Introduction

- Random bit streams used in key generation and encryption…
- Two strategies:

1. Compute bits deterministically using an algorithm- Pseudo-Random Number Generators (PRNGs) or Deterministic Random Bit Generators (DRBGs).
2. Produce bits non-deterministically using some physical source that produces some sort of random output- True Random Number Generators (TRNGs) or Non-deterministic Random Bit Generators (NRBGs).

The use of Random Numbers

- A number of network security algorithms and protocols based on cryptography make use of random binary numbers:

1. Key distribution and reciprocal (mutual) authentication schemes
2. Session key generation
3. Generation of keys for the RSA public-key encryption algorithm
4. Generation of a bit stream for symmetric stream encryption

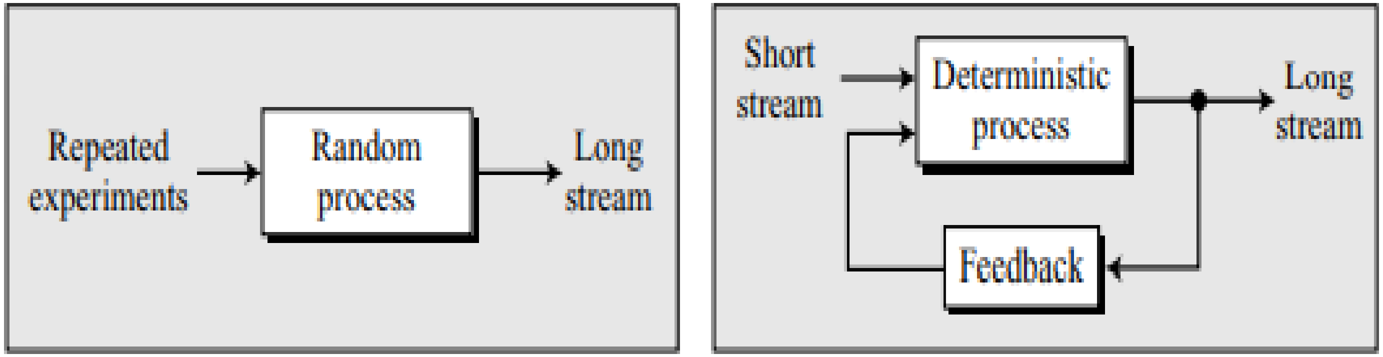
- Two requirements for a sequence of random numbers: randomness and unpredictability

The Two Approaches

- There are two approaches to generating a long stream of random bits:
  - Using a natural random process, such as flipping a coin many times and interpreting heads and tails as 0-bits and 1-bits,
  - Or using a deterministic process with feedback.

- The first approach is called a **True Random Number Generator (TRNG);**
- The second is called a **Pseudo-Random Number Generator (PRNG).**

The Two Approaches



a. TRGN

b. PRNG

PRNG

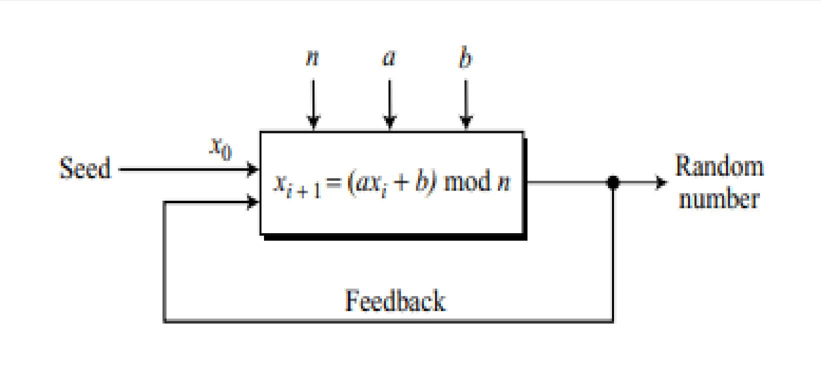
- A reasonably random stream of bits can be achieved using a deterministic process with a short random stream as the input (seed).
- The generated number is not truly random because the process that creates it is deterministic.
- PRNGs can be divided into **two broad categories**:
  - **Congruential generators** and
  - **Generators using cryptographic ciphers**

Congruential Generators: Linear Congruential Generator

- This method recursively creates a sequence of pseudorandom numbers using a linear congruence equation of the form:

  $\mathbf{x_{i+1} = (ax_i + b) \bmod n}$,

  where x_0, called the seed, is a number between 0 and n − 1.



Congruential Generators: Linear Congruential Generator

- The sequence is periodic, where the period depends one how carefully the coefficients, a and b, are selected. The ideal is to make the period as large as the modulus n.
- **Example**
  - Assume that a = 4, b = 5, n = 17, and x0 = 7.
  - Find out the sequence and the period

Congruential Generators: Linear Congruential Generator

- The sequence is periodic, where the period depends one how carefully the coefficients, a and b, are selected. The ideal is to make the period as large as the modulus n.
- **Example**
  - Assume that a = 4, b = 5, n = 17, and x0 = 7.
  - The sequence is 16, 1, 9, 7, 16, 1, 9, 7, …, which is definitely a poor pseudorandom sequence; the period is only 4.

Congruential Generators: Linear Congruential Generator

- **Criteria for an acceptable PRNG:**

1. The period must be equal to n (the modulus). This means that, before the integers in the sequence are repeated, all integers between 0 and $n - 1$ must be generated.
2. The sequence in each period must be random.
3. The generating process must be efficient. Most computers today are efficient when arithmetic is done using 32-bit words.

Congruential Generators: Linear Congruential Generator

- **Recommendation:**

For selecting the coefficients of the congruence equation and the value of the modulus:

1. For the modulus, n, choose the largest prime number close to the size of a word in the computer being used. The recommendation is to use the thirty-first Mersenne prime as the modulus: $n = M_{31} = 2^{31} - 1$.

2. To create a period as long as the modulus, the value of the first coefficient, a, should be a primitive root of the prime modulus. Although the integer 7 is a primitive root of M_{31}, it is recommended to use 7^k , where k is an integer coprime with $(M_{31} - 1)$. Some recommended values for k are 5 and 13. This means that $(a = 7^5)$ or $(a = 7^{13})$.

3. For the second recommendation to be effective, the value of the second coefficient, b, should be zero.

Congruential Generators: Linear Congruential Generator

- Linear Congruential Generator:
  $\mathbf{x_{i+1} = (ax_i + b) \bmod n}$,
  where $n = 2^{31} - 1$ and ($a = 7^5$) or ($a = 7^{13}$).
- **Security**
  - Shows reasonable randomness if the previous recommendations are followed.
  - The sequence is useful in some applications where only randomness is required (such as simulation);
  - It is useless in cryptography where both randomness and secrecy are desired.

Congruential Generators: Linear Congruential Generator

- Because n is public, the sequence can be attacked by Eve using one of the two strategies:
  - If Eve knows the value of the seed (x_0) and the coefficient a, she can easily regenerate the whole sequence.
  - If Eve does not know the value of x_0 and a, she can intercept the first two integers and use the following two equations to find x_0 and a:

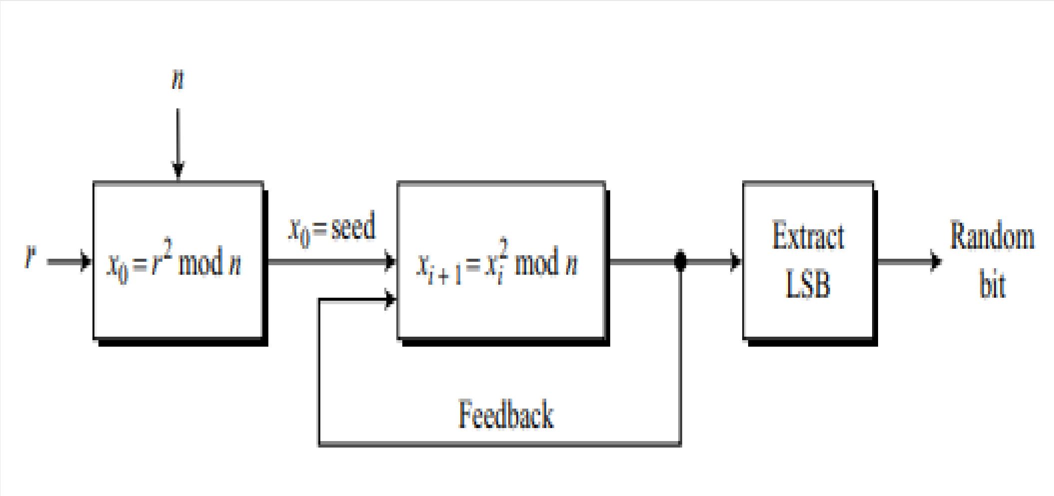
    $$\mathbf{x_1 = ax_0 \bmod n}$$

    $$\mathbf{x_2 = ax_1 \bmod n}$$

Other Congruential Generators

- **Quadratic Residue Generator**
  - To make the pseudorandom sequence less predictable.
  - $\mathbf{x_{i+1} = x_i^2 \bmod n}$, where x_0, called the seed, is a number between 0 and n − 1.
- **Blum Blum Shub(BBS) Generator**
  - After the names of its three inventors.
  - BBS uses quadratic residue congruence, but it is a pseudorandom bit generator instead of a pseudorandom number generator;
  - It generates a sequence of bits (0 or 1).

Congruential Generators: Blum Blum Shub(BBS) Generator



Congruential Generators: Blum Blum Shub(BBS) Generator

- **Steps:**
  1. Find two large primes numbers p and q in the form 4k + 3, where k is an integer (both p and q are congruent to 3 modulo 4).
  2. Select the modulus $n = p \times q$.
  3. Choose a random integer r which is coprime to n.
  4. Calculate the seed as $x_0 = r^2 \bmod n$.
  5. Generate the sequence as $x_{i+1} = x_i^2 \bmod n$.
  6. Extract the least significant bit of the generated random integer as the random bit

Congruential Generators: Blum Blum Shub(BBS) Generator

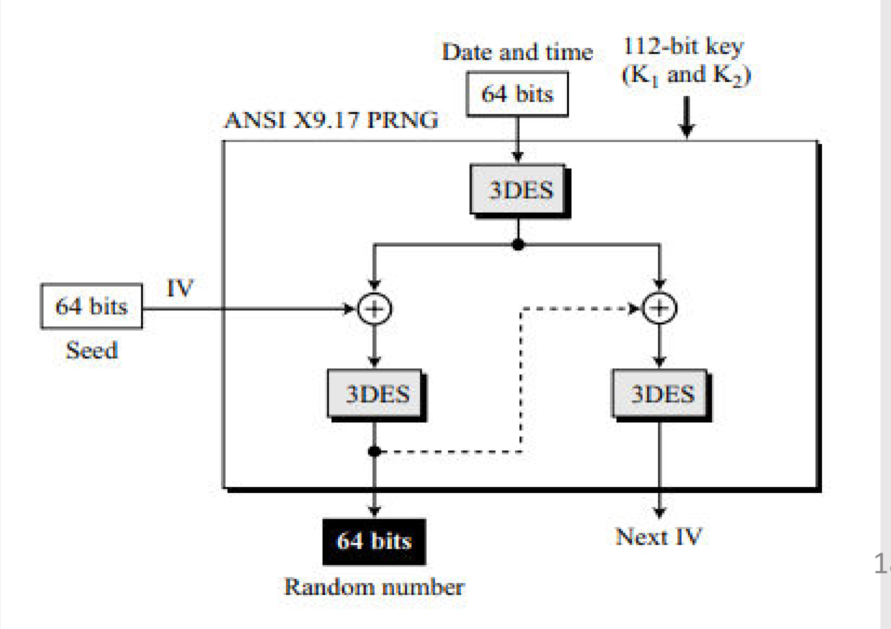
- **Security**
  - If p and q are known, the i^{th} bit in the sequence can be found as the least significant bit of
  - $\mathbf{x_i = x_0^{\,2^i \bmod [(p-1)(q-1)]} \bmod n}$
  - If Eve knows the value of p and q, she can find the value of the i^{th} bit by trying possible values of x_0 (the value of n is usually public).
  - This means that the complexity of this generator is the same as the factorization of n.
  - If n is large enough, the sequence is secure (unpredictable).
  - It has been proved that with a very large n, Eve cannot guess the value of the next bit in the sequence even if she knows the values of all previous bits.
  - The probability of each bit being 0 or 1 is very close to 50 %.

Cryptosystem-Based Generators

- A cryptosystem such as an encryption cipher or a hash function can also be use to generate a random stream of bits.
- **ANSI X9.17 PRNG**
- A cryptographically strong pseudorandom number generator.
- The generator uses three 3DES with two keys (encryption-decryption-encryption.

Cryptosystem-Based Generators: ANSI X9.17 PRNG

- **Fig: The Cipher-Block Chaining (CBC) mode**
  - Note that the first pseudorandom number uses a 64-bit seed as the initial vector (IV);
  - the rest of the pseudorandom numbers use the seed shown as the next IV.
  - The same 112-bit secret key (K_1 and K_2 in 3DES), are used for all three 3DES ciphers.



Cryptosystem-Based Generators: ANSI X9.17 PRNG

- Uses two stages of the block chaining.
- The plaintext for each stage comes from the output of the first 3DES, which uses the 64-bit date and time as the plaintext.
- The ciphertext created from the second 3DES is the random number.
- The ciphertext created from the third 3DES is the next IV for the next random number.

Cryptosystem-Based Generators: ANSI X9.17 PRNG

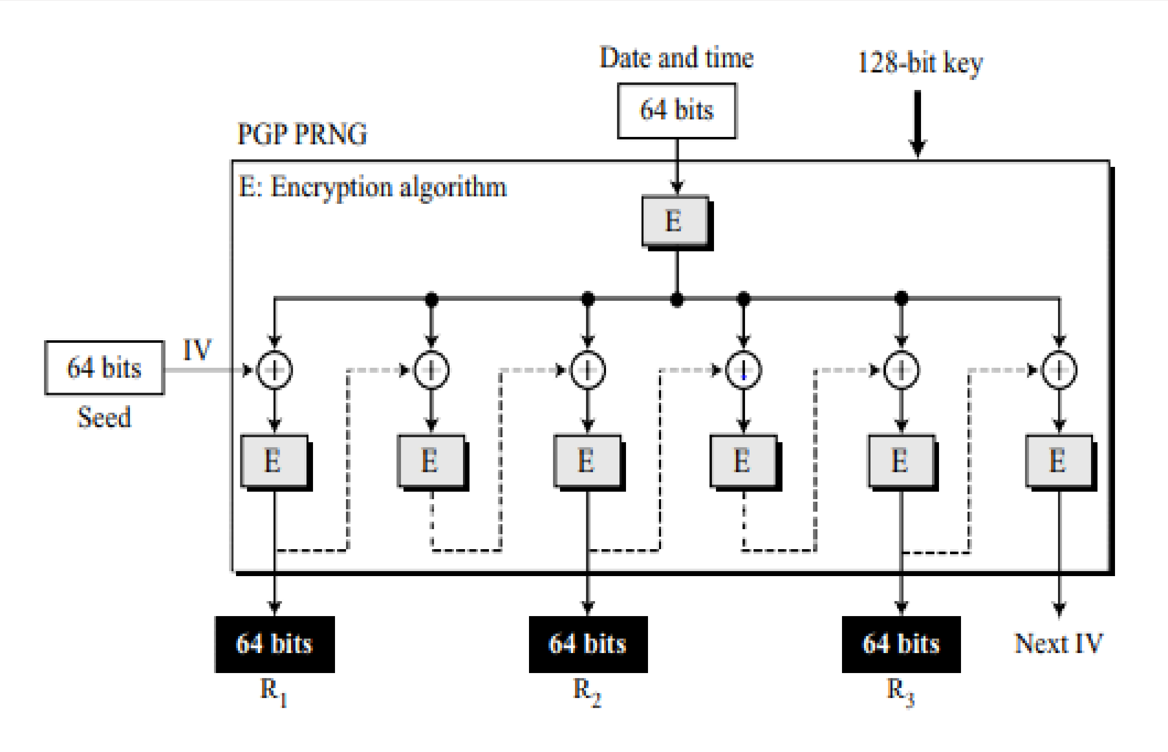
- The strength of X9.17 can be due to the following facts:
  1. The key is 112 (2 × 56) bits.
  2. The date-and-time input of 64 provides a good timestamp preventing replay attack.
  3. The system provides an excellent confusion-diffusion effect with six encryptions and three decryptions.

Cryptosystem-Based Generators: PGP PRNG

- **PGP PRNG**
  - Uses the same idea as X9.17 with several changes:

1. PGP PRNG uses seven stages instead of two.
2. The cipher is either IDEA or CAST-128.
3. The key is normally 128 bits. PGP PRNG creates three 64-bit random numbers: the first is used as the IV secret (for communication using PGP, not for PRNG), the second and the third are concatenated to create a 128-bit secret key (for communication using PGP).

Cryptosystem-Based Generators: PGP PRNG



- The strength of PGP PRNG is in its key size and in the fact that the original IV (seed) and the 128-bit secret key can be generated from a 24-byte true random variable.