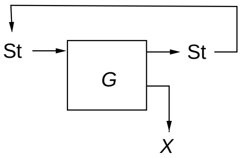
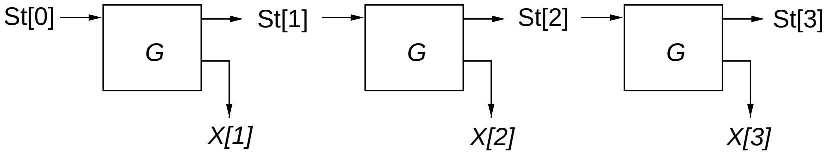
STREAM CIPHERS and PRGs

Stateful Generators

Initially, St is a random seed



Operation:



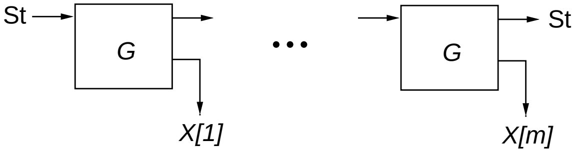
$X[1]X[2]X[3]...$ is the output sequence and should be "pseudorandom".

Notation

$$(X[1] \ldots X[m], \mathrm{St}) \leftarrow G(\mathrm{St}, m)$$

means we

- Run G with starting state St for m steps
- Let $X[1] \ldots X[m]$ be the output blocks produced
- Let St be the updated state



Usage for Encryption

Alice maintains a state St_A and Bob maintains a state St_B.
Initially: $\mathrm{St}_A = \mathrm{St}_B$ is a random seed.

$\mathcal{E}(M[1] \dots M[m])$ $(X[1] \dots X[m], \mathrm{St}_A) \leftarrow G(\mathrm{St}_A, m)$ for $i = 1, \dots, m$ do $\quad C[i] \leftarrow X[i] \oplus M[i]$	$\mathcal{D}(C[1] \dots C[m])$ $(X[1] \dots X[m], \mathrm{St}_B) \leftarrow G(\mathrm{St}_B, m)$ for $i = 1, \dots, m$ do $\quad M[i] \leftarrow X[i] \oplus C[i]$

Note that the states must be synchronized!

Usage for Pseudorandom Bit Generation

G is initialized with a random seed and its outputs are then used coins for any purpose needing randomness, including:

- Keys
- IVs for block-cipher based encryption
- Nonces
- Simulations

Methods

- Linear Congruential Generators (LCGs)
- Linear Feedback Shift Registers (LFSRs)

These have

- Good statistical properties: $\#1$'s $\approx$ $\#0$'s; Chi-square; ...
- But are predictable: Given some outputs can infer future ones

Predicatability can be exploited to break encryption privacy via a chosen-message attack.

Cryptographic constructs

- (Alleged)-RC4
- SEAL (1.0, 2.0)

Theoretical Security Requirement

INDR : Indistinguishablity from random

- Pick a random seed St and let $(X_1[1] \dots X_1[m], \mathrm{St}) \leftarrow G(\mathrm{St}, m)$
- Pick $X_0[1] \dots X_0[m]$ at random
- Pick a challenge bit b at random

$$X_b[1] \dots X_b[m] \longrightarrow \boxed{\quad A \quad} \longrightarrow b'$$

A is trying to compute b.

G is secure if no practical A has high advantage.

Formalization

Let G be a stateful generator with seed length s and output-block length n.

Game INDR_G

procedure Initialize	
$\mathrm{St} \xleftarrow{\$} \{0,1\}^s; b \xleftarrow{\$} \{0,1\}$	
procedure Next(m)	**procedure Finalize**(b')
$(X_1[1]\dots X_1[m], \mathrm{St}) \leftarrow G(\mathrm{St}, m)$	return $(b = b')$
$X_0[1]\dots X_0[m] \xleftarrow{\$} \{0,1\}^{nm}$	
return $X_b[1]\dots X_b[m]$	

The indr advantage of adversary A is

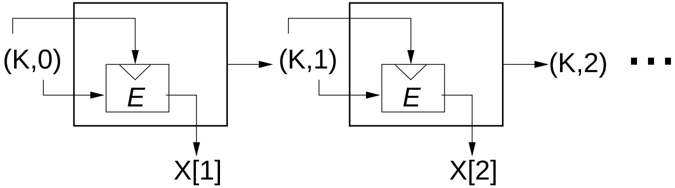
$$\mathbf{Adv}_G^{\mathrm{indr}}(A) = 2\Pr\left[\mathrm{INDR}_G^A \Rightarrow \mathsf{true}\right] - 1$$

Stream Ciphers / PRGs from Block Ciphers

Let $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ be a block cipher, and define:

$\text{algorithm } G(\text{St})$
$(K, i) \leftarrow \text{St}$ // Parse St as (K, i)
$X \leftarrow E_K(i+1)\,;\ \text{return } (X, (K, i+1))$

- State has the form (K, i) where K is a key for E and i is an n-bit integer $(0 \leq i < 2^n)$.
- Initial state is $(K, 0)$ where $K \xleftarrow{\$} \{0,1\}^k$.



Security of the CTR mode PRG

Fact: If E is a secure PRF, then G is an INDR secure PRG.

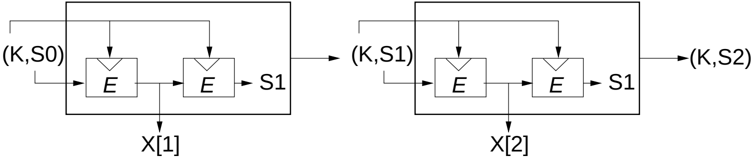
Similarly, other modes of operation of block ciphers also give rise to PRGs.

ANSI X9.17 PRG

Let $E : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$ be a block cipher and define

$\text{algorithm } G(\mathrm{St})$
$(K,S) \leftarrow \mathrm{St}$ // Parse St as (K,S)
$X \leftarrow E_K(S); S \leftarrow E_K(X); \mathrm{return}(X,(K,S))$

- state has the form (K,S) where K is a key for E and $S \in \{0,1\}^n$
- Initial state has both K and S chosen at random.

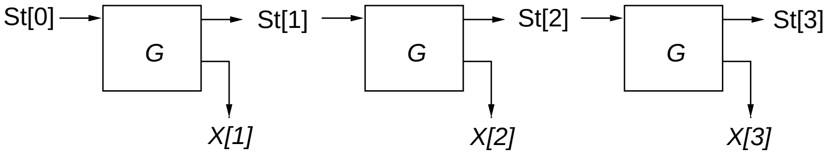


The standard uses $E =$ DES in 2-key EDE mode.
Analysis: [DHL02]

Other Standards, Implementations

- Two PRGs are specified in FIPS-186, based on DES or SHA-1
- NIST SP 800-90 specifies hash-based, HMAC-based, CTR-based, and ECC based generators.
- ANSI X9.31 and ANSI X9.62
- OpenSSL specifies a SHA-1 based PRG.

Forward Security



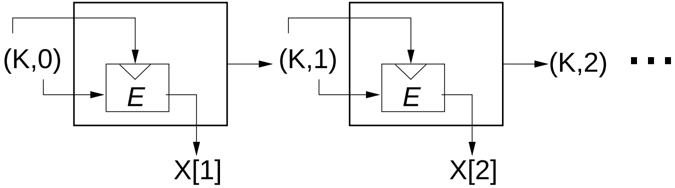
Suppose adversary obtains St_2. Then

- It can compute $X[3]X[4]\ldots$
- But can it compute $X[1]X[2]$?

Forward security requires that the answer to the latter question be "NO".

Important in the face of exposure due to malware and system compromise.

Forward Security Failures



If adversary gets $(K, 2)$ then it can compute

$$X[1] = E_K(0)\,;\ X[2] = E_K(1).$$

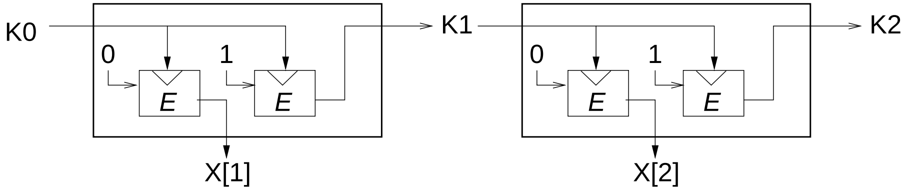
Similar failures for ANSI X9.17.

A Forward Secure PRG [BY]

Let $E : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ be a block cipher, and define:

algorithm $G(K)$
$X \leftarrow E_K(0)\,;\ K \leftarrow E_K(1)\,;$ return (X, K)

- State is a key K for E
- Initial state is a random $K \stackrel{\$}{\leftarrow} \{0,1\}^n$.



The Full Process

In practice, random number generation (RNG) involves

- Seeding
- PRG

Failures in RNG are common: Netscape, Debian Linux, ...
They arise from failures in either component.
In principle, we know how to design good PRGs.
But seeding remains a problem.

Typical methods:

- Maintain an "entropy pool" based on system events, user keystrokes, ...
- Mix in more entropy as needed.

Example: OpenSSL
Alternative: Hardware RNGs