol of Figure 4. At the completion of step 2, both Alice and Bob have each other's commitment (something they failed to achieve in the past, which is why they're in the middle of a divorce now), but neither know the other's private bit. They each learn each other's bit at the completion of steps 3 and 4.

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.

We haven't really addressed the question for any of these protocols about what happens if one party quits in the middle or one party detects the other party cheating. We have only been concerned until now with 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. That in turn 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? It may be instructive to think about the recourse that Bob has in similar real-life situations and to consider the reasons why such situations rarely arise. For example, what happens if someone fails to follow the provisions of a contract or if someone ignores a summons to appear in court?

There is another subtle problem with the protocol of Figure 4. Suppose that 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. In step 4, Bob will compute

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

so he won't detect that anything is wrong, and the coin outcome is $b = b_A \oplus b_B = b_A \oplus b_A = 0$. Hence, Alice can force the outcome to be 0 simply by playing copycat.

This problem is not so easy to overcome. One possibility is for both Alice and Bob to check after step 1 that $k_A \neq k_B$. That way, if Alice, say, plays copycat on steps 2 and 3, there is a good chance that

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

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.

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 $\mathbf{reveal}(s_A, h(k_B), c_A)$ and $\mathbf{reveal}(s_B, h(k_A), c_B)$ being uncorrelated, even if $s_A = s_B$ and $c_A = c_B$.

108 Locked Box Paradigm

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 cryptosystems.

108.1 Coin-flipping using locked boxes

We discussed the coin-flipping problem in section 107 and presented a protocol based on bit-commitment. Here we present a coin-flipping protocol based on the idea of locked 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 and places one of these slips in each box. She 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 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. Now the box has two locks on it, one belonging to Alice and one to Bob. He gives the doubly-locked box back to Alice.
- Alice removes her lock and returns the box to Bob.
- Bob removes his lock, opens the box, and learns the outcome of the coin toss. He gives the slip of paper from the unlocked box back to Alice.
- Alice verifies that it is her slip of paper, with her handwriting on it, that she prepared at the beginning. She sends her key to Bob.
- Bob removes Alice's lock from the other box and verifies that she carried out her protocol correctly. (In particular, he checks that the slip of paper in the other box contains the other coin value.)

108.2 Commutative cryptosystems

Alice and Bob can carry out this protocol electronically using any *commutative* cryptosystem, that is, one in which $E_A(E_B(m)) = E_B(E_A(m))$ for all messages m. RSA is commutative for keys 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. Alice and Bob jointly chose primes p and q, and both compute $n = pq$. Alice then 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).

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 pieces of information allows the third to be reconstructed. Thus, knowing the factorization of n and e lets one compute d (easy). We also showed in section 55.3 ((lecture notes 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 quantities. 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.

108.3 Coin-flipping using commutative cryptosystems

Assuming RSA used in this new way is secure, we can implement the locked box protocol as shown in Figure 5. 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 $r = (m_i - i)/2$.

To see that the protocol works when both Alice and Bob are honest, observe that in step 3, $c_{ab} = E_B(E_A(m_j))$ for some j. Then in step 4, $c_b = D_A(E_B(E_A(m_j))) = E_B(m_j)$ by the commutativity of E_A and E_B. Hence, in step 5, $m = m_j$ is one of Alice's strings from step 2.

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, $c_b = E_B(m_j) = E_{B'}(m_{1-j})$. 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.

108.4 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.

	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 $i = 0$ outcome is “tails”. If $i = 1$ outcome is “heads”.
6.	Let $m = D_B(c_b)$. Check $m \in \{m_0, m_1\}$. If $m = m_0$ outcome is “tails”. If $m = m_1$ outcome is “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 that $i' \neq i$ and $r' = r$.

Figure 5: Distributed coin flip protocol using locked boxes.