Pseudo Random Generators

Cryptography and Protocols
Andrei Bulatov

Pseudorandom Generators

- Let T(n), ε(n) be functions. A collection $\{X_n\}$ of random variables with $X_n \in \{0,1\}^n$ is called (T,ε)-pseudorandom if $\{X_n\} \approx_{T,\varepsilon} \{U_n\}$

- A collection of functions $g_n : \{0,1\}^n \to \{0,1\}^{m(n)}$ is called a (T,ε)-pseudorandom generator if $\{g_n(U_n)\}$ is (T,ε)-pseudorandom

Good Pseudorandom Generators

- m(n) > n Otherwise it is trivial and useless

  m(n) – n is the stretch of a PRG
- A functions T is called superpolynomial if for any polynomial p(n), $p \in o(T)$
- A pair of functions (T,ε) is superpolynomial if T is superpolynomial and $\varepsilon(n) = {}^{1}\!/\!{}_{f(n)}$ where f is superpolynomial
- A PRG should be (T,ε)-pseudorandom for some superpolynomial pair (T,ε)
- g_n must be efficiently computable
- **PRG Axiom**: A good PRG exists

PRGs and Statistical Security

- **Lemma.**

  If m(n) > n then for any collection of functions $\{g_n\}$ we have

  $\Delta(g_n(U_n), U_m) \geq \frac{1}{2}$

- **Proof.**

  Let S_n be the image of $g_n(\{0,1\}^n)$. Clearly $|S_n| \leq 2^n \leq 2^{m(n)-1}$

  Thus $\Pr[g_n(U_n) \in S_n] = 1$ while $\Pr[U_{m(n)} \in S_n] \leq \frac{1}{2}$

Candidate PRGs: Blum – Blum - Shub

- This is a PRG that given an input of length $2n$ produces a string of bits of length m, where m is as big as we want

- Input: an n-bit integer N and integer X, $1 \leq X \leq N$

```
num_outputted = 0;
while num_outputted < m:
        X := X*X mod N
        num_outputted := num_outputted + 1
        output (least-significant-bit(X) )
endwhile
```

Blum – Blum – Shub is Good

- **Theorem**.

  The BBS PRG is (T,ε)-pseudorandom for some superpolynomial pair (T,ε) if the assumption below is true

- **Assumption**.

  There is a superpolynomial pair (T,ε) such that for any probabilistic algorithm Alg with time complexity less than $T(n)$ the following holds

  $$\Pr[\text{Alg finds factorization of a random n-bit integer}] < \varepsilon(n)$$

Candidate PRGs: RC4

- RC4 stands for Ron's Cipher no. 4
- Widely used: SSL (and then TLS), SSH, WEP, WPA (IEEE 802.11), BitTorrent protocol encryption, Microsoft Point-to-Point Encryption,
- A byte is a number from {1,…,256}
- Input: a permutation S: {1,…,256} → {1,…,256}

```
i := 0 j := 0
  while num_outputted < m :
    i := (i + 1) mod 256        j := (j + S[i]) mod 256
    swap(S[i],S[j])
    output (S[(S[i] + S[j]) mod 256] )
  endwhile
```

Candidate PRGs: RC4 (cntd)

- RC4 given an input of length 2048 produces an output of length m, which is as big as we want
- If 2048 is too much, there is another algorithm – KSE, the Key Scheduling Algorithm – that uses an input of length $40 \le n \le 128$ to generate S
- Input: a key k of length n, $40 \le n \le 128$

```
for i from 0 to 255     S[i] := i      endfor
j := 0
for i from 0 to 255
   j := (j + S[i] + k[i mod n]) mod 256
   swap(S[i],S[j])
endfor
```

JAVA Pseudorandom Generator

- JAVA uses a linear congruential pseudorandom generator

```
synchronized protected int next(int bits) {
   seed = (seed * 0x5DEECE66DL + 0xBL) & ((1L << 48) - 1);
   return (int)(seed >>> (48 - bits));
}
```