Combining sources of randomness

- Suppose r1, r2, …, rk are random numbers from different sources. E.g.,

  r1 = electrical noise from a resistor or semiconductor

  r2 = sample of hip-hop music on radio

  r3 = clock on computer

  b = r1⊕r2⊕…⊕rk truely number?

  If any one of r1, r2, …, rk is truly random, then so is b

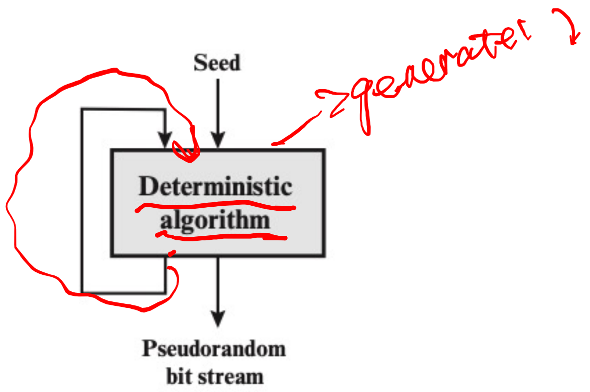
  Many poor sources + 1 good source = good entropy

Pseudorandom Number Generators (PRNGs)

- True randomness is expensive
- **Pseudorandom number generator** (**PRNGs**): An algorithm that uses a little bit of true randomness to generate a lot of random-looking output
  - Also called **deterministic random bit generators** (**DRBGs**)
- PRNGs are deterministic: Output is generated according to a set algorithm
  - However, for an attacker who can’t see the internal state, the output is *computationally indistinguishable* from true randomness

PRNG: Definition

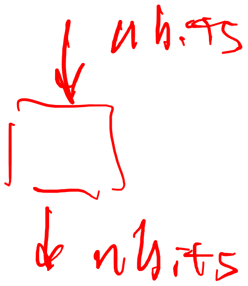
- A PRNG has two functions:
  - PRNG.Seed(randomness): Initializes the internal state using the entropy
    - Input: Some truly random bits
  - PRNG.Generate(*n*): Generate *n* pseudorandom bits
    - Input: A number *s* → algorithm
    - Output: *n* pseudorandom bits $s << n$
    - Updates the internal state as needed



- Properties
  - **Correctness**: Deterministic
  - **Efficiency**: Efficient to generate pseudorandom bits
  - **Security**: Indistinguishability from random
  - **Rollback resistance**: cannot deduce anything about any previously-generated bit

↓ to key

Example construction of PRNG

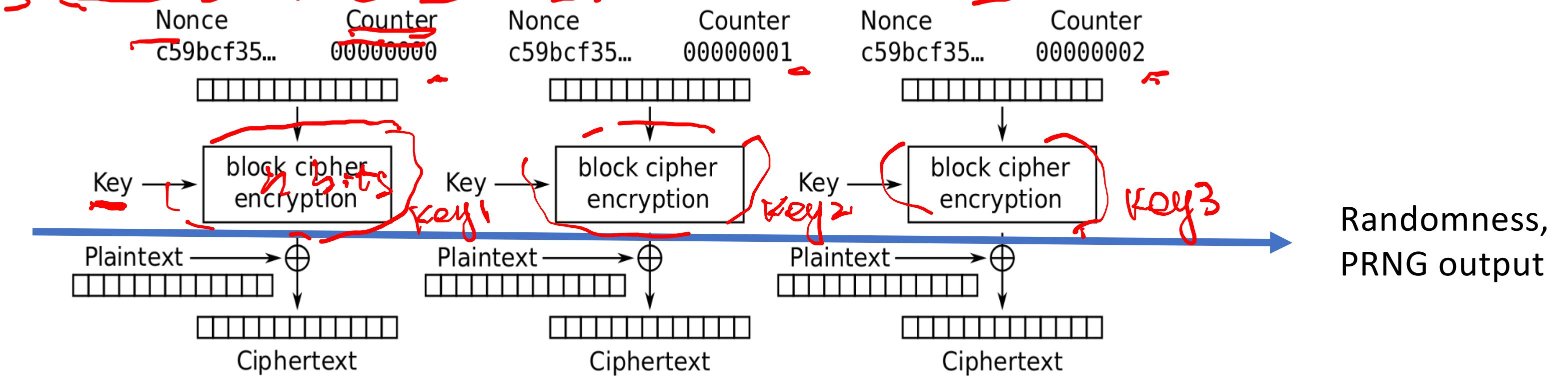


m > n.

$\lceil m/n. \rceil$

- Using block cipher in Counter (CTR) mode:
- If you want m random bits, and a block cipher with E_k has n bits, apply the block cipher m/n times and concatenate the result:
- PRNG.Seed(K | IV) = E_k(IV, 1) | E_k(IV, 2) | E_k(IV, 3) … E_k(IV, ceil(m/n)),
  - | is concatenation
  - Initialization vector (IV) / Nonce – typically is random or pseudorandom

m bits



Counter (CTR) mode encryption

PRNG: Security

- Can we design a PRNG that is truly random?
- A PRNG cannot be truly random
  - The output is deterministic given the initial seed
- A secure PRNG is computationally indistinguishable from random to an attacker
  - Game: Present an attacker with a truly random sequence and a sequence outputted from a secure PRNG
  - An attacker should be able to determine which is which with probability ≈ 0
- Equivalence: An attacker cannot predict future output of the PRNG

Create pseudorandom numbers

- Truly random numbers are impossible with any program!
- However, we can generate seemingly random numbers, called pseudorandom numbers
- The function rand() returns a non-negative number between 0 and RAND_MAX
- For C, it is defined in stdlib.h
- arc4random() is a function available in some operating systems (primarily BSD-based systems like macOS and FreeBSD) that generates random numbers. It is part of the C standard library and provides a more secure and higher-quality source of random numbers compared to rand()

RC4

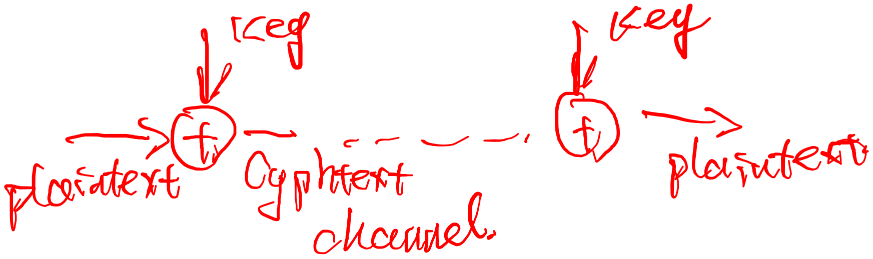
WEP WAP

PRNGs: Summary

- True randomness requires sampling a physical process
- PRNG: An algorithm that uses a little bit of true randomness to generate a lot of random-looking output
  - Seed(entropy): Initialize internal state
  - Generate(n): Generate n bits of pseudorandom output
- Security: computationally indistinguishable from truly random bits

Stream Ciphers

Stream Ciphers

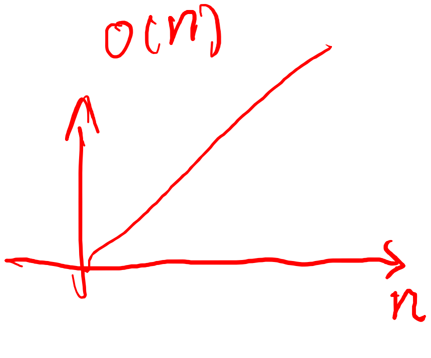


- process the message bit by bit (as a stream)
- typically have a (pseudo) random **stream key**
- combined (XOR) with plaintext bit by bit
- randomness of **stream key** completely destroys any statistically properties in the message ✓
  - $C_i = M_i \text{ XOR } StreamKey_i$
- what could be simpler!!!!
- but must never reuse stream key
  - otherwise, can remove effect and recover messages, $M \oplus K \oplus K = M$

{ large size
fast,

Stream Cipher Properties

- some design considerations are:
  - statistically random
  - depends on large enough key
  - large linear complexity → fast
  - correlation immunity
  - confusion
  - diffusion



How to generate Stream Key?

- How to generate Stream Key?

Stream Ciphers

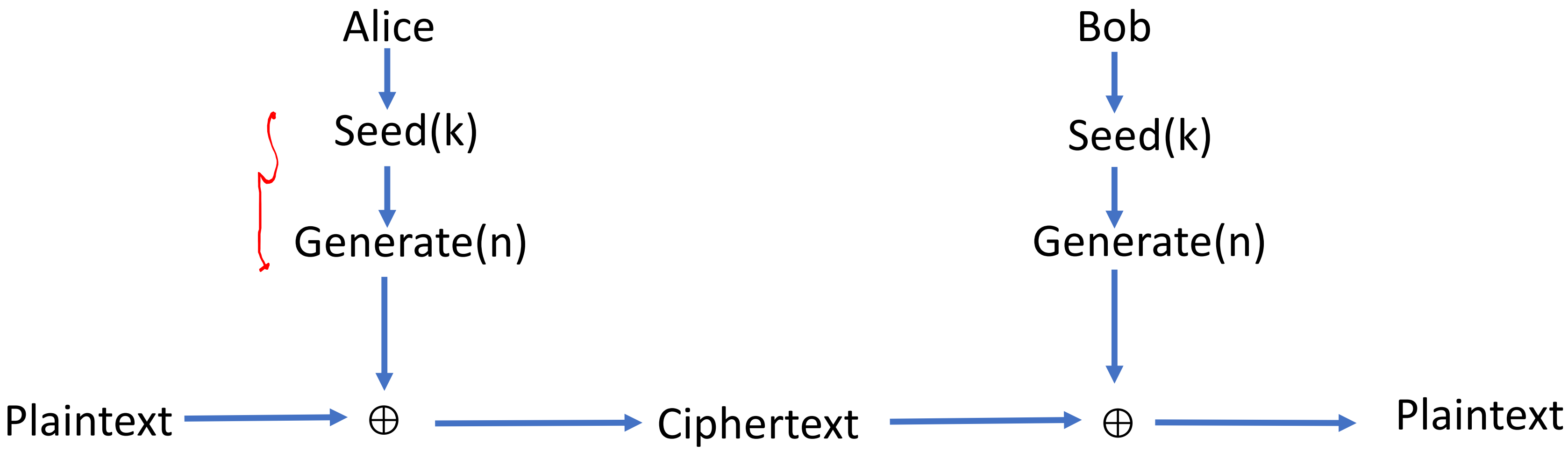
- Idea: replace “rand” by “pseudo rand”
- Use Pseudo Random Number Generator
  - A secure PRNG produces output that looks indistinguishable from random
  - An attacker who can’t see the internal PRNG state can’t learn any output
- PRNG: $\{0,1\}^s \rightarrow \{0,1\}^n$ $s << n$
  - expand a short (e.g., 128-bit) random seed into a long (typically unbounded) string that “looks random”
- Secret key is the seed
  - Basic encryption method: $E_{key}[M] = M \oplus PRNG(key)$

Key

Stream Ciphers

- Protocol: Alice and Bob both seed a secure PRNG with their symmetric secret key, and then use the output as the key for stream key

k → pre-share



Alice

Seed(k)

Generate(n)

Bob

Seed(k)

Generate(n)

Plaintext → ⊕ → Ciphertext → ⊕ → Plaintext