Cryptography

5. Pseudorandom Generators

5.1 Introduction

A pseudorandom generator(PRG) is a deterministic function that

- stretches a "short", truly random input (seed) to a long sequence of pseudorandom bits
- directly gives a stream cipher
- perhaps the most basic toll of cryptography

Definition:

A PRG is a deterministic function $G : \{0,1\}^\lambda \rightarrow \{0,1\}^{\lambda+l}$ that satisfies:

$L_{PRG-real}$
$Query()$:
s *(seed)* $\leftarrow \{0,1\}^\lambda$
return G(s)

$L_{PRG-rand}$
$Query()$:
$r \leftarrow \{0,1\}^{\lambda+l}$
return r

Example:

A length-doubling PRG:

$$\begin{aligned} G : \{0,1\}^\lambda &\rightarrow \{0,1\}^{2\lambda} \\ \mid dom(G) \mid &= 2^\lambda \\ \mid range(G) \mid &\leq 2^{2\lambda} \end{aligned}$$

- Even if $G(s)$ for random $s \leftarrow \{0,1\}^\lambda$ outputs only a small subset of $\{0,1\}^{2\lambda}$, no efficient (ppt) algorithm can distinguish $G(s)$ from a truly random 2λ-bit string.

(Counter-)Example:

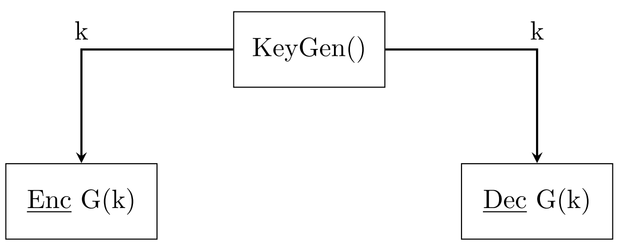
Java: java.util.random
⋆ new Random(seed s)
linear congruential generator:

$$\begin{aligned} s_0 &:= seed \\ s_{i+1} &:= (s_i \cdot a + c) mod n \end{aligned}$$

Insecure: NOT A PRG !!!

5.3 Stream cipher - Encryption from a PRG

- A stream cipher performs encryption like the OTP, but:
  - uses a short key
  - with computational security (conditional security)
- Prototype for a symmetric cryptosystem; also fastest



Remarks

- Encryption and decryption are stateful
- Requires PRG outputs the key stream in pieces
- Stream cipher directly malleable and provides no integrity or authenticity
  $\rightarrow$ needs additional tools to ensure integrity ($\rightarrow$ MAC)

One-time encryption with computational security

Definition

An encritpion scheme $\Sigma = (KeyGen,\ Enc,\ Dec)$ has computational one-time secrecy if

$$L^{\Sigma}_{ots-L} \ \tilde{\approx}\ L^{\Sigma}_{ots-R}$$

$$\begin{array}{l}
L_{ots-L} \\ \hline
\underline{Eavesdrop(m_L, m_R):} \\
\quad k \leftarrow \Sigma.KeyGen() \\
\quad c := \Sigma.Enc(k, m_L) \\
\quad \underline{\text{return}}\ \text{c} \\ \hline
\end{array}
\qquad
\begin{array}{l}
L_{ots-R} \\ \hline
\underline{Eavesdrop(m_L, m_R):} \\
\quad k \leftarrow \Sigma.KeyGen() \\
\quad c := \Sigma.Enc(k, m_R) \\
\quad \underline{\text{return}}\ \text{c} \\ \hline
\end{array}$$

Stream cipher s using a PRG G

$$\begin{aligned}
K &= \{0,1\}^{\lambda} \\
M &= \{0,1\}^{\lambda+l} \\
C &= \{0,1\}^{\lambda+l}
\end{aligned}$$

$$\begin{array}{l}
\underline{KeyGen():} \\
\quad k \leftarrow \{0,1\}^{\lambda} \\
\quad \underline{\text{return}}\ \text{k}
\end{array}
\qquad
\begin{array}{l}
\underline{Enc(k,m):} \\
\quad \underline{\text{return}}\ G(k) \oplus m
\end{array}
\qquad
\begin{array}{l}
\underline{Enc(k,m):} \\
\quad \underline{\text{return}}\ G(k) \oplus c
\end{array}$$

Theorem:

<u>If</u> G is a secure PRG, <u>then</u> stream cipher s has computational one-time secrecy:

$$i.e.\ L^{\Sigma}_{ots-L} \approx L^{\Sigma}_{ots-R}$$

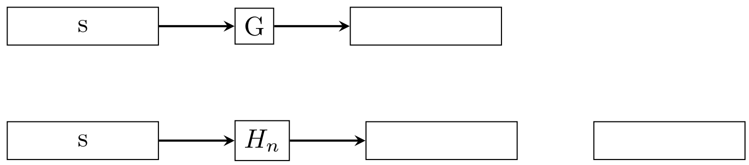
$$\begin{array}{l}
L_{ots-L} \\ \hline
\underline{Eavesdrop(m_L, m_R):} \\
\quad k \leftarrow \Sigma.KeyGen() \\
\quad c := \Sigma.Enc(k, m_L) \\
\quad \underline{\text{return}}\ \text{c} \\ \hline
\end{array}
\equiv
\begin{array}{l}
\underline{Eavesdrop(m_L, m_R):} \\
\quad k \leftarrow \{0,1\}^{\lambda} \\
\quad c := G(k) \oplus m_L \\
\quad \underline{\text{return}}\ \text{c} \\ \hline
\end{array}
\equiv
\begin{array}{l}
\underline{Eavesdrop(m_L, m_R):} \\
\quad k \leftarrow \{0,1\}^{\lambda} \\
\quad \text{z} := \text{G(k)} \\
\quad c := z \oplus m_L \\
\quad \underline{\text{return}}\ \text{c} \\ \hline
\end{array}$$

$$\approx
\begin{array}{l}
\underline{Eavesdrop(m_L, m_R):} \\
\quad z \leftarrow \{0,1\}^{\lambda+l} \\
\quad c := z \oplus m_L \\
\quad \underline{\text{return}}\ \text{c} \\ \hline
\end{array}
\equiv
\begin{array}{l}
\underline{Eavesdrop(m_L, m_R):} \\
\quad z \leftarrow \{0,1\}^{\lambda+l} \\
\quad c := z \oplus m_R \\
\quad \underline{\text{return}}\ \text{c} \\ \hline
\end{array}
\approx
\begin{array}{l}
\underline{Eavesdrop(m_L, m_R):} \\
\quad k \leftarrow \{0,1\}^{\lambda} \\
\quad \text{z} := \text{G(k)} \\
\quad c := z \oplus m_R \\
\quad \underline{\text{return}}\ \text{c} \\ \hline
\end{array}$$

$$\equiv
\begin{array}{l}
\underline{Eavesdrop(m_L, m_R):} \\
\quad k \leftarrow \{0,1\}^{\lambda} \\
\quad c := G(k) \oplus m_R \\
\quad \underline{\text{return}}\ \text{c} \\ \hline
\end{array}
\equiv
\begin{array}{l}
L_{ots-R} \\ \hline
\underline{Eavesdrop(m_L, m_R):} \\
\quad k \leftarrow \Sigma.KeyGen() \\
\quad c := \Sigma.Enc(k, m_R) \\
\quad \underline{\text{return}}\ \text{c} \\ \hline
\end{array}$$

5.5 Extending the stretch of a PRG

1. A length-doubling PRG



2. Extension of length doubling G to (n+1) fold expansion:

$\underline{H_n(s \in \{0,1\}^\lambda)}$
$s_0 := s$
$\underline{\text{for}}\ i := 1, ..., n\ \underline{\text{do}}$
$\quad t_i \| s_i := G(s_{i-1}$
$\underline{\text{return}}\ t_i \| ... \| t_n \| s_n$

We want to show:

$L^H_{PRG-real}$
$\underline{Query():}$
$s\ (seed) \leftarrow \{0,1\}^\lambda$
$\underline{\text{return}}\ H_n(s)$

$\approx$

$L^H_{PRG-rand}$
$\underline{Query():}$
$r \leftarrow \{0,1\}^{\lambda+l}$
$\underline{\text{return}}$ r