

Pseudo-Random Generators

Douglas Wikström
KTH Stockholm
dog@csc.kth.se

March 9

- The Problem
- Increasing Extension
- PRG vs PRF
- Constructions
- Practice

Quote of the Day

Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin.

– von Neumann

Randomness

- ▶ Everything we have done so far requires randomness!

Randomness

- ▶ Everything we have done so far requires randomness!
- ▶ Can we “generate” random strings?

Physical Randomness and Deterministic Algorithms

- ▶ We could flip actual coins. This would be extremely impractical and slow (and boring unless you are Rainman).

Physical Randomness and Deterministic Algorithms

- ▶ We could flip actual coins. This would be extremely impractical and slow (and boring unless you are Rainman).
- ▶ We could generate “physical” randomness using hardware, e.g., measuring radioactive decay
 - ▶ Slow or expensive.
 - ▶ Hard to verify and trust.
 - ▶ Biased output.

Physical Randomness and Deterministic Algorithms

- ▶ We could flip actual coins. This would be extremely impractical and slow (and boring unless you are Rainman).
- ▶ We could generate “physical” randomness using hardware, e.g., measuring radioactive decay
 - ▶ Slow or expensive.
 - ▶ Hard to verify and trust.
 - ▶ Biased output.
- ▶ We could use a deterministic algorithm that outputs a “random looking string”, but what would von Neumann think of us?

Pseudo-Random Generator

A pseudo-random generator requires a **short random string** and deterministically expands this to a **longer “random looking” string**.

Pseudo-Random Generator

A pseudo-random generator requires a **short random string** and deterministically expands this to a **longer “random looking” string**.

This looks promising:

- ▶ Fast and cheap?
- ▶ Practical since it can be implemented in software or hardware?
- ▶ What is “random looking”?

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

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

is negligible.

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

Increasing Extension (1/2)

Before we consider how to construct a PRG we consider what the definition gives us:

Increasing Extension (1/2)

Before we consider how to construct a PRG we consider what the definition gives us:

- ▶ Suppose that there exists a PRG that extends its output by a **single bit**.

Increasing Extension (1/2)

Before we consider how to construct a PRG we consider what the definition gives us:

- ▶ Suppose that there exists a PRG that extends its output by a **single bit**.
- ▶ This would **not be very useful** to us, e.g., to generate a random prime we need many random bits.

Increasing Extension (1/2)

Before we consider how to construct a PRG we consider what the definition gives us:

- ▶ Suppose that there exists a PRG that extends its output by a **single bit**.
- ▶ This would **not be very useful** to us, e.g., to generate a random prime we need many random bits.
- ▶ Can we use the given PRG to construct another PRG which extends its output more?

Increasing Extension (2/2)

Construction. Let PRG be a pseudo-random generator. We let PRG_t be the algorithm that takes $s_{-1} \in \{0, 1\}^n$ as input, computes s_0, s_2, \dots, s_{t-1} and b_0, \dots, b_{t-1} as

$$(s_i, b_i) = \text{PRG}(s_{i-1})$$

and outputs (b_0, \dots, b_{t-1}) .

Increasing Extension (2/2)

Construction. Let PRG be a pseudo-random generator. We let PRG_t be the algorithm that takes $s_{-1} \in \{0, 1\}^n$ as input, computes s_0, s_2, \dots, s_{t-1} and b_0, \dots, b_{t-1} as

$$(s_i, b_i) = \text{PRG}(s_{i-1})$$

and outputs (b_0, \dots, b_{t-1}) .

Theorem. Let $p(n)$ be a polynomial and PRG a pseudo-random generator. Then $\text{PRG}_{p(n)}$ is a pseudo-random generator that on input $s \in \{0, 1\}^n$ outputs a string in $\{0, 1\}^{p(n)}$.

Increasing Extension (2/2)

Construction. Let PRG be a pseudo-random generator. We let PRG_t be the algorithm that takes $s_{-1} \in \{0, 1\}^n$ as input, computes s_0, s_2, \dots, s_{t-1} and b_0, \dots, b_{t-1} as

$$(s_i, b_i) = \text{PRG}(s_{i-1})$$

and outputs (b_0, \dots, b_{t-1}) .

Theorem. Let $p(n)$ be a polynomial and PRG a pseudo-random generator. Then $\text{PRG}_{p(n)}$ is a pseudo-random generator that on input $s \in \{0, 1\}^n$ outputs a string in $\{0, 1\}^{p(n)}$.

We can go on “forever”!

Random String From Random Oracle

Theorem. If $F : \{0, 1\}^n \rightarrow \{0, 1\}^m$ is a random function, then $(F(0), F(1), F(2), \dots, F(t-1))$ is an tm -bit string.

Random String From Random Oracle

Theorem. If $F : \{0, 1\}^n \rightarrow \{0, 1\}^m$ is a random function, then $(F(0), F(1), F(2), \dots, F(t-1))$ is an tm -bit string.

Can we do this using a pseudo-random function?

Pseudo-Random Function

Recall the definition of a pseudo-random function.

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

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

is negligible.

Pseudo-Random Generator From Pseudo-Random Function

Theorem. Let $\{F_K\}_{K \in \{0,1\}^k}$ be a pseudo-random function for a random choice of K . Then the PRG defined by:

$$\text{PRG}(s) = (F_s(0), F_s(1), F_s(2), \dots, F_s(t))$$

is a pseudo-random generator.

Pseudo-Random Function From Pseudo-Random Generator

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

Let $\text{Half}_b(r_0, r_1) = r_b$ for $(r_0, r_1) \in \{0, 1\}^{2k}$.

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

1. Computes $(r_0^0, r_1^0) = \text{PRG}(K)$.
2. Computes $r_{x_i}^i = \text{Half}_{x_i}(\text{PRG}(r_{x_{i-1}}^{i-1}))$ for $i = 1, \dots, n - 1$.
3. Outputs $r_{x_{n-1}}$.

Pseudo-Random Function From Pseudo-Random Generator

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

Let $\text{Half}_b(r_0, r_1) = r_b$ for $(r_0, r_1) \in \{0, 1\}^{2k}$.

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

1. Computes $(r_0^0, r_1^0) = \text{PRG}(K)$.
2. Computes $r_{x_i}^i = \text{Half}_{x_i}(\text{PRG}(r_{x_{i-1}}^{i-1}))$ for $i = 1, \dots, n - 1$.
3. Outputs $r_{x_{n-1}}$.

Theorem. F is a pseudo-random function.

One-Way Permutation

Definition. A family $F = \{f_n\}$ of **permutations** $f_n : \{0, 1\}^n \rightarrow \{0, 1\}^n$ 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 $f_n : \{0, 1\}^n \rightarrow \{0, 1\}^n$ and $B = \{b_n\}$ a family of “bits” $b_n : \{0, 1\}^n \rightarrow \{0, 1\}$. 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 function, there is a (possibly different) one-way function with a hardcore bit.

PRG from One-Way Permutation

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

PRG from One-Way Permutation

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

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

PRG From Any One-Way Function

Theorem. There is a construction PRG_f that is a PRG if f is a one-way function.

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 daemon with estimate of amount of entropy.
Fairly secure, but there are slight weaknesses
 - ▶ 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 CBC mode.
- ▶ Hashfunction with secret prefix and counter (our PRF→PRG construction).

Infamous Mistakes

- ▶ 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.
- ▶ Debian's OpenSSL commented out a critical part of the code that reduced the entropy of keys drastically!

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.