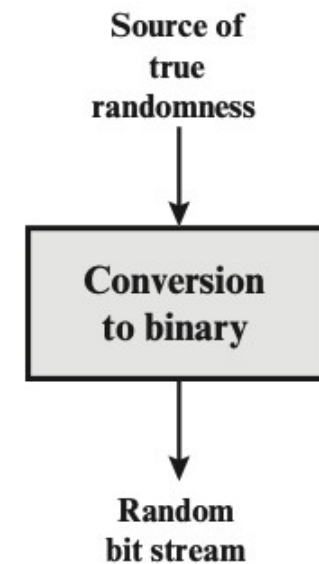


Lecture 9

True random numbers generators

- Several sources of randomness – natural sources of randomness
 - decay times of radioactive materials
 - electrical noise from a resistor or semiconductor
 - radio channel or audible noise
 - keyboard timings
 - disk electrical activity
 - mouse movements
 - Physical unclonable function (PUF)
- Some are better than others



(a) TRNG

Combining sources of randomness

- Suppose r_1, r_2, \dots, r_k are random numbers from different sources.
E.g.,

r_1 = electrical noise from a resistor or semiconductor

r_2 = sample of hip-hop music on radio

r_3 = clock on computer

$b = r_1 r_2 \dots r_k$

If any one of r_1, r_2, \dots, r_k is truly random, then so is b

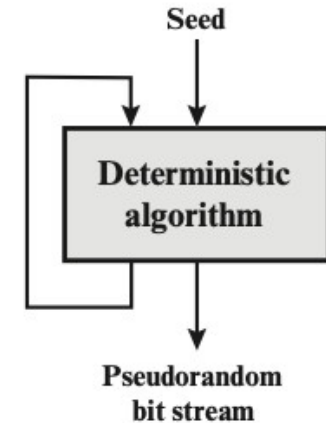
Many poor sources + 1 good source = good entropy

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

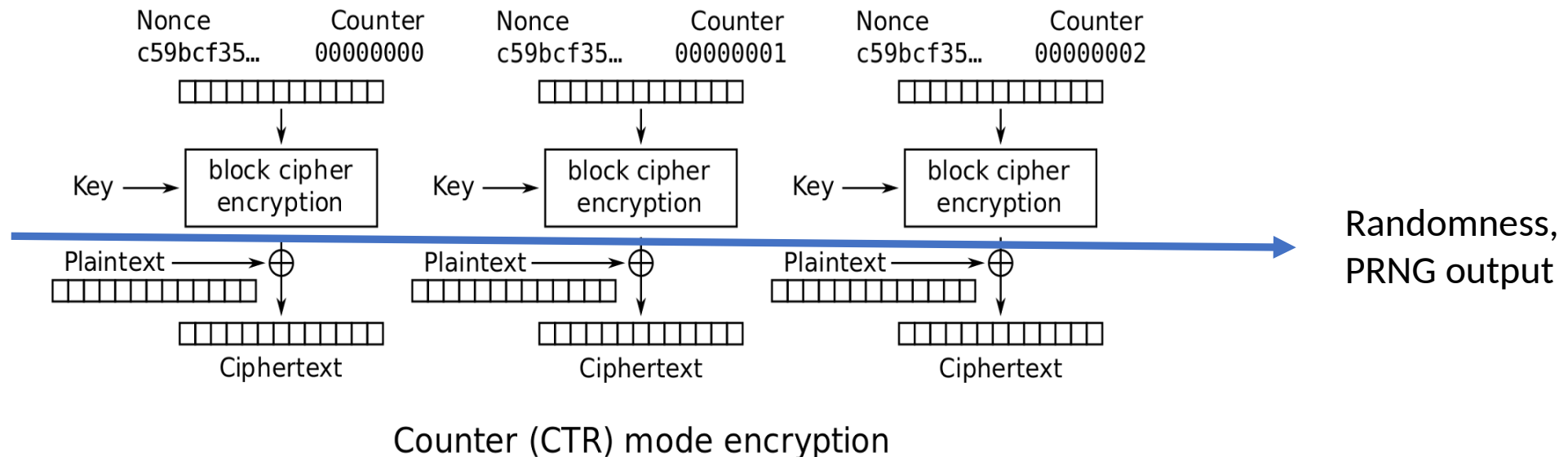
PRNG: Definition

- 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 n
 - Output: n pseudorandom bits
 - 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



Example construction of PRNG

- 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:
- $\text{PRNG.Seed}(K \mid \text{IV}) = E_k(\text{IV}, 1) \mid E_k(\text{IV}, 2) \mid E_k(\text{IV}, 3) \dots E_k(\text{IV}, \text{ceil}(m/n))$,
 - \mid is concatenation
 - Initialization vector (IV) / Nonce – typically is random or pseudorandom



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`

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