Cryptography 101: From Theory to Practice

Chapter 7 – Pseudorandom Generators

Rolf Oppliger

February 24, 2022

Terms of Use

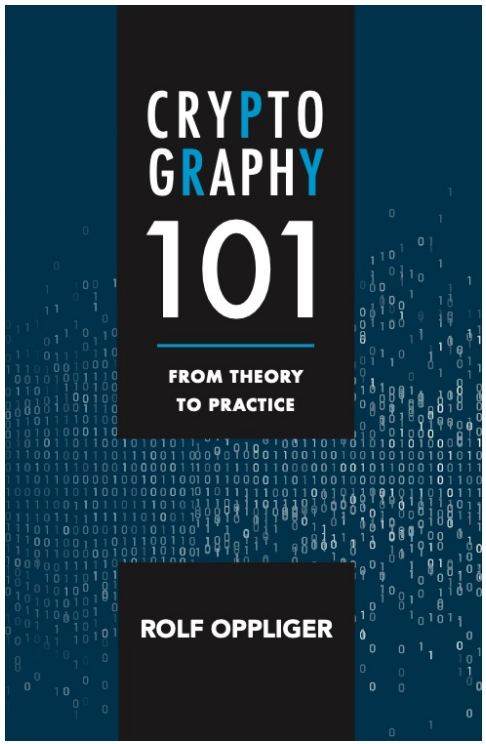
- This work is published with a CC BY-ND 4.0 license (cc by nd)
  - CC = Creative Commons (cc)
  - BY = Attribution (by)
  - ND = No Derivatives (nd)

whoami

rolf-oppliger.ch
rolf-oppliger.com

- Swiss National Cyber Security Centre NCSC (scientific employee)
- eSECURITY Technologies Rolf Oppliger (founder and owner)
- University of Zurich (adjunct professor)
- Artech House (author and series editor for information security and privacy)

Reference Book



ISBN 978-1-63081-846-3

`https://books.esecurity.ch/crypto101.html`

Challenge Me

Part II

SECRET KEY CRYPTOSYSTEMS

Outline

7. Pseudorandom Generators

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

– John von Neumann

7. Pseudorandom Generators

7. Pseudorandom Generators

7.1 Introduction

- According to Definition 2.7, a **PRG** is an efficiently computable function that takes as input a relatively short value of length n (i.e., seed) and generates as output a value of length $l(n) \gg n$ that appears to be random
- $L(n)$ is a stretch function, i.e., a function that stretches an n-bit value into a longer $l(n)$-bit value with $n < l(n) \leq \infty$
- A PRG is a secret key cryptosystem, because the seed can be seen as a secret key

7. Pseudorandom Generators

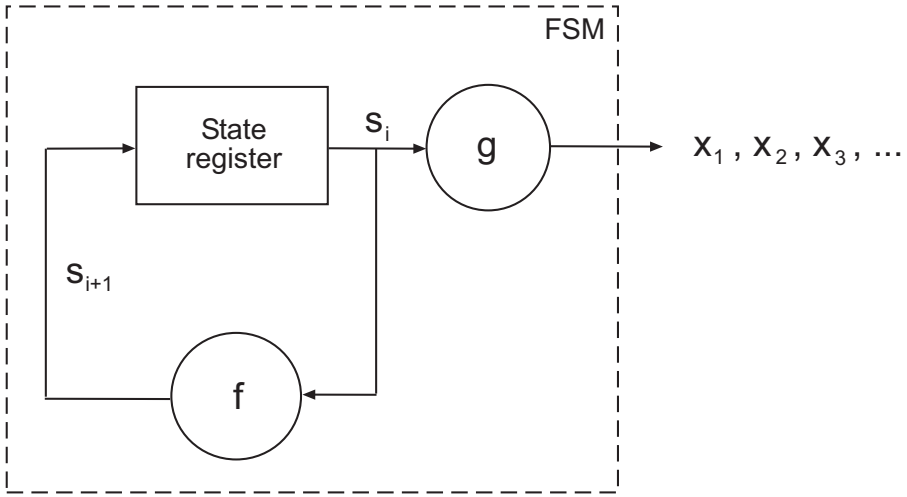
7.1 Introduction

- If the input and output values are bit sequences, then the PRG is a **PRBG**
- Mathematically, a PRBG G is a mapping from key space $\mathcal{K} = \{0,1\}^n$ to $\{0,1\}^{l(n)}$, i.e., $G : \mathcal{K} \longrightarrow \{0,1\}^{l(n)}$, for which the output appears to be random
- A proper definition of "appears to be random" is challenging, because a PRG operates deterministically
- This is in contrast a (true) random generator

7. Pseudorandom Generators

7.1 Introduction

- An idealized model of a PRG consists of
  - A state register (of length n)
  - A next-state or state-transition function f
  - An output function g



7. Pseudorandom Generators

7.1 Introduction

- The state register is initialized with an n-bit seed s_0
- In each cycle $i \geq 0$, the next-state function f computes s_{i+1} from s_i, i.e., $s_{i+1} = f(s_i)$, and s_i is subject to the output function g
- The result is $x_i = g(s_i)$, and the bit sequence

$$(x_i)_{i \geq 1} = x_1, x_2, x_3, \ldots$$

is the output of the PRG

7. Pseudorandom Generators

7.1 Introduction

- In the model, the function f operates recursively on the state register, and the seed is the only input value
- Some PRGs deviate from this idealized model by allowing the state register to be reseeded periodically
- This may be modeled by having a function f take into account additional sources of randomness (not illustrated)
- In this case, the distinction between a PRG and a true random generator gets fuzzy

7. Pseudorandom Generators

7.1 Introduction

- In a PRG, the number of states is finite and depends on n, i.e., the length of the state register
- There are at most $2^n - 1$ possible states
- This means that after at most $2^n - 1$ cycles, the register is in the same state and the output values start repeating themselves
- The sequence of output values is thus cyclic (with a potentially very large cycle)
- This is why one cannot require that the output of a PRG is truly random, but only that it appears to be so

7. Pseudorandom Generators

7.1 Introduction

- A minimal security requirement for a PRG is that n is sufficiently large so that an exhaustive search over all $2^n - 1$ possible states is computationally infeasible
- Also, the output bit sequence must pass all relevant statistical randomness tests
- One must be cautious here, because passing statistical randomness tests is a necessary but usually not sufficient requirement for a PRG to be (cryptographically) secure

7. Pseudorandom Generators

7.1 Introduction

- There are PRGs that pass most statistical randomness tests but are inappropriate for cryptographic use
  - PRGs that employ the binary expansion of numbers like $\sqrt{2}$, $\sqrt{3}$, or $\sqrt{5}$
  - Linear congruential generators that take as input a seed $x_0 = s_0$ and three integer parameters $a, b, n \in \mathbb{N}$ with $a, b < n$, and that use the linear recurrence

$$x_i = (ax_{i-1} + b) \bmod n$$

to recursively generate an output sequence $(x_i)_{i \geq 1}$

7. Pseudorandom Generators

7.2 Exemplary Constructions

- It is sometimes argued that a PRG can be built from a one-way function f by randomly selecting a seed s_0 and generating the output sequence

$$(x_i)_{i \geq 1} \quad = \quad f(s_0), f(s_0 + 1), f(s_0 + 2), f(s_0 + 3), \ldots$$

- The output values need not have good randomness characteristics
- If, for example, g is a one-way function and f extends g by appending a 1, i.e., $f(x) = g(x) \| 1$, then f is still one-way, but it outputs values that all end with a 1

7. Pseudorandom Generators

7.2 Exemplary Constructions

- More involved constructions are required to build a PRG from a one-way function f
  - In each iteration, only a hard-core predicate of f is used (see below)
  - The function f is required to be pseudorandom (see next chapter)
- Pseudorandomness is an inherently different property than one-wayness (the construction therefore works for pseudorandom functions but not for one-way functions)

7. Pseudorandom Generators

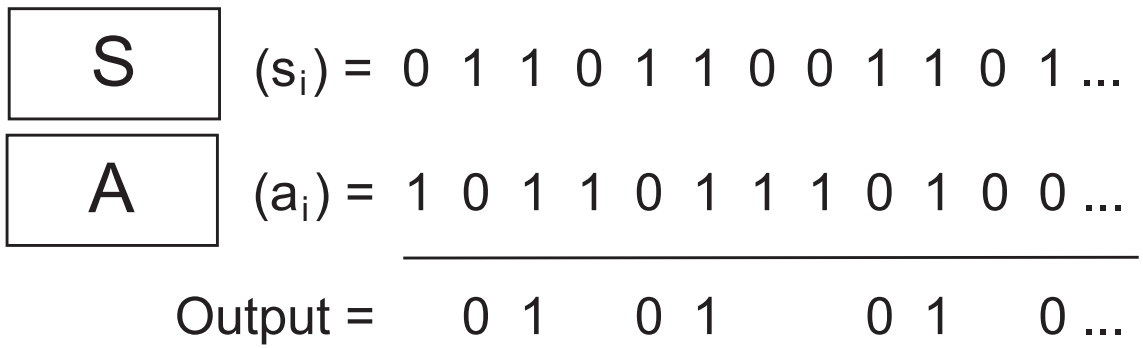
7.2 Exemplary Constructions

- In the past, people have tried to build PRGs from **linear feedback shift registers (LFSRs)**
- Using a single LFSR has turned out to be insufficient
- So people have tried to use multiple LFSRs with irregular clocking, e.g., A5/1 and A5/2 (GSM), CSS (DVD encryption), or E0 (Bluetooth encryption)
- Most of these LFSR-based PRGs are "insecure"
- More secure variants
  - Shrinking generator
  - Self-shrinking generator

7. Pseudorandom Generators

7.2 Exemplary Constructions

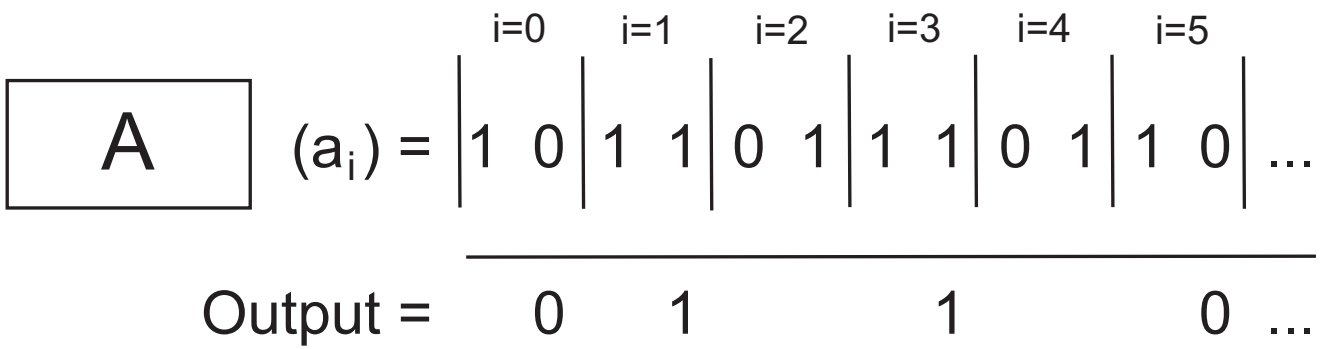
- The **shrinking generator** employs 2 LFSRs A and S to generate two sequences $(a_i)_{i \geq 0}$ and $(s_i)_{i \geq 0}$
- In clock cycle $i \geq 0$, the generator outputs a_i if and only if $s_i = 1$ (otherwise, a_i is discarded)



7. Pseudorandom Generators

7.2 Exemplary Constructions

- The **self-shrinking generator** employs only one LFSR A to generate the sequence $(a_i)_{i \geq 0}$
- In clock cycle i, the generator outputs a_{2i+1} if and only if $a_{2i} = 1$ (otherwise, a_{2i+1} is discarded)



7. Pseudorandom Generators

7.2 Exemplary Constructions

- LFSR-based PRGs are not so popular anymore, mainly because they depend on hardware
- Most people prefer software implementations
- A practically relevant PRG is specified in ANSI X9.17 (with DES or 3DES)

Algorithm 7.1 ANSI X9.17 PRG

(s_0, k, n)

$I = E_k(D)$
$s = s_0$
for $i = 1$ to n do
  $x_i = E_k(I \oplus s)$
  $s = E_k(x_i \oplus I)$
  output x_i

$(x_1, x_2, \ldots, x_n)$

7. Pseudorandom Generators

7.2 Exemplary Constructions

- Besides ANSI X9.17, there are several other PRGs used in the field (e.g., Yarrow, Fortuna, ...)
- There are only a few security analyses for these PRGs
- In some literature, such they are called **practically strong**
- A practically strong PRG is designed in an ad hoc way but believed to resist known attacks
- This is different from a cryptographically secure PRG

7. Pseudorandom Generators

7.3 Cryptographically Secure PRGs

- There are several possibilities to formally define the cryptographical strength (and security) of a PRG
- Historically, the first definition was proposed by Manuel Blum and Silvio Micali in the early 1980s
- They argued that a PRG is **cryptographically secure**, if an adversary — after having seen a sequence of output values — is not able to predict the next value with a success probability that is better than guessing (i.e., next-bit test)
- They also proposed a cryptographically secure PRG that is based on the DLP

7. Pseudorandom Generators

7.3 Cryptographically Secure PRGs

- Shortly after this seminal work, Manuel Blum – together with Leonore Blum and Michael Shub – proposed the **BBS PRG** or **squaring generator**
- It is cryptographically secure assuming the intractability of the quadratic residuosity problem (QRP)
- The BBS PRG is still the yardstick for cryptographically secure PRGs (see below)

7. Pseudorandom Generators

7.3 Cryptographically Secure PRGs

- A cryptographically secure PRG is also **perfect** in the sense that no PPT algorithm can tell whether an n-bit string has been sampled uniformly at random from $\{0,1\}^n$ or generated with the PRG (using a proper seed) with a success probability that is better than guessing
- This means that a PRG that passes the next-bit test is perfect in the sense that it passes all polynomial-time (statistical) tests to distinguish it from a true random generator

7. Pseudorandom Generators

7.3 Cryptographically Secure PRGs

- The (mathematical) tool to argue about this notion of security is **computational indistinguishability**
- Formally, a **probability ensemble** is a family of probability distributions (or random variables) $X = \{X_i\}_{i \in I}$, where I is an index set and each X_i is a distinct probability distribution (sometimes also denoted P_{X_i})
- Typically, $I = \mathbb{N}$, and hence there is an X_n for every $n \in \mathbb{N}$

7. Pseudorandom Generators

7.3 Cryptographically Secure PRGs

- Let

$$X = \{X_n\}_{n \in \mathbb{N}} = \{X_1, X_2, X_3, \ldots\}$$

and

$$Y = \{Y_n\}_{n \in \mathbb{N}} = \{Y_1, Y_2, Y_3, \ldots\}$$

be two probability ensembles, i.e., for all $n \in \mathbb{N}$ X_n and Y_n refer to probability distributions on $\{0,1\}^n$

- $t \leftarrow X_n$ ($t \leftarrow Y_n$) means that t is sampled according to the probability distribution X_n (Y_n)

7. Pseudorandom Generators

7.3 Cryptographically Secure PRGs

- X is **polytime indistinguishable** from Y, if for every PPT algorithm A and every polynomial p, there exists a $n_0 \in \mathbb{N}$ such that for all $n > n_0$

$$\left| \Pr_{t \leftarrow X_n}[A(t) = 1] - \Pr_{t \leftarrow Y_n}[A(t) = 1] \right| \leq \frac{1}{p(n)}$$

- This means that for sufficiently large t, no PPT algorithm A can distinguish whether it is sampled according to X_n or Y_n
- In some literature, A is called a **polynomial-time statistical test** or **distinguisher** (sometimes denoted D)

7. Pseudorandom Generators

7.3 Cryptographically Secure PRGs

- Using this notion of indistinguishability, pesudorandomness can be defined precisely
- $X = \{X_n\}$ is **pseudorandom** if it is polytime indistinguishable from $U = \{U_n\}$, i.e., the uniform probability distribution on $\{0,1\}^n$ for $n \in \mathbb{N}$
- This means that for every PPT algorithm A and every polynomial p, there exists a $n_0 \in \mathbb{N}$ such that for all $n > n_0$

$$\left| \Pr_{t \leftarrow X_n}[A(t) = 1] - \Pr_{t \leftarrow U_n}[A(t) = 1] \right| \leq \frac{1}{p(n)}$$

7. Pseudorandom Generators

7.3 Cryptographically Secure PRGs

Let G be a PRG with stretch function $l : \mathbb{N} \to \mathbb{N}$ and $l(n) > n$ for $n \in \mathbb{N}$, and $\{G_n\}$ be the distribution defined as the $l(n)$-bit output of G on a seed that is sampled uniformly at random from $\{0,1\}^n$

Definition 7.1 (Cryptographically secure PRG)

G is *cryptographically secure* if $\{G_n\}$ is pseudorandom, i.e., it is polytime indistinguishable from $\{U_{l(n)}\}$

7. Pseudorandom Generators

7.3 Cryptographically Secure PRGs

This means that that for every PPT algorithm A and every polynomial p, there exists a $n_0 \in \mathbb{N}^+$ such that for all $n > n_0$

$$\left| \Pr_{t \leftarrow U_n}[A(G(t)) = 1] \quad - \quad \Pr_{t \leftarrow U_{l(n)}}[A(t) = 1] \right| \leq \frac{1}{p(n)}$$

The leftside term stands for the PRG advantage of A with respect to PRG G, denoted $\mathrm{Adv}_{\mathrm{PRG}}[A, G]$

7. Pseudorandom Generators

7.3 Cryptographically Secure PRGs

- To argue about the security of G, one must be interested in the PPT algorithm A with maximal PRG advantage
- This yields the PRG advantage of G that is defined as

$$\mathrm{Adv}_{\mathrm{PRG}}[G] = \max_{A}\{\mathrm{Adv}_{\mathrm{PRG}}[A, G]\}$$

- G is secure, if $\mathrm{Adv}_{\mathrm{PRG}}[G]$ is negligible, i.e., for every polynomial p, there exists a $n_0 \in \mathbb{N}$ such that for all $n > n_0$

$$\mathrm{Adv}_{\mathrm{PRG}}[G] \leq \frac{1}{p(n)}$$

7. Pseudorandom Generators

7.3 Cryptographically Secure PRGs

- If f is a one-way function with hard-core predicate B, then the following PRG G with seed s_0 is cryptographically secure:

$$G(s_0) = B(f(s_0)), B(f^2(s_0)), \ldots, B(f^{l(n)}(s_0))$$

- Talking in terms of the idealized model of a PRG, the state register is initialized with s_0, the next-state function f is the one-way function, and the output function g refers to the hard-core predicate B
- This idea is used in many cryptographically secure PRGs

7. Pseudorandom Generators

7.3 Cryptographically Secure PRGs

Algorithm 7.2 The Blum-Micali PRG

(p, g)

$x_0 \xleftarrow{r} \mathbb{Z}_p^*$
for $i = 1$ to ∞ do
  $x_i = g^{x_{i-1}} \bmod p$
  $b_i = msb(x_i)$
  output b_i

$(b_i)_{i \geq 1}$

Algorithm 7.3 The RSA PRG

(n, e)

$x_0 \xleftarrow{r} \mathbb{Z}_n^*$
for $i = 1$ to ∞ do
  $x_i = x_{i-1}^e \bmod n$
  $b_i = lsb(x_i)$
  output b_i

$(b_i)_{i \geq 1}$

7. Pseudorandom Generators

7.3 Cryptographically Secure PRGs

Algorithm 7.4 The BBS PRG

(n)

$x_0 \xleftarrow{r} \mathbb{Z}_n^*$
for $i = 1$ to ∞ do
  $x_i = x_{i-1}^2 \bmod n$
  $b_i = lsb(x_i)$
  output b_i

$(b_i)_{i \geq 1}$

- The BBS PRG has the practically relevant property that x_i can be computed directly for $i \geq 1$ if one knows the factorization of n

$$x_i = x_0^{(2^i) \bmod ((p-1)(q-1))}$$

7. Pseudorandom Generators

7.4 Final Remarks

- All PRGs in use today critically assume that their internal state can be kept secret
- In practice, it may still happen that the adversary can acquire the internal state
- This may make it necessary to periodically reseed the state
- Some practically strong PRGs take this into account and have an accumulator that collects and pools entropy from various sources to periodically reseed the generator

7. Pseudorandom Generators

7.4 Final Remarks

- There are many applications of PRGs
- If a lot of keying material is required, then they can complement (rather than replace) true random bit generators
- If a PRG is used to derive keying material from a single master key or password, then it is called a **key derivation function (KDF)** or a **mask generation function (MGF)**

7. Pseudorandom Generators

7.4 Final Remarks

- Technically speaking, a KDF (MGF) can be implemented with a function f_k (from PRF family F) as follows:

$$KDF(k, c, l) = f_k(c \parallel 0) \parallel f_k(c \parallel 1) \parallel \ldots \parallel f_k(c \parallel n-1)$$

- In this notation, c is a context string (acting as "salt") and l is the number of bytes that need to be generated
- If b is the output length of f_k, then $n = \lceil l/b \rceil$

7. Pseudorandom Generators

7.4 Final Remarks

- The security of this construction requires k to be uniform in $\mathcal{K}$
- This requirement is crucial and may not always be fulfilled
- For example, if k is the outcome of a key agreement, then k may be biased or originate from a relatively small subset of $\mathcal{K}$
- Some preprocessing may be required here (to extract a uniform and pseudorandom key from the source key)
- There are standards that serve this purpose, such as KDF1 to KDF4 and the **HMAC-based extract-and-expand key derivation function (HKDF)**

7. Pseudorandom Generators

7.4 Final Remarks

- Another example where k may not be uniform in $\mathcal{K}$ is when a user selects a password
- User-selected passwords do not provide a lot of entropy
- For this use case, there are special-purpose **password-based key derivation functions (PBKDF)**, such as PBKDF1 and PBKDF2
- These functions are typically slowed down artificially (e.g., through iteration) to mitigate (offline) password guessing attacks

7. Pseudorandom Generators

7.4 Final Remarks

- Another approach to mitigate (offline) password guessing attacks is to make the PBKDF memory-hard (e.g., scrypt and Balloon)
- The **Password Hashing Competition (PHC)** was a privately initiated competition for a standardized PBKDF that took place from 2013 to 2015
- The final winner of the PHC was Argon2, but special recognition was also given to Catena, Lyra2, yescrypt, and Makwa

Questions and Answers

Thank you for your attention