YALE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

CPSC 467a: Cryptography and Computer Security *Notes 22 (rev. 1)*
Professor M. J. Fischer *November 30, 2006*

Lecture Notes 22

109 Oblivious Transfer

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,

109.1 Oblivious transfer of a secret

Alice has a secret s. In an oblivious transfer protocol, half of the time Bob learns s and half of the time he learns nothing. Afterwards, Alice doesn't know whether or not Bob learned s. Bob can do nothing to increase his chances of getting s, and Alice can do nothing to learn whether or not Bob got her secret. Rabin proposed an oblivious transfer protocol based on quadratic residuosity, shown in Figure 1, in the early 1980's.

	Alice		Bob
1.	Secret s. Choose primes p, q, and let $n = pq$. Generate 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 $a \in \mathrm{QR}_n$. Choose y at random from among the 4 square roots of $a \pmod n$.	$\xrightarrow{y}$	
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.

Figure 1: Rabin's oblivious transfer protocol.

Alice can can carry out step 3 since she knows the factorization of n and can find all of the square roots of a. However, she 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 $y \equiv \pm x \pmod n$, he learns nothing, and Alice's secret is as secure as RSA itself.

There is one potential problem with this protocol. A cheating Bob in step 2 might send a number a which he generated by some other means than squaring a random x. In this case, he 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. We don't know of any method by which Bob could find a quadratic residue without also knowing one of its square roots. We certainly don't know of any method that would produce a quadratic residue a and some other information Ξ that, combined with y, would allow Bob to factor n. But we also cannot prove that no such method exists.

We can fix this protocol by inserting between steps 2 and 3 a zero knowledge proof that Bob knows a square root of a. This is essentially what the simplified Feige-Fiat-Shamir protocol does, but we have to reverse the roles of Alice and Bob. Now Bob is the one with the secret square root x. He wants to prove to Alice that he knows x, but he does not want Alice to get any information about x, since if she learns x, she could choose $y = x$ and reduce his chances of learning s while still appear to be playing honestly. Again, details are left to the reader.

109.2 One-of-two oblivious transfer

In the *one-of-two oblivious transfer*, Alice has two secrets, s_0 and s_1. Bob always gets exactly one of the secrets. He gets each with probability 1/2, and Alice does not know which he got.

The locked box protocol is one way to implement one-of-two oblivious transfer. Another is based on a public key cryptosystem (such as RSA) and a symmetric cryptosystem (such as AES). This protocol, given in Figure 2, does not rely on the cryptosystems being commutative.

	Alice		Bob
1.	Choose two PKS key pairs (e_0, d_0) and (e_1, d_1).	$\xrightarrow{e_0,e_1}$	
2.		$\xleftarrow{c}$	Generate random key k for a symmetric cryptosystem $(\hat{E}, \hat{D})$. Choose random $b \in \{0, 1\}$. Compute $c = E_{e_b}(k)$.
3.	Compute $k_i = D_{d_i}(c)$ for $i = 0, 1$. Choose $b' \in \{0, 1\}$. Compute $c_i = \hat{E}_{k_i}(s_{i \oplus b'})$ for $i = 0, 1$.	$\xrightarrow{c_0,c_1}$	
4.			Output $s = s_{b \oplus b'} = \hat{D}_k(c_b)$.

Figure 2: One-of-two oblivious transfer using two PKS key pairs.

In step 2, Bob encrypts a randomly chosen key k for the symmetric cryptosystem using one of the PKS encryption keys that Alice sent him in step 1. In step 2, Bob selects one of the two encryption keys from Alice, uses it to encrypt k, and sends the encryption to Alice. In step 3, Alice decrypts c using both decryption keys d_0 and d_1 to get k_0 and k_1. One of the k_i is Bob's key k (k_b to be specific) and the other is garbage, but because k is random and she doesn't know b, she can't tell which is k. She then encrypts one secret with k_0 and the other with k_1, using the random bit b' to ensure that each secret is equally likely to be encrypted by the key that Bob knows. In step 4, Bob decrypts the ciphertext c_b using key his key $k = k_b$ to recover the secret $s = s_{b \oplus b'}$. He can't decrypt the other ciphertext $c_{1 \oplus b}$ since he doesn't know the key $k_{1 \oplus b}$ used to produce it, nor does he know the decryption key $d_{1 \oplus b}$ that would allow him to find it from c.

110 Pseudorandom Sequence Generation

We mentioned pseudorandom sequence generation in section 29 of lecture notes 6 and section 105 of lecture notes 21. In a little more detail, a *pseudorandom sequence generator G* is a function from a domain of *seeds* $\mathcal{S}$ to a domain of strings $\mathcal{X}$. We will generally find it convenient to assume that all of the seeds in $\mathcal{S}$ have the same length m and that all of the strings in $\mathcal{X}$ have the same length n, where $m \ll n$ and n is polynomially related to m.

Intuitively, we want the strings $G(s)$ to "look random". But what does that mean? Chaitin and Kolmogorov proposed ways of defining what it means for an individual sequence to be considered random. While philosophically very interesting, these notions are somewhat different than the statistical notions that most people mean by randomness.

We take a different tack. We assume that the seeds are chosen uniformly at random from $\mathcal{S}$, that is, we consider a uniformly distributed random variable S over $\mathcal{S}$. Then $X = G(S)$ is a random variable over $\mathcal{X}$. For $x \in \mathcal{X}$,

$$\mathrm{prob}[X = x] = \frac{|\{s \in \mathcal{S} \mid G(s) = x\}|}{|\mathcal{S}|}.$$

That is, the probability is the fraction of seeds that give rise to x. Because $m \ll n$, $|\mathcal{S}| = 2^m$, and $|\mathcal{X}| = 2^n$, most strings in $\mathcal{X}$ are not in the range of G and hence have probability 0. If G happens to be one-to-one, then the remaining strings each have probability $1/2^m$.

We also consider the uniform random variable $U \in \mathcal{X}$, where $U = x$ with probability $1/2^n$ for every $x \in \mathcal{X}$. U is what we usually mean by a "truly random" variable on n-bit strings.

We will say that G is a *cryptographically strong* pseudorandom sequence generator if X and U are *indistinguishable* to all probabilistic polynomial Turing machines. We have already seen that the probability distributions of X and U are quite different. Nevertheless, they are indistinguishable if there is no feasible algorithm to determine whether random samples come from X or from U.

Before going further, let me describe some functions G for which $G(S)$ is readily distinguished from U. Suppose every string $x = G(s)$ has the form $b_1b_1b_2b_2b_3b_3\ldots$, for example 0011111100001100110000.... An algorithm that guesses that x came from $G(S)$ if x is of that form, and guesses that x came from U otherwise, will be right almost all of the time. True, it is possible to get a string like this from U, but it is extremely unlikely.

Formally speaking, a *judge* is a probabilistic Turing machine J that takes an n-bit string as input and produces a single bit b as output. Because it is probabilistic, it actually defines a random function from $\mathcal{X}$ to $\{0, 1\}$. This means that for every input x, there is some probability p_x that the output is 1. If the input string is itself a random variable X, then the probability that the output is 1 is the weighted sum over all possible inputs that the judge outputs 1, where the weights are the probabilities of the corresponding inputs occurring. Thus, the output value is itself a random variable which we denote by $J(X)$.

Now, we say that two random variables X and Y are *ϵ-indistinguishable by judge J* if

$$|\mathrm{prob}[J(X) = 1] - \mathrm{prob}[J(Y) = 1]| < \epsilon.$$

Intuitively, we say that G is *cryptographically strong* if $G(S)$ and U are ϵ-indistinguishable for suitably small ϵ by all judges that do not run for too long. A careful mathematical treatment of the concept of indistinguishability must relate the length parameters m and n, the error parameter ϵ, and the allowed running time of the judges, all of which is beyond the scope of this course.
[Note: The topics covered by these lecture notes are presented in more detail and with greater mathematical rigor in handout 15.]

111 BBS Pseudorandom Sequence Generator

We present a cryptographically strong pseudorandom sequence generator BBS, due to Blum, Blum, and Shub. BBS is defined by a Blum integer n and an integer ℓ. It maps strings in $\mathbf{Z}_n^*$ to strings in $\{0,1\}^\ell$. Given a seed $s_0 \in \mathbf{Z}_n^*$, we define a sequence $s_1, s_2, s_3, \ldots, s_\ell$, where $s_i = s_{i-1}^2 \bmod n$ for $i = 1, \ldots, \ell$. The ℓ-bit output sequence $\mathrm{BBS}(s_0)$ is $b_1, b_2, b_3, \ldots, b_\ell$, where $b_i = \mathrm{lsb}(s_i)$.

The security of BBS is based on the difficulty of determining, for a given $a \in \mathbf{Z}_n^*$ with Jacobi symbol $\left(\frac{a}{n}\right) = 1$, whether or not a is a quadratic residue, i.e., whether or not $a \in \mathrm{QR}_n$. Recall from section 66 of lecture notes 12, that this is the property upon on which the security of the Goldwasser-Micali probabilistic encryption system relies.

Blum integers were introduced in section 95 of lecture notes 19. Recall that a Blum prime is a prime p such $p \equiv 3 \pmod 4$, and a Blum integer is a number $n = pq$, where p and q are distinct Blum primes. Blum primes and Blum integers have the important property that every quadratic residue a has exactly one square root y which is itself a quadratic residue. We call such a y the *principal square root* of a and denote it by $\sqrt{a} \pmod n$ or simply by $\sqrt{a}$ when it is clear that $\pmod n$ is intended.

We need two other facts about Blum integers n.

Claim 1 *Let $a \in \mathrm{QR}_n$. Then* $\left(\frac{a}{n}\right) = \left(\frac{-a}{n}\right) = 1$.

Proof: This follows from the fact that if a is a quadratic residue modulo a Blum prime, then $-a$ is a quadratic non-residue. Hence,

$$\left(\frac{a}{p}\right) = -\left(\frac{-a}{p}\right) \text{ and } \left(\frac{a}{q}\right) = -\left(\frac{-a}{q}\right),$$

so

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p}\right) \cdot \left(\frac{a}{q}\right) = \left(-\left(\frac{-a}{p}\right)\right) \cdot \left(-\left(\frac{-a}{q}\right)\right) = \left(\frac{-a}{n}\right).$$

■

Let $\mathrm{lsb}(x)$ be the least significant bit of integer x. That is, $\mathrm{lsb}(x) = (x \bmod 2)$.

Claim 2 *Let $x \in \mathbf{Z}_n$, where n is odd. Then* $\mathrm{lsb}(x) \oplus \mathrm{lsb}(-x) = 1$.

112 Security of BBS generator

We begin our proof that BBS is cryptographically strong by showing that there is no probabilistic polynomial time algorithm A that, given $b_2, \ldots, b_\ell$, is able to predict b_1 with accuracy greater than $1/2 + \epsilon$. This is not sufficient to establish that the pseudorandom sequence $\mathrm{BBS}(S)$ is indistinguishable from the uniform random sequence U, but if it did not hold, there certainly would exist a distinguishing judge. Namely, the judge that outputs 1 if $b_1 = A(b_2, \ldots, b_\ell)$ and 0 otherwise would output 1 with probability greater than $1/2 + \epsilon$ in the case that the sequence came from $\mathrm{BBS}(S)$ and would output 1 with probability exactly $1/2$ in the case that the sequence was truly random.

We now show that if a probabilistic polynomial time algorithm A exists for predicting b_1 with accuracy greater than $1/2 + \epsilon$, then there is a probabilistic polynomial time algorithm Q for testing quadratic residuosity with the same accuracy. Thus, if quadratic-residue-testing is "hard", then so first-bit prediction for BBS.[1]

[1]See handout 15 for further results on the security of BBS.

Theorem 1 *Let A be a first-bit predictor for* $\mathrm{BBS}(S)$ *with accuracy* $1/2+\epsilon$*. Then we can find an algorithm Q for testing whether a number x with Jacobi symbol 1 is a quadratic residue, and Q will be correct with probability at least* $1/2+\epsilon$.

Proof: Assume that A predicts b_1 given $b_2, \ldots, b_\ell$. Algorithm $Q(x)$, shown in Figure 3, tests whether or not a number x with Jacobi symbol 1 is a quadratic residue modulo n. It outputs 1 to mean $x \in \mathrm{QR}_n$ and 0 to mean $x \notin \mathrm{QR}_n$.

```
To $Q(x)$:
1.  Let $\hat{s}_2 = x^2 \bmod n$.
2.  Let $\hat{s}_i = \hat{s}_{i-1}^2 \bmod n$, for $i = 3, \ldots, \ell$.
3.  Let $\hat{b}_1 = \mathrm{lsb}(x)$.
4.  Let $\hat{b}_i = \mathrm{lsb}(\hat{s}_i)$, for $i = 2, \ldots, \ell$.
5.  Let $c = A(\hat{b}_2, \ldots, \hat{b}_\ell)$.
6.  If $c = \hat{b}_1$ then output 1; else output 0.
```

Figure 3: Algorithm Q for testing $x \in \mathrm{QR}_n$.

Since $\left(\frac{x}{n}\right) = 1$, then either x or $-x$ is a quadratic residue. Let s_0 be the principal square root of x or $-x$. Let $s_1, \ldots, s_\ell$ be the state sequence and $b_1, \ldots, b_\ell$ the corresponding output bits of $\mathrm{BBS}(s_0)$. We have two cases.

Case 1: $x \in \mathrm{QR}_n$. Then $s_1 = x$, so the state sequence of $\mathrm{BBS}(s_0)$ is

$$s_1, s_2, \ldots, s_\ell = x, \hat{s}_2, \ldots, \hat{s}_\ell,$$

and the corresponding output sequence is

$$b_1, b_2, \ldots, b_\ell = \hat{b}_1, \hat{b}_2, \ldots, \hat{b}_\ell.$$

Since $\hat{b}_1 = b_1$, $Q(x)$ correctly outputs 1 whenever A correctly predicts b_1. This happens with probability at least $1/2+\epsilon$.

Case 2: $x \in \mathrm{QNR}_n$, so $-x \in \mathrm{QR}_n$. Then $s_1 = -x$, so the state sequence of $\mathrm{BBS}(s_0)$ is

$$s_1, s_2, \ldots, s_\ell = -x, \hat{s}_2, \ldots, \hat{s}_\ell,$$

and the corresponding output sequence is

$$b_1, b_2, \ldots, b_\ell = \neg\hat{b}_1, \hat{b}_2, \ldots, \hat{b}_\ell.$$

Since $\hat{b}_1 = \neg b_1$, $Q(x)$ correctly outputs 0 whenever A correctly predicts b_1. This happens with probability at least $1/2+\epsilon$.

In both cases, $Q(x)$ gives the correct output with probability at least $1/2+\epsilon$. ■