Pseudorandomness and Stream Ciphers

Randomness

Almost all cryptographic algorithms have to use randomness at one point. An important, practical question to ask is:
Where does this randomness come from?

Short answer:
Cryptographic application extract randomness from hard to predict physical phenomena.

Long answer:
In the universe, most things are predictable (deterministic) if you know **all** the initial conditions. For example, if you throw a die and you know the air friction, the weight of the die, the force you're throwing with, etc you could predict what side it's going to land on. However, any small mistake / ommision in the initial conditions can lead to totally different results (check out the butterfly effect and chaos theory). This assumption that is **impractical to comptute** and therefore predict physical phenomena is what cryptographic randomness is based on.

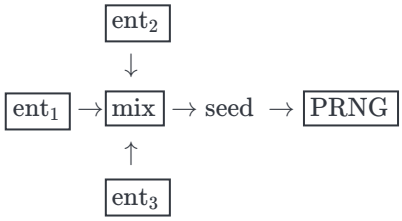
The RNG based on physical processes are called True random number generators and have different ways to gather randomness:

- Measuring Hardware interrupts
- Measuring Thermal noise
- Mouse movements
- Device temperature

Some of them are better and some of them are worse. Usually the OS cleans up these noises and **mixes** them up to get a good random number (mixing is done by hashing + xor-ing)

However, the bad news is that this *true* randomness is often not enough (in terms of size) for our cryptographic applications. To circumvent this we will use **PRNG** (pseudorandom number generators), discussed below, whose purpose is to take a truly randomly generated *seed* and **deterministically* output something that looks random.

A common workflow diagram:

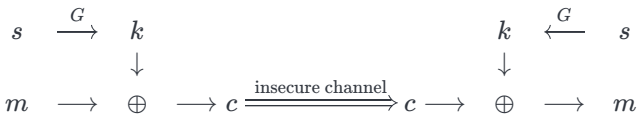


Pseudorandomness

The idea of secret key encryption is to provide confidentiality (hide a message) by letting two parties encrypt and decrypt different messages using the **same** secret key (known only by the parties).
There are 2 main types of ciphers that can achieve this

- **Stream ciphers** -- Intuitively, they expand a small secret key into a big secret key and use it as an one time pad. This expansion is done by a **pseudorandom generator**.
- **Block ciphers** -- Intuitively, they scramble the input. They ususally act as **pseudorandom permutations**.

Stream ciphers



Block ciphers

The main proprieties of these pseudorandom objects is that the results must be **computationally indistinguishable** from random. Intuitively, if an adversary receives an output of a PRNG as a string and another truly random string the adversary mustn't be able to distinguish between them.

Pseudorandom generators

Let $\mathcal{S} = \{0,1\}^l$ be the **seed space** of l-bit strings and $\mathcal{R} = \{0,1\}^L$ be the output space of L bit strings, with $l \ll L$.

Pseudorandom generators -- definition

An **efficient deterministic algorithm** $G : \{0,1\}^l \to \{0,1\}^L$ is called a **pseudorandom number generator**. The output $G(s)$ must be **computationally indistinguishable** from a random string $r \in \{0,1\}^L$

Intuition: Given a seed $s \in \{0,1\}^l$ where $l \ll L$ we want to stretch the seed into a a longer key.
Remark: Since $l < L$ the PRNG cannot achieve perfect security. However semantic security is enough for us.

Computational indistinguishability -- Definition and attack game

Given the output of a PRNG and a truly random string an efficient adversary shouldn't be able to distinguish between them.
Attack game:

- The challenger samples the seed $s \xleftarrow{R} \mathcal{S}$ and computes a string $r_0 = G(s)$ and samples another string $r_1 \xleftarrow{R} \mathcal{R}$.
- Then it flips a bit b and gives one of them to the adversary based on the bit value ($b = 0 \Rightarrow r_0,\ b = 1 \Rightarrow r_1$).
- The adversary is tasked to say if he received r_0 or r_1 (guess the bit). He wins if he has a high probability of distinguishing them.

A PRNG G is secure if the advantage is negligible $\forall$ efficient adversaries.

The definition can be expanded to probability distributions:

Computantional indistinguishability 2 -- attack game

Let P_1, P_2 be two distributions over $\{0,1\}^n$. We say that P_1, P_2 are **computationally indisinguishable** if $\forall$ efficient statistical tests A

$$|Pr[A(x) = 1] - Pr[A(x) = 1] < \epsilon$$
$$\underset{x \leftarrow P_1}{} \qquad \underset{x \leftarrow P_2}{}$$

where ϵ is **negligible**.

PRNG security

Statistical tests

We want the PRNG output distribution to be indistinguishable from the true random distribution. Usually, testing for this propriety is done is done using **statistical tests**:

$$A : \mathcal{R} \to \{0,1\}$$

An **effective** statistical test is one that given $r_0 = G(s)$ and $r_1 \xleftarrow{R} \mathcal{R}$ the probability of outputing 1 for r_0 and r_1 differs significantly.

Examples of statistical tests

- $\#(1)$ - Number of ones. For a truly random string we expect $\#1 \approx L/2$.
- $|\#(0) - \#(1)|$ (num of 0s - num of 1)
- $\#(00)$ -- Count the pair of 00
- longest sequence of 0

Let $G : \mathcal{S} \to \{0,1\}^L$ and define **Advantage** as:

$$\mathrm{Adv}_{PRNG}(A, G) = |Pr[A(G(s)) = 1] - Pr[A(r) = 1]| \in [0,1]$$

- If $\mathrm{Adv} \to 1 \Rightarrow A$ can distinguish from random
- If $\mathrm{Adv} \to 0 \Rightarrow A$ can't distinguish from random

Example of a bad PRNG:
Suppose

- $msb(G(s)) = 1$ for $2/3$ of $s \in \mathcal{S}$
- $A(x) = 1 \iff msb(x) = 1$

Then

- $\mathrm{Adv}_{PRNG}(A, G) = |Pr[A(G(s)) = 1] - Pr[A(r) = 1]| = |2/3 - 1/2| = 1/6$

PRNG security -- Definition

A PRNG is secure if for all effiecient statistical tests A the $\mathrm{Adv}_{PRNG}(A, G)$ is negligible:

$$\mathrm{Adv}_{PRNG}(A, G) = |Pr[A(G(s)) = 1] - Pr[A(r) = 1]| < \epsilon$$

PRNG Unpredictability -- Definition and attack game

For a given PRNG G, if an attacker is given the first i bits of G's output it cannot predict the next ($i+1$th) bit with probability significantly better than $1/2$. If we can effectively predict the next bit we immediately have an **effective statistical test**.
Attack game:

- The adversary sends an index i to the challenger
- The challenger computes $s \xleftarrow{R} \mathcal{S}, r = G(s)$. and sends the first i bits $r[0..i-1]$ to the adversary
- The adversary wins if he guesses the next bit $r[i]$ with a significant probability $> 1/2$

A PRNG is **unpredictable** if the advantage of any efficient adversary is negligible.

Theorem

If the generator G is secure $\Rightarrow$ a PRNG based on it is unpredictable.

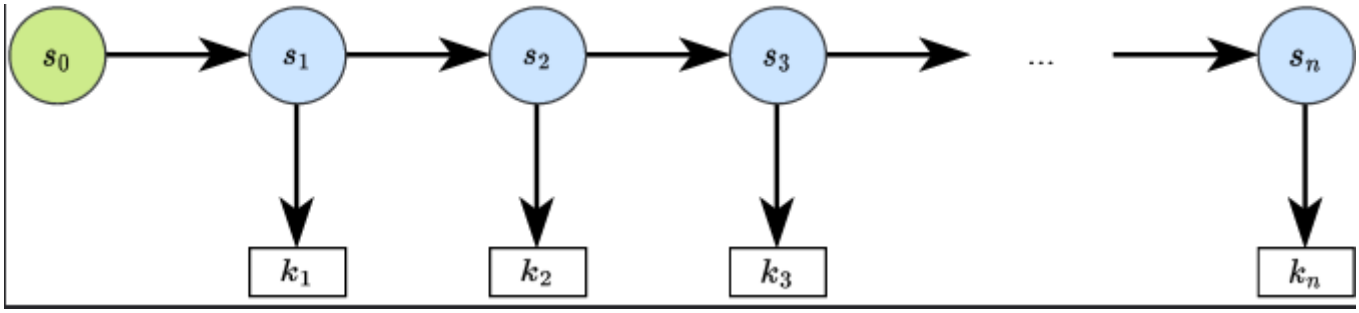
Theorem

An unpredictable PRNG is secure.

Composing PRNG

The PRNG's output is an L bit string, however in practice we usually need more than L bits. To do this we use many applications of G with many seeds $(s_1, ...s_n)$ and concatenate the results. This is called a parallel construction. A sequential construction is given a seed s_0 and generates the next string and the seed for the next use of the PRNG.

Usually, pseudorandom generators start from a seed and compute some internal **state**. This internal state is then used to compute the random number and the next internal state ($s_n \rightarrow s_{n+1}$).



There are 2 desirable properties that we want to satisfy:

1. **Forward secrecy** - Given a later state an attacker must not be able to compute a previous state / random number.
2. **Backwards secrecy** Given a current / previous state s_n an attacker must not be able to compute a future state / random number

Pseudorandom functions

Pseudorandom function -- Definition

A pseudo-random function (PRF) $F : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$ is a deterministic algorithm that has two inputs: key $k \in \mathcal{K}$ and an input data block $x \in \mathcal{X}$ and has the output $y := F(k, x)$. For a randomly chosen key k the PRF F must look like a random function from $\mathcal{X}$ to $\mathcal{Y}$

PRF Security

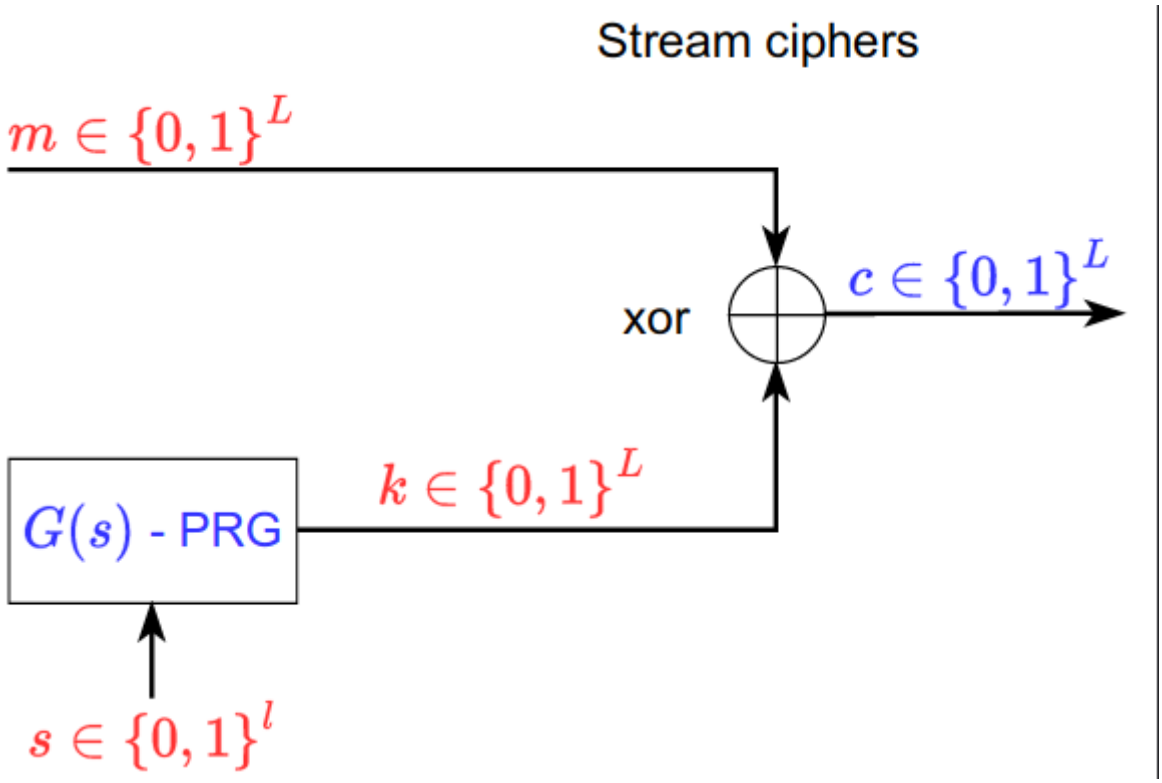
A PRF F is secure if it's indistinguishable from a random function (The advantage for all efficient adversaries is negligible)

Weak security

A PRF F is secure if it's indistinguishable from a random function when the queries are limited(The advantage for all efficient adversaries is negligible)

Stream ciphers

The idea is to make our ciphertext as indistinguishable from random as possible. Since the message is fixed we want to generate the key in a smart way to make it as random as possible. We are using pseudorandom generators to generate the key and `xor`-ing it with the message like in the OTP setting.



We use a seed to generate a key (Pseudorandom generator).

$$\begin{array}{ccccccccccc} s & \xrightarrow{G} & k & & & & k & \xleftarrow{G} & s \\ & & \downarrow & & & & \downarrow & & \\ m & \longrightarrow & \oplus & \longrightarrow & c & \xRightarrow{\text{insecure channel}} & c & \longrightarrow & \oplus & \longrightarrow & m \end{array}$$

$$k = G(s)$$

where s = seed. Let (E, D) be a cipher:

$$c = E(k, m) = E(G(s), m) = G(s) \oplus m$$
$$m = D(k, c) = D(G(s), c) = G(s) \oplus c$$

Ex: We put a 128-bit seed into a function G that outputs a 2048-bit key. We `xor` this key with the message m

Remark

- Stream ciphers cannot have perfect secrecy since $|\mathcal{M}| < |\mathcal{K}|$. But they are semantically secure.

Theorem

G is a secure PRNG => the Stream cipher based on it is semantically secure

Proof sketch

In order to prove that the stream cipher is secure we must reduce the security of breaking the stream cipher to the security of breaking the PRNG. **Reducing** the security of one problem to another problem is a **common pattern** to do when building and analyzing primitives in cryptography.

So, for a stream cipher (E, D) and a PRNG G we want to prove that

$$\text{Adv}_{SS}[\mathcal{A}, (E, D)] = \text{Adv}_{PRNG}[\mathcal{B}, G]$$

$\mathcal{B}$ is the adversary designed to attack G. So he receives either

- $r_0 = G(s)$ where $s \xleftarrow{R} \mathcal{S}$ or
- $r_1 \xleftarrow{R} \mathcal{R}$

Then $\mathcal{B}$ acts as a challenger for $\mathcal{A}$ in the bit flipping SS game:

- $\mathcal{B}$ has 2 messages m_0, m_1
- Flips a random bit b
- Based on b, it encrypts using `xor`: $c = m_b \oplus r$ and sends the encryption c to $\mathcal{A}$.

$\mathcal{A}$ will output the bit $\hat{b}$ based on what he things the initial message was (m_0 or m_1). He wins if $\hat{b} = b$ (guesses the bit b). $\mathcal{B}$ outputs 1 if $\hat{b} = b$ or 0 if $\hat{b} \neq b$.

Now let's analyze the cases

1. $\mathcal{B}$ receives r_1 (the random string). Then $\mathcal{A}$ attacks a variable length OTP, which is semantically secure: $\mathcal{A}$ guess is random, so the probability of guessing the bit is $p_0 = 1/2$, so advantage is $\text{Adv}_{SS}[\mathcal{A}, (E, D)] = 1/2 - 1/2 = 0$
2. $\mathcal{B}$ receives $r_0 = G(s)$. Let p_1 be the probability of $\mathcal{A}$ guessing the bit sent to him by $\mathcal{B}$ ($\hat{b} = b$). Then the advantage is $\text{Adv}_{SS}[\mathcal{A}, (E, D)] = p_1 - 1/2$.

We have:

$$\text{Adv}_{PRNG}[\mathcal{B}, G] = |p_1 - p_0| = p_1 - 1/2$$

If G is secure then this advantage is negligible. If this advantage is negligible then $p_1 \approx 1/2$ and $\text{Adv}_{SS}[\mathcal{A}, (E, D)]$ is neglijible

More Resources

- wikipedia csprng
- pseudorandom functions
- wikipedia stream cipher
- https://www.youtube.com/watch?v=rAFNmO-4CIA - Another short explanation
- https://www.youtube.com/watch?v=W39KqX0ZTbU - Another long explanation
- http://netlab.cs.ucla.edu/wiki/files/shannon1949.pdf - Shannon security paper