18.425/6.875 Introduction to Cryptography

Lecture #9, October 16, 1991

1 Introduction

In this lecture we introduce next bit tests and give examples of pseudo-random generators.

2 Next Bit Tests

Definition 1 A next bit test is a special kind of statistical test which takes as input a prefix of a sequence and outputs a prediction of the next bit.

Definition 2 A (discrete) probability distribution on a set $S$ is a function $D : S \to [0, 1] \subseteq \mathbb{R}$ so that $\sum_{s \in S} D(s) = 1$. For brevity, probability distributions on $\{0, 1\}^k$ will be subscripted with a $k$. The notation $x \in X_n$ means that $x$ is chosen so that $\forall z \in \{0, 1\}^n \Pr[x = z] = X_n(z)$. In what follows, $U_n$ is the uniform distribution.

Recall the definition of a pseudo-random number generator:

Definition 3 A pseudo-random number generator (PSRG) is a polynomial time deterministic algorithm so that:

1. if $|x| = k$ then $|G(x)| = \hat{k}$
2. $\hat{k} > k$,
3. $G_{\hat{k}}$ is pseudo-random$^2$, where $G_{\hat{k}}$ is the probability distribution induced by $G$ on $\{0, 1\}^k$.

Definition 4 We say that a pseudo-random generator passes the next bit test $A$ if for every polynomial $Q$ there exists an integer $k_0$ such that for all $\hat{k} > k_0$ and $p < \hat{k}$

$$\Pr_{t \in G_{\hat{k}}} [A(t_1 t_2 \ldots t_p) = t_{p+1}] < \frac{1}{2} + \frac{1}{Q(k)}$$

Theorem 1 $G$ passes all next bit tests $\iff$ $G$ passes all statistical tests.

$^1$These notes were scribed by Alexander Russell.

$^2$A pseudo-random distribution is one which is polynomial time indistinguishable from $U_{\hat{k}}$. 

1
Claim 1. By analysis similar to that done in the previous lecture, $\Pr[A'(t_1 t_2 \ldots t_i) = t_{i+1}] > \frac{1}{2} + \frac{1}{kQ(k)}$.

Thus we reach a contradiction: $A'$ is a next bit test that $G$ fails, which contradicts our assumption that $G$ passes all next bit tests.

3 Examples of Pseudo-Random Generators

Each of the one way functions we have discussed induces a pseudo-random generator. Listed below are these generators (including the Blum/Blum/Shub generator which will be discussed afterwards) and their associated costs. See [1, 2, 3].
### One Way Function

<table>
<thead>
<tr>
<th>Name</th>
<th>One way function</th>
<th>Cost of computing one way function</th>
<th>Cost of computing (j)th bit of generator</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA</td>
<td>(x^e \mod n, n = pq)</td>
<td>(k^3)</td>
<td>(jk^3)</td>
</tr>
<tr>
<td>Rabin</td>
<td>(x^e \mod n, n = pq)</td>
<td>(k^2)</td>
<td>(jk^2)</td>
</tr>
<tr>
<td>Blum/Micali</td>
<td>(\text{EXP}(p, g, x))</td>
<td>(k^3)</td>
<td>(jk^3)</td>
</tr>
<tr>
<td>Blum/Blum/Shub</td>
<td>(see below)</td>
<td>(k^2)</td>
<td>(\max(k^2 \log j, k^3))</td>
</tr>
</tbody>
</table>

3.1 **Blum/Blum/Shub Pseudo-Random Generator**

The Blum/Blum/Shub pseudo-random generator uses the (proposed) one way function \(g_n(x) = x^2 \mod n\) where \(n = pq\) for primes \(p\) and \(q\) so that \(p \equiv q \equiv 3 \mod 4\). In this case, the squaring endomorphism \(x \mapsto x^2\) on \(\mathbb{Z}_n^*\) restricts to an isomorphism on \((\mathbb{Z}_n^*)^2\), so \(g_n\) is a permutation on \((\mathbb{Z}_n^*)^2\). (Recall that every square has a unique square root which is itself a square.)

**Claim 2** The least significant bit of \(x\) is a hard bit for the one way function \(g_n\).

The \(j\)th bit of the Blum/Blum/Shub generator may be computed in the following way:

\[
B(x^{2^j} \mod n) = B(x^\alpha \mod m)
\]

where \(\alpha \equiv 2^j \mod \phi(n)\). If the factors of \(n\) are known, then \(\phi(n) = (p-1)(q-1)\) may be computed so that \(\alpha\) may be computed prior to the exponentiation. \(\alpha = 2^j \mod \phi(n)\) may be computed in \(O(k^2 \log j)\) time and \(x^\alpha\) may be be computed in \(k^3\) time so that the computation of \(B(x^{2^j})\) takes \(O(\max(k^3, k^2 \log j))\) time.

An interesting feature of the Blum/Blum/Shub generator is that if the factorization of \(n\) is known, the \(2^\sqrt{n}\) bit can be generated in time polynomial in \(|n|\). The following question can be raised: let \(G_{BBS}^{BBS}(x, p, q) = B(f^{2^\sqrt{n}}(x)) \circ \ldots \circ B(f^{2^\sqrt{n}+2k}(x))\) for \(n = pq\) and \(|x| = k\). Let \(G_{BBS}^{2k}\) be the distribution induced by \(G_{BBS}^{BBS}\) on \(\{0,1\}^{2k}\).

**Open Problem 1** Is this distribution \(G_{BBS}^{2k}\) pseudo-random? Namely, can you prove that

\[
\forall Q \in \mathbb{Q}[x], \forall PTM A, \exists k_0, \forall k > k_0 |Pr_{t \in \mathbb{G}_{BBS}^{2k}}[A(t) = 1] - Pr_{t \in \mathbb{U}_{2k}}[A(t) = 1]| < \frac{1}{Q(2k)}
\]

The previous proof that \(G\) is pseudo-random doesn’t work here because in this case the factorization of \(n\) is part of the seed so no contradiction will be reached concerning the difficulty of factoring.

More generally,

**Open Problem 2** Pseudo-random generators, given seed \(x\), implicitly define an infinite string \(g_1^x g_2^x \ldots\). Find a pseudo-random generator so that the distribution created by restricting to any polynomially selected subset of bits of \(g^x\) is pseudo-random. By polynomially selected we mean examined by a polynomial time machine which can see \(g_i^x\) upon request for a polynomial number of \(i\)’s (the machine must write down the \(i\)’s, restricting \(|i|\) to be polynomial in \(|x|\)).

3
References

