YALE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

CPSC 467a: Cryptography and Computer Security *Notes 22 (rev. 1)*
Professor M. J. Fischer *December 1, 2008*

Lecture Notes 22

104 Pseudorandom Sequence Generation

We mentioned cryptographically strong pseudorandom sequence generators in section 31 of lecture notes 7 in connection with stream ciphers. We now explore this topic in greater depth and show how to build one that is provably secure using the same quadratic residuosity assumption that was used in section 67, lecture notes 15, to establish the security of the Goldwasser-Micali probabilistic cryptosystem.

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, but polynomially related to the seed length. In practice, we usually think 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. In this case, "long" refers to the maximum number of bits that can be delivered while still maintaining security. Also, in practice, the bits are generally delivered a block at a time rather than all at once, so we don't need to announce in advance how many bits we want but can go back as needed to get more.

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 n and that all of the strings in $\mathcal{X}$ have the same length ℓ, where $n \ll \ell$ and ℓ is polynomially related to n.

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 and and do not seem to be useful for cryptography.

We take a different tack. We assume that the seeds are chosen truly at random from $\mathcal{S}$ according to the uniform distribution. Let S be a uniformly distributed random variable over $\mathcal{S}$. Then $X \in \mathcal{X}$ is a derived random variable defined by $X = G(S)$. For $x \in \mathcal{X}$,

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

Thus, $\mathrm{prob}[X = x]$ is the probability of obtaining x as the output of the PRSG for a randomly chosen seed.

We can think of $G()$ as a randomness amplifier. We start with a short truly random seed and obtain a long string that "looks like" a random string, even though we know it's not uniformly distributed. In fact, its distribution is very much non-uniform. Because $n \ll \ell$, $|\mathcal{S}| \leq 2^n$, and $|\mathcal{X}| = 2^\ell$, most strings in $\mathcal{X}$ are not in the range of G and hence have probability 0. For the uniform distribution over $\mathcal{X}$, all strongs have the same non-zero probability probability $1/2^\ell$.

More formally, let U be the uniformly distributed random variable over $\mathcal{X}$, so $\mathrm{prob}[U = x] = 1/2^\ell$ for every $x \in \mathcal{X}$. U is what we usually mean by a "truly random" variable on ℓ-bit strings.

We have already seen that the probability distributions of $X = G(S)$ and U are quite different. Nevertheless, it may be the case that all feasible probabilistic algorithms behave essentially the same whether given a sample chosen according to X or chosen according to U. If that is the case, we say that X and U are *algorithmically indistinguishable* and that G is a *cryptographically strong* pseudorandom sequence generator.

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 $b_1 b_1 b_2 b_2 b_3 b_3 \ldots$, for example 001111110000110011000$\ldots$. 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 of this special form from U, but it is extremely unlikely.

Formally speaking, a *judge* is a probabilistic Turing machine J that takes an ℓ-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, the output is 1 with some probability p_x, and the output is 0 with probability $1 - p_x$. If the input string is itself a random variable X, then the probability that the output is 1 is the weighted sum of p_x over all possible inputs x, where the weight is the probability $\mathrm{prob}[X = x]$ of input x 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 n and ℓ, 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 17.]

105 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 0based on the assumed 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 67 of lecture notes 15 that this is the same property upon on which the security of the Goldwasser-Micali probabilistic encryption system relies.

Blum integers were introduced in section 101 of lecture notes 21. 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 $(\text{mod } n)$ is intended.

We need two other facts about Blum integers n. The first claim is that for Blum integers, a quadratic residue and its negation both have the same Jacobi symbol 1.

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

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

■

The second claim simply says that the low-order bits of $x \bmod n$ and $(-x) \bmod n$ always differ when n is odd. Let $\mathrm{lsb}(x) = (x \bmod 2)$ be the least significant bit of integer x.

Claim 2 *Let n be odd. Then* $\mathrm{lsb}(x \bmod n) \oplus \mathrm{lsb}((-x) \bmod n) = 1$.

106 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 is first-bit prediction for BBS.[1]

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 106.1, 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 $x \notin \mathrm{QR}_n$.

[1] See handout 17 for further results on the security of BBS.

```
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 106.1: 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 $1/2 + \epsilon$. ■