COMP412 Computer Security

Lec 05 Pseudo-Random Numbers and Stream Ciphers

Dr. Xiaochen Yuan
2021/2022

Contents

Random Numbers

Use of Random Numbers

- Key distribution and authentication schemes
- Generation of session keys or keys for RSA
- Generation of bit stream for stream ciphers

Randomness

- Uniform distribution: frequency of occurrence of 1's and 0's approximately equal
- Independence: no sub-sequence can be inferred from others

Example of check for randomness.

Unpredictability

- Hard to predict next value in sequence

TRNG, PRNG and PRF

True Random Number Generator

- Non-deterministic source, physical environment
- Detect ionizing radiation events, leaky capacitors, thermal noise from resistors or audio inputs
- Mouse/keyboard activity, I/O operations, interrupts
- Inconvenient, small number of values

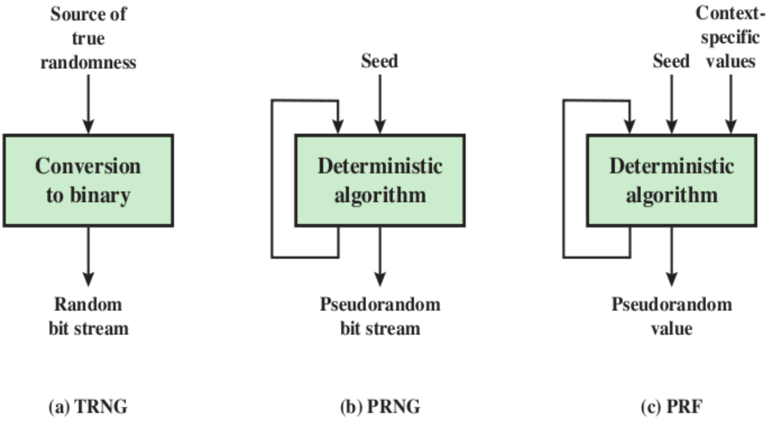
Pseudo Random Number Generator

- Deterministic algorithms to calculate numbers in "relatively random" sequence
- Seed is algorithm input
- Produces continuous **stream of random bits**

Pseudo Random Function

- Same as PRNG but produces **string of bits** of some fixed length

Random and Pseudo-Random Number Generators

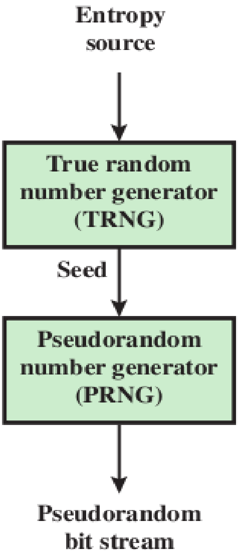


Requirements of PRNG

Hard to determine pseudo-random stream if don't know seed (but know algorithm)

- **Randomness**
  - Test for uniformity, scalability, consistency
  - Examples: Frequency, runs, compressability
- **Unpredictability**
  - Forward and backward unpredictability
- **Seed must be secure**
  - Use TRNG to generate seed

Generation of Seed Input to PRNG



Contents

Linear Congruential Generator

Parameters:

- m, the modulus, $m > 0$
- a, the multiplier, $0 < a < m$
- c, the increment, $0 \leq c < m$
- X_0, the seed, $0 \leq X_0 < m$

Generate sequence of pseudo-random numbers, $\{X_n\}$:

$$\boxed{X_{n+1} = (aX_n + c) \bmod m}$$

Choice of a, c and m is important:

- m should be large, prime, e.g. $2^{31} - 1$
- If $c = 0$, few good values of a, e.g. $7^5 = 16807$

If attacker knows parameters and one number, can easily determine subsequent numbers

Example of different parameter settings.

Example Operation of LC Generator

Case 1:

$a = 1$, $c = 1$, $m = 100$

Seed: $X_0 = 23$

- Generate the pseudo-random number sequence.
- Find the sequence period (how many different numbers in the generated stream).

Case 2:

$a = 7$, $c = 0$, $m = 32$

Seed: $X_0 = 1$

- Generate the pseudo-random number sequence.
- Find the sequence period (how many different numbers in the generated stream).

Example Operation of LC Generator

Case 3:

a = 5, c = 0, m = 32

Seed: $X_0 = 1$

- Generate the pseudo-random number sequence.
- Find the sequence period (how many different numbers in the generated stream).

Case 4:

a = 5, c = 0, m = 32

Seed: ***$X_0 = 3$***

- Generate the pseudo-random number sequence.
- Find the sequence period (how many different numbers in the generated stream).

Blum Blum Shub (BBS) Generator

Parameters:

- p, q: large prime numbers such that $p \equiv q \equiv 3 \pmod 4$
- $n = p \times q$
- s, random number relatively prime to n → *Neither p nor q is a factor of s.*

Generate sequence of bits, B_i:

$$
\begin{aligned}
X_0 &= s^2 \bmod n \\
\text{for } i &= 1 \rightarrow \infty \\
X_i &= (X_{i-1})^2 \bmod n \\
B_i &= X_i \bmod 2
\end{aligned}
$$

→ *Select the least significant bit*

Cryptographically secure pseudo-random bit generator

Example Operation of BBS Generator

$n = 192649 = 383 \times 503, s = 101355$

i	X_i	B_i
0	20749	
1	143135	1
2	177671	1
3	97048	0
4	89992	0
5	174051	1
6	80649	1
7	45663	1
8	69442	0
9	186894	0
10	177046	0

i	X_i	B_i
11	137922	0
12	123175	1
13	8630	0
14	114386	0
15	14863	1
16	133015	1
17	106065	1
18	45870	0
19	137171	1
20	48060	0

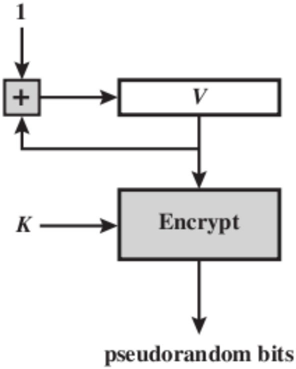
383 mod 4 = ?
503 mod 4 = ?

Contents

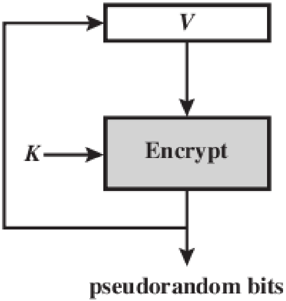
PRNG Mechanisms Based on Block Ciphers

Use symmetric block ciphers (e.g. AES, DES) to produce pseudo-random bits

- Seed is encryption key *K*, and value *V* (which is updated each block of pseudorandom numbers is generated.)



Counter Mode



OFB Mode

ANSI X9.17 PRNG

Cryptographically secure (one of the strongest) PRNG using Triple DES Parameters:

- ***Input 1:*** 64-bit representation of the date & time, Dt_i
  - Updated on each number generation.
- ***Input 2:*** 64-bit seed value, V_i
  - Initialized to arbitrary value, being updated.
- ***Keys:*** Pair of 56-bit DES keys, K_1 and K_2

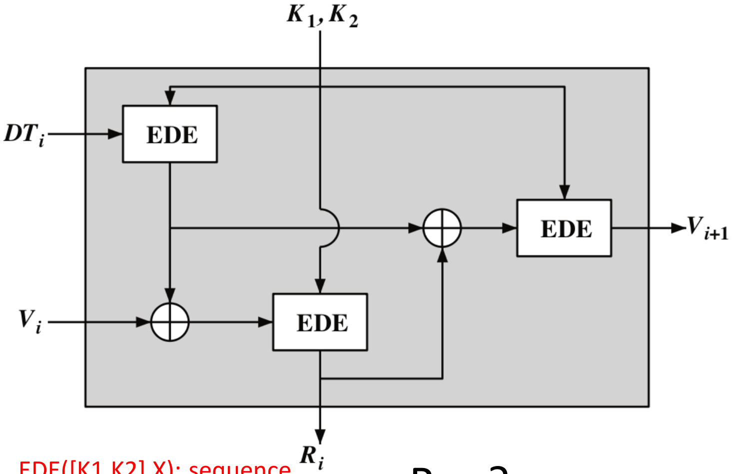
Operation:

- Uses Triple DES three times
- (see next slide)

Output:

- 64-bit pseudo-random number, R_i
- 64-bit seed value, V_{i+1}

ANSI X9.17 PRNG



EDE([K1,K2],X): sequence encrypt-decrypt-encrypt using two-key triple DES for encryption.

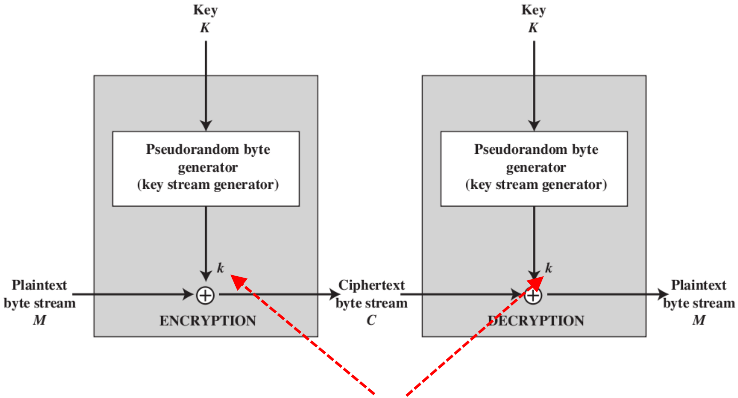
R_i = ?

V_{i+1} = ?

Contents

Stream Ciphers

Encrypt one byte at a time by **XOR** with pseudo-random byte



Output of generator is called keystream

Design Criteria for Stream Ciphers

Important Considerations

- Encryption **sequence** should have large period
- **Keystream** should approximate true random number stream
- **Key** must withstand brute force attacks

Comparison to Block Ciphers

- Stream ciphers often simpler to implement, faster
- Block ciphers can re-use keys

Contents

RC4

- Designed by Ron Rivest in 1987
- Used in secure web browsing and wireless LANs
- Very simple and efficient implementation
- Can use **variable size key**: 8 to 2048 bits

- Several theoretical limitations of RC4
  - No known attacks if use 128-bit key and discard initial values of stream
  - RC4 is used in Wireless Encryption Protocol, WEP (shown to be weak security for wireless LANs)—problem with how keys are used, not RC4 algorithm

RC4 Algorithm

Parameters and Variables

- Variable length key, ***K***, from 1 to 256 Bytes
- State vector, ***S***, 256 Bytes
- Temporary vector, ***T***, 256 Bytes
- A byte from keystream, ***k***, generated from *S*

Steps

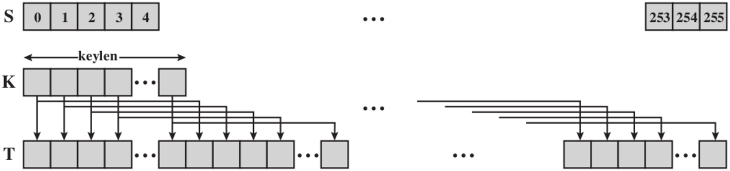
1. Initialise ***S*** to values 0 to 255; initialise ***T*** with repeating values of key, ***K***
2. Use ***T*** to create initial permutation of ***S***
3. Permutate ***S*** and generate keystream, ***k*** from *S*
4. Encrypt a byte of plaintext, *p*, by XOR with *k*

1. Initial State of S and T

```
for i =0 to 255 do S[i] =
    i;
    T[i] = K[i mod keylen];
```

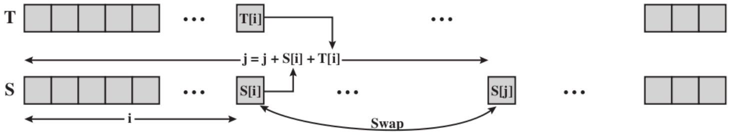
Initialise ***S*** to values 0 to 255

initialise ***T*** with repeating values of key ***K***



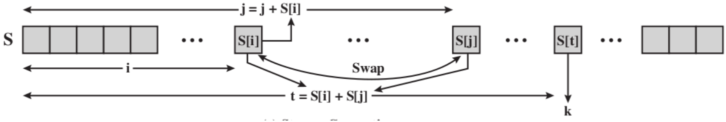
2. Initial Permutation of S

```
j = 0;
for i = 0 to 255 do
    j = (j + S[i] + T[i]) mod 256;
  Swap (S[i], S[j]);
```



3. Stream Generation

```
i, j =0; while
(true)
    i = (i + 1) mod 256;
    j = (j + S[i]) mod 256;
    Swap (S[i], S[j]);
    t =(S[i] +S[j]) mod256; k =
    S[t];
```



4. Encryption and Decryption

To encrypt:

$$C = p \text{ XOR } k$$

To decrypt:

$$p = C \text{ XOR } k$$

RC4 Example – Simplified RC4

- Instead of using the full 256 bytes (256 x 8-bit), we use 8 x 3-bit to demonstrate RC4 algorithm.

Key (4 x 3-bit):	K = [1 2 3 6]
Plaintext (4 x 3-bit):	P = [1 2 2 2]
Ciphertext	C = ?