# Lecture 12

Douglas Wikström
KTH Stockholm
dog@csc.kth.se

May 8, 2015

# Randomness

- Everything we have done so far requires randomness!

# Randomness

- Everything we have done so far requires randomness!
- Can we "generate" random strings?

# Pseudo-Random Generator

**Definition.** An efficient algorithm PRG is a **pseudo-random generator (PRG)** if there exists a polynomial $p(n) > n$ such that for every polynomial time adversary $A$, if a seed $s \in \{0,1\}^n$ and a random string $u \in \{0,1\}^{p(n)}$ are chosen randomly, then

$$|\Pr[A(\mathrm{PRG}(s)) = 1] - \Pr[A(u) = 1]|$$

is negligible.

Informally, $A$ can not distinguish $\mathrm{PRG}(s)$ from a truly random string in $\{0,1\}^{p(n)}$.

# Pseudo-Random Function

Recall the definition of a pseudo-random function.

**Definition.** A family of functions $F : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ is pseudo-random if for all polynomial time oracle adversaries $A$

$$\left| \Pr_K \left[ A^{F_K(\cdot)} = 1 \right] - \Pr_{R:\{0,1\}^n \to \{0,1\}^n} \left[ A^{R(\cdot)} = 1 \right] \right|$$

is negligible.

# Pseudo-Random Function From Pseudo-Random Generator

**Construction.** Let $\mathrm{PRG} : \{0,1\}^k \to \{0,1\}^{2k}$ be a pseudo-random generator, and define a family of functions $F = \{F_K\}_{K \in \{0,1\}^k}$ as follows.

Let $x_{[i]} = (x_0, \ldots, x_i)$.

On key $K$ and input $x$, $F_K$ computes its output as follows:

1. Computes $(r_0^0, r_1^0) = \mathrm{PRG}(K)$.
2. Computes
$$(r^i_{x_{[i-1]}\|0}, r^i_{x_{[i-1]}\|1}) = \mathrm{PRG}(r^{i-1}_{x_{[i-1]}})$$
for $i = 1, \ldots, n-1$.
3. Outputs $r_{x_{[n-1]}}$.

# One-Way Permutation

**Definition.** A family $F = \{f_n\}$ of **permutations** $f_n : \{0,1\}^n \to \{0,1\}^n$ is said to be **one-way** if for a random $x \in \{0,1\}^n$ and every polynomial time algorithm $A$

$$\Pr[A(f_n(x)) = x] < \epsilon(n)$$

for a negligible function $\epsilon(n)$.

(Note that for permutations we may request the unique preimage.)

# Hardcore Bit

**Definition.** Let $F = \{f_n\}$ be a family of permutations $f_n : \{0,1\}^n \to \{0,1\}^n$ and $B = \{b_n\}$ a family of "bits" $b_n : \{0,1\}^n \to \{0,1\}$. Then $B$ is a **hardcore bit** of $F$ if for random $x \in \{0,1\}^n$ and every polynomial time algorithm $A$

$$|\Pr[A(f_n(x)) = b_n(x)] - 1/2| < \epsilon(n)$$

for a negligible function $\epsilon(n)$.

**Theorem.** For every one-way permutation, there is a (possibly different) one-way permutation with a hardcore bit.

# PRG from One-Way Permutation

**Construction.** Let $F$ be a one-way permutation and define PRG by $\mathrm{PRG}_n(s) = f_n(s) \| b_n(s)$ for $x \in \{0,1\}^n$.

# PRG from One-Way Permutation

**Construction.** Let $F$ be a one-way permutation and define PRG by $\mathrm{PRG}_n(s) = f_n(s) \| b_n(s)$ for $x \in \{0,1\}^n$.

**Theorem.** $\mathrm{PRG} : \{0,1\}^n \to \{0,1\}^{n+1}$ is a pseudo-random generator.

# PRG From Any One-Way Function

**Theorem.** There is a construction $\mathrm{PRG}_f$ that is a PRG if $f$ is a one-way function (possibly non-permutation).

The construction is very involved and is completely impractical.

# What Is Used In Practice?

Various standards contain some of the following elements.

- Fast hardware generator + "algorithmic strengthening".
- `/dev/random`
  - Entropy gathering deamon with estimate of amount of entropy.
  - FreeBSD: Executes the PRG *Yarrow* (or *Futura*) pseudo-random algorithm.
  - SunOS and Un*xes use similar approaches.
  - Windows has similar devices.
- Stream cipher, e.g. block-cipher in CFB or CTR mode.
- Hashfunction with secret prefix and counter (essentially our PRF$\rightarrow$PRG construction).

# Infamous Mistakes

- (1995) The original Netscape SSL code used time of the day and process IDs to seed its pseudorandom giving **way too little** entropy in the seed.
- (2008) Debian's OpenSSL commented out a critical part of the code that reduced the entropy of keys drastically!
- (2012) RSA public keys with common factors.

# Important Conclusions

**Some Important Conclusions:**

- Security bugs are **not** found by testing!
- With an insecure pseudo-random generator anything on top of it will be insecure.
- Any critical code must be reviewed after every modification, e.g, keep hashes of critical code.