Topics to Cover

- Principles of Pseudorandom Number generation
- Pseudorandom Number Generators
- Pseudorandom Number Generation using a Block Cipher
- Stream Ciphers
- RC4

PRINCIPLES OF PSEUDORANDOM NUMBER GENERATION

Applications of Random Numbers in Cryptography and N/w security

- Key Distribution and Authentication Schemes.
- Session Key Generation
- Generation of bit stream for symmetric stream encryption
- IVs
- Nonces
- Digital Signatures

Requirements while Generating Random Numbers

- Uniform Distribution
- Independence
- Unpredictability
- Random Number Generator Source
- Length of Random Numbers
- Reproducibility
- Validation

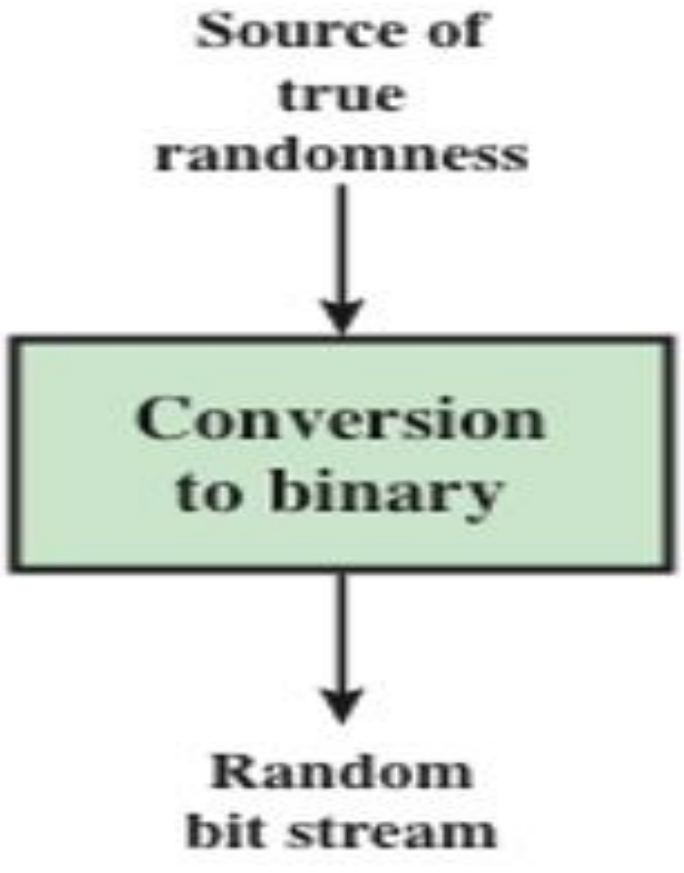
Different Types of Random Number Generators (RNGs)

- Pseudorandom RNGs (PRNGs)
- True RNGs (TRNGs)
- Cryptographically Secure PRNGs (CSPRNGs)
- Hybrid RNGs (HRNGs)

Pseudorandom Numbers

- Cryptographic systems generally utilize algorithms to generate random numbers.
- These algorithms operate in a predictable manner, leading to number sequences that lack true randomness.
- However, a well-designed algorithm can yield sequences that meet various randomness criteria.
- The numbers produced by these algorithms are known as pseudorandom numbers.

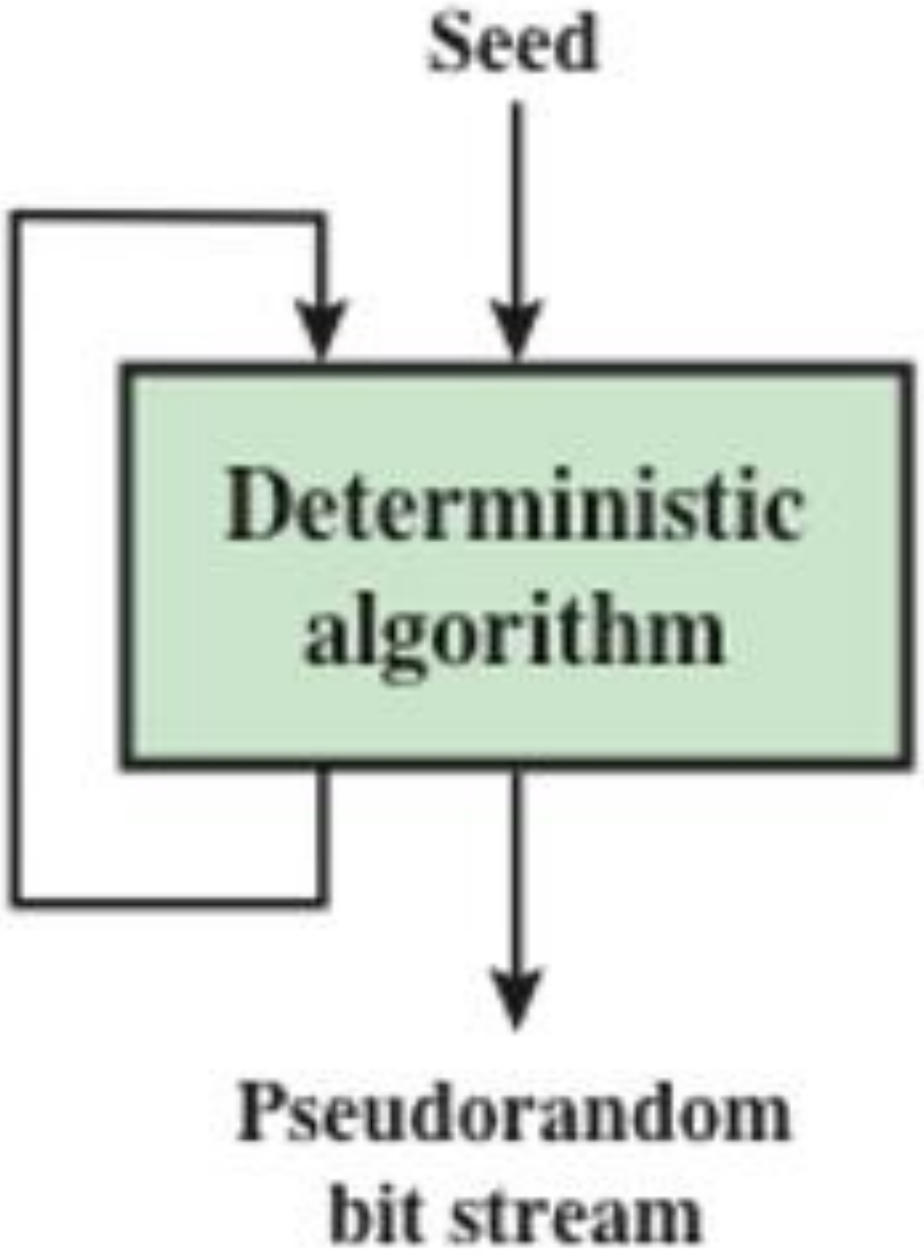
TRNGs


Source of true randomness
Conversion to binary
Random bit stream

TRNGs (Entropy Sources)

- Mouse Movements
- Keystrokes
- Disk Electric Activity
- Instantaneous values of System Clock
- Avalanche Noise
- Magnetic Fluctuations
- Brownian Motion
- Environmental Conditions

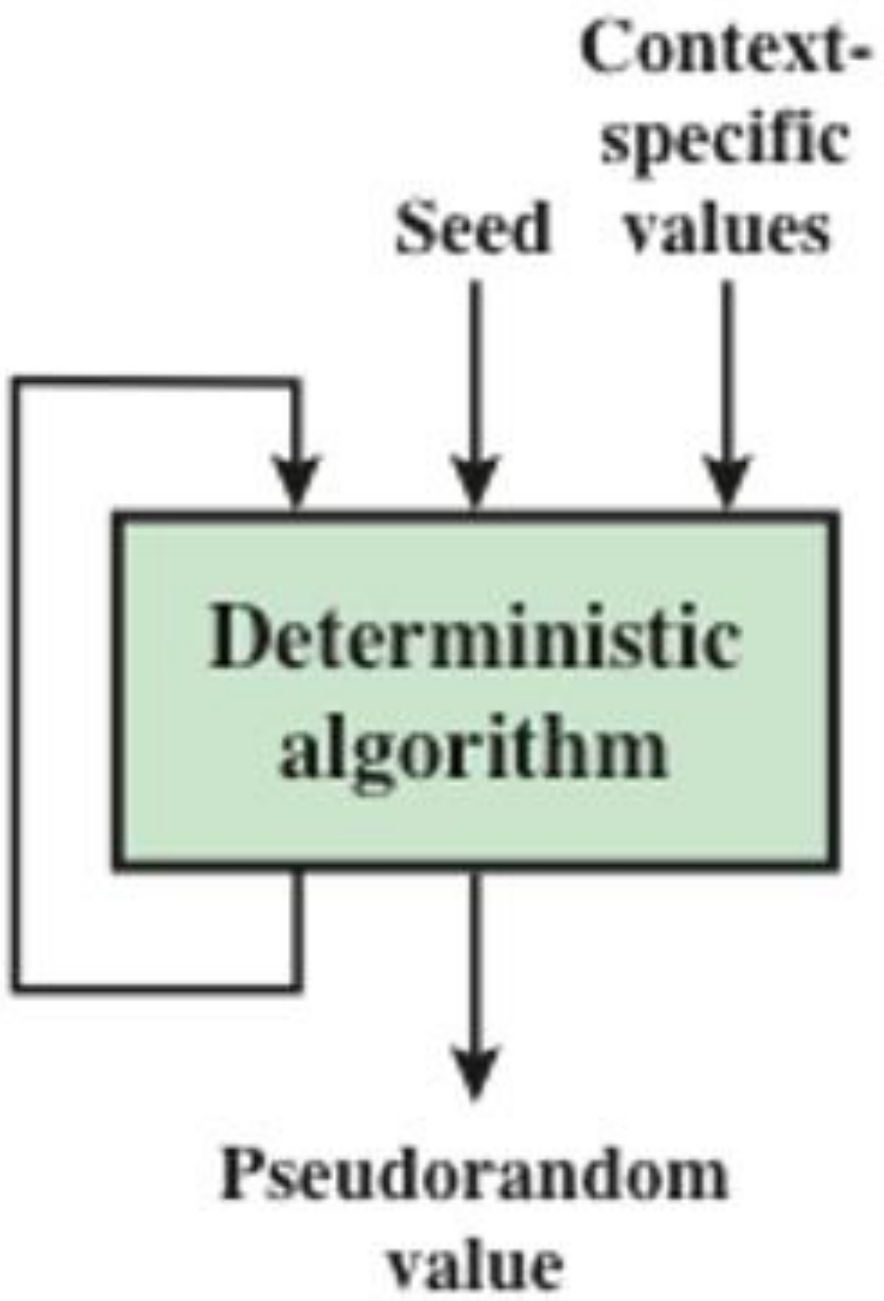
PRNGs



PRNGs

- Produce sequences which approximate the properties of Random Numbers.
- Uses Seed as the initial value which is fed to a deterministic algorithm.
- Period of the sequence is decided by the deterministic algorithm.
- A good PRNG follows a uniform distribution.
- Most common deterministic algorithms are LCG, Mersenne Twister, XorShift, etc.
- Most common application is Symmetric Stream Ciphers.

Pseudorandom Function (PRF)



Differences between PRNGs and PRF

PRNGs	PRF
• Inputs:- Seed	• Inputs:- Seed, Context Specific Value
• Output is predictable if seed is known	• Output is deterministic, but random
• Limited Key Space	• Has more Key Space
• Less Secure	• More Secure

Purpose of PRNG and PRF Requirements

- Output Secrecy
- Risk of Knowledge by an attacker
- Brute Force Attack (BFA)
- Areas of Requirements:- Randomness, Unpredictability, and Seed Requirements

Randomness in PRNGs

- PRNGs produce bit streams which are random, though deterministic.
- Multiple Tests are required to test the Randomness of PRNGs.

- NIST SP 800-22 specified Characteristics for Randomness:-

- Uniformity:- Probability of ‘1’s and ‘0’s is 50%.
- Scalability:- Any random subsequence taken from the main sequence should also appear random.
- Consistency:- The PRNG's output should be reliable and similar regardless of the seed value

15 Tests Defined by SP 800-22 for Randomness in PRNGs

- *Frequency (Monobit)*
- *Runs Test*
- *Maurer Universal Statistical*
- Block Frequency
- Longest Run of Ones
- Binary Matrix Rank Test
- Discrete Fourier (Spectral)
- Non-Overlapping Template

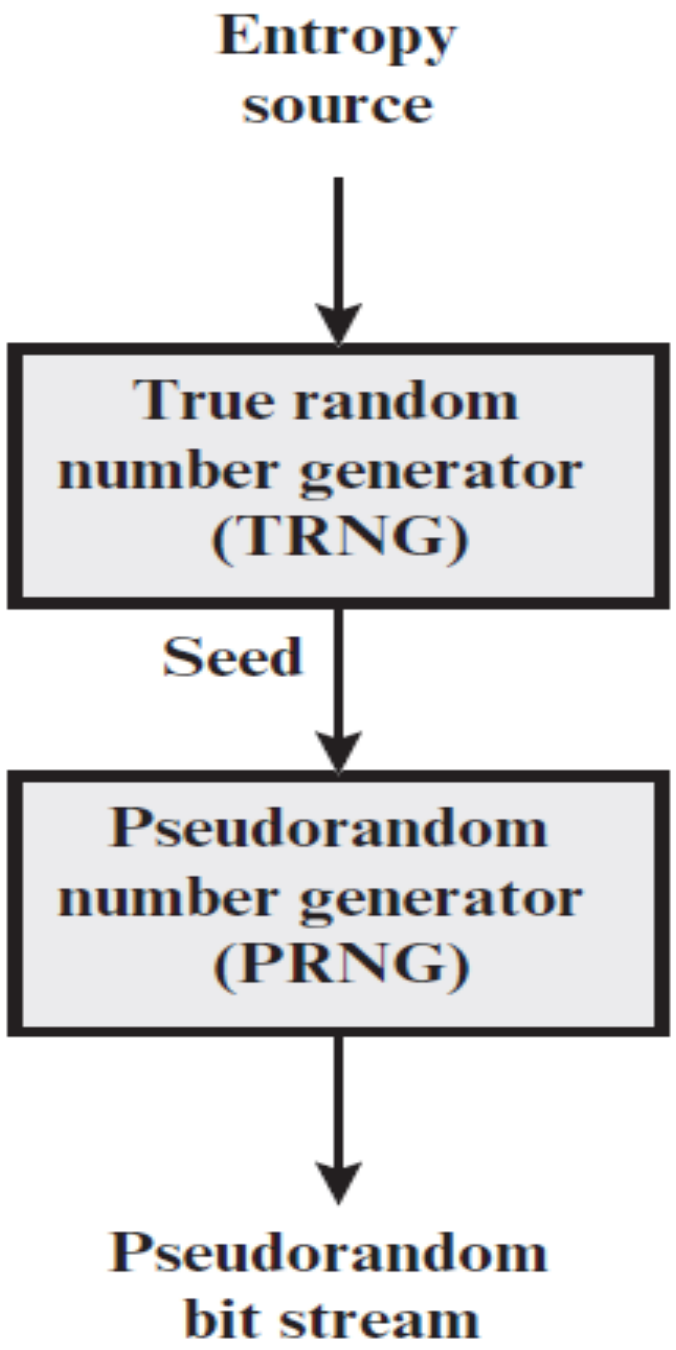
15 Tests Defined by SP 800-22 for Randomness in PRNGs (Contd..)

- Overlapping Template Matching
- The Linear Complexity Test
- The Serial Test
- Approximate Entropy
- Cumulative Sums
- Random Excursions
- Random Excursions Variant

Unpredictability in PRNGs

- Forward Unpredictability
- Backward Unpredictability
- Tests used for Randomness are also used for testing Unpredictability
- Each bit of a sequence is like an independent event.

Seed Requirements for PRNGs



Seed Requirements for PRNGs (Contd..)

- Seed for PRNG must be secure and unpredictable.
- An attacker can predict the output if he/she has knowledge regarding the seed.
- TRNG provides more randomness in the seed generation.
- PRNGs can be directly used for some applications like stream ciphers, where TRNGs are not practical.
- Moreover, TRNG can have biased output, which can be eliminated by using PRF.

PSEUDORANDOM NUMBER GENERATORS

Linear Congruential Generator (LCG)

- Uses a Linear Congruence Formula for generating a sequence of random numbers.

- $X_{n+1} = (a*X_n + c) \bmod m$
- $m, a, c, X_n \in Z$
- $m > 0$
- $0 < a < m$
- $0 \leq c < m$
- $0 \leq X_n < m$

- Selection of a, c, m, play a crucial role in developing a good random number generator.

LCG (Contd..)

- Large ‘m’ would produce a long series of distinct random numbers.
- Length of LCG cycle = Period
- LCG is said to achieve its full cycle if Period = m
- Period of LCG is high for good choices of a, m, c.
- $m \approx 2^{31}$ in programming environments.
- If m is of the form 2^n ($n \in Z$), and right choices for ‘a’ and ‘c’ are made, then the period of LCG can be maximized.

LCG (Contd..)

- 3 Tests proposed by PARK88 in evaluating an RNG:-

- T1: The random number generator should create all numbers from 0 to m−1 before starting over.
- T2: The sequence of numbers produced should look random.
- T3: The generator should work efficiently using 32-bit math.

- Appropriate values of a, c, m, will make T1, T2, and T3 pass.

LCG (Contd..)

- For T1, if m = prime, c = 0, specific values of ‘a’ produce a period = m-1
- For 32-bit Arithmetic, a convenient prime is $2^{31} – 1$.
- For 32-bit Arithmetic, $X_{n+1} = a*X_n \bmod (2^{31} – 1)$

- Methods to increase unpredictability feature in LCG:-

- Using an internal system clock to modify the random number stream.
- Restarting the sequence after every N numbers with the current clock value (mod m) as the new seed.
- Adding the current clock value to each random number (mod m) for further unpredictability.

LCG (Numerical 1)

- Generate a sequence of random numbers (show first 6 numbers) which would be generated using LCG with modulus = 123, seed = 73, multiplier = 5, and increment = 2.

Solution:-

- $m = 123;\ X_0 = 73;\ a = 5;\ c = 2$

- $X_{n+1} = (5*X_n + 2) \bmod 123$

LCG (Numerical 1) (Contd..)

- $X_1 = (5*X_0 + 2) \bmod 123 = 121$
- $X_2 = (5*X_1 + 2) \bmod 123 = 115$
- $X_3 = (5*X_2 + 2) \bmod 123 = 85$
- $X_4 = (5*X_3 + 2) \bmod 123 = 58$
- $X_5 = (5*X_4 + 2) \bmod 123 = 46$

- Sequence = {73, 121, 115, 85, 58, 46, ……..}

LCG (Numerical 2)

- Generate a sequence of random numbers (show first 6 numbers) which would be generated using LCG with modulus = 100, seed = 27, multiplier = 17, and increment = 43.

Solution:-

- $m = 100;\ X_0 = 27;\ a = 17;\ c = 43$

- $X_{n+1} = (17 * X_n + 43) \bmod 100$

LCG (Numerical 2) (Contd..)

- $X_1 = (17*X_0 + 43) \bmod 100 = 2$
- $X_2 = (17*X_1 + 43) \bmod 100 = 77$
- $X_3 = (17*X_2 + 43) \bmod 100 = 52$
- $X_4 = (17*X_3 + 43) \bmod 100 = 27$
- $X_5 = (17*X_4 + 43) \bmod 100 = 2$

- Sequence = {27, 2, 77, 52, 27, 2, …..}
- Period = 4

LCG (Numerical 3)

- Calculate the period for LCG with modulus = 8, seed = 4, multiplier = 5, and increment = 3.

Solution:-

- $m = 8;\ X_0 = 4;\ a = 5;\ c = 3$

- $X_{n+1} = (5*X_n + 3) \bmod 8$

LCG (Numerical 3) (Contd..)

- $X_1 = (5*X_0 + 3) \bmod 8 = 7$
- $X_2 = (5*X_1 + 3) \bmod 8 = 6$
- $X_3 = (5*X_2 + 3) \bmod 8 = 1$
- $X_4 = (5*X_3 + 3) \bmod 8 = 0$
- $X_5 = (5*X_4 + 3) \bmod 8 = 3$
- $X_6 = (5*X_5 + 3) \bmod 8 = 2$
- $X_7 = (5*X_6 + 3) \bmod 8 = 5$
- $X_8 = (5*X_7 + 3) \bmod 8 = 4$

- Period = 8

Pros and Cons of LCG

- Simple to understand and easy to implement
- Efficient
- Limited Memory consumption
- Minimum state information is sufficient to produce large outputs.

- Poor Randomness Quality
- Usually, LCG has a short period
- Predictability
- Sequential numbers generation

Blum Blum Shub Generator (BBSG)

- Choose 2 large primes ('p' and 'q') such that $p \equiv q \equiv 3 \pmod 4$
- $n = p * q$
- Choose a random number 's' such that GCD(s,n) = 1

- Algorithm:-

$X_0 = s^2 \bmod n$

for i = 1 to ∞

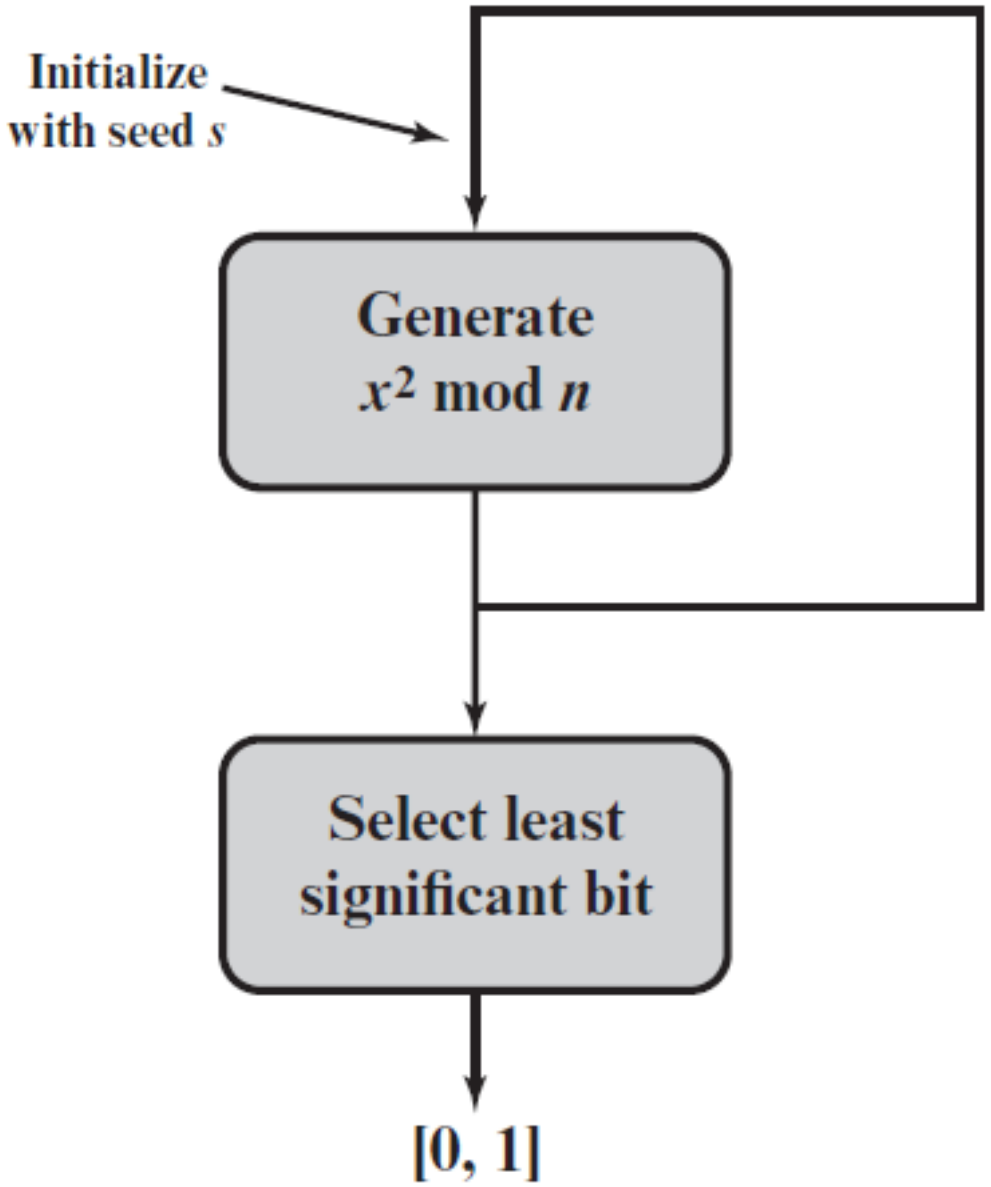
{

$X_i = (X_{i-1})^2 \bmod n$

$B_i = X_i \bmod 2$

}

BBSG (Contd..)


Initialize
with seed s
Generate
x² mod n
Select least
significant bit
[0, 1]

BBSG (Contd..)

- BBSG is referred to as a CSPRNG.
- BBSG passes the Next Bit test.
- The goal is to make the sequence of bits unpredictable for any practical algorithm.
- The security of BBS relies on the challenge of factoring a large number n into its two prime factors ‘p’ and ‘q’.

BBSG (Numerical 1)

- Generate the random bit sequence (first 3 iterations) using BBSG with following parameters:- 2 primes (7 and 11), and a seed of 12.

Solution:-

- $p = 7, q = 11, s = 12$
- $n = p*q = 7*11 = 77$
- $X_0 = s^2 \bmod n = 12^2 \bmod 77 = 67$

- $X_1 = (X_0)^2 \bmod n = 23$
- $B_1 = X_1 \bmod 2 = 1$

BBSG (Numerical 1) (Contd..)

- $X_2 = (X_1)^2 \bmod n = 67$
- $B_2 = X_2 \bmod 2 = 1$

- $X_3 = (X_2)^2 \bmod n = 23$
- $B_3 = X_3 \bmod 2 = 1$

- Sequence = {1, 1, 1, …….}

BBSG (Numerical 2)

- Generate the random bit sequence (first 10 iterations) using BBSG with following parameters:- 2 primes (31 and 59), and a seed of 45.

Solution:-

- p = 31, q = 59, s = 45
- n = p*q = 1829
- $X_0 = s^2 \bmod n = 196$

- $X_1 = (X_0)^2 \bmod n = 7$
- $B_1 = X_1 \bmod 2 = 1$

BBSG (Numerical 2) (Contd..)

- $X_2 = (X_1)^2 \bmod n = 49$
- $B_2 = X_2 \bmod 2 = 1$

- $X_3 = (X_2)^2 \bmod n = 572$
- $B_3 = X_3 \bmod 2 = 0$

- $X_4 = (X_3)^2 \bmod n = 1622$
- $B_4 = X_4 \bmod 2 = 0$

BBSG (Numerical 2) (Contd..)

- $X_5 = (X_4)^2 \bmod n = 782$
- $B_5 = X_5 \bmod 2 = 0$

- $X_6 = (X_5)^2 \bmod n = 638$
- $B_6 = X_6 \bmod 2 = 0$

- $X_7 = (X_6)^2 \bmod n = 1006$
- $B_7 = X_7 \bmod 2 = 0$

- $X_8 = (X_7)^2 \bmod n = 599$
- $B_8 = X_8 \bmod 2 = 1$

BBSG (Numerical 2) (Contd..)

- $X_9 = (X_8)^2 \bmod n = 317$
- $B_9 = X_9 \bmod 2 = 1$

- $X_{10} = (X_9)^2 \bmod n = 1723$
- $B_{10} = X_{10} \bmod 2 = 1$

- Sequence = {1, 1, 0, 0, 0, 0, 0, 1, 1, 1, ……}

BBSG (Numerical 3)

- Calculate the period of BBSG with following parameters:- 2 primes (103 and 211), and a seed of 100. Also display, the corresponding random bit sequence.

Solution:-

- $p = 103$, $q = 211$, $s = 100$
- $n = p*q = 21733$
- $X_0 = s^2 \bmod n = 10000$

- $X_1 = (X_0)^2 \bmod n = 6467$
- $B_1 = X_1 \bmod 2 = 1$

BBSG (Numerical 3) (Contd..)

- $X_2 = (X_1)^2 \bmod n = 7797$
- $B_2 = X_2 \bmod 2 = 1$

- $X_3 = (X_2)^2 \bmod n = 6008$
- $B_3 = X_3 \bmod 2 = 0$

- $X_4 = (X_3)^2 \bmod n = 19284$
- $B_4 = X_4 \bmod 2 = 0$

- $X_5 = (X_4)^2 \bmod n = 21026$
- $B_5 = X_5 \bmod 2 = 0$

BBSG (Numerical 3) (Contd..)

- $X_6 = (X_5)^2 \bmod n = 21723$
- $B_6 = X_6 \bmod 2 = 1$

- $X_7 = (X_6)^2 \bmod n = 100$
- $B_7 = X_7 \bmod 2 = 0$

- $X_8 = (X_7)^2 \bmod n = 10000$
- $B_8 = X_8 \bmod 2 = 0$

- Period = 8; Sequence = {110001001100010011000100110001001……}

Pros and Cons of BBSG

- Simple to understand and easy to implement.
- Provides high security when large primes are chosen.
- Good statistical properties
- Minimum state information is sufficient to produce large outputs.

- Management of Primes
- Management of Seed
- Sequential bits generation
- Less efficiency when Period is high

PSEUDORANDOM NUMBER GENERATION USING A BLOCK CIPHER

PRNG using Block Cipher

- A common way to create a PRNG is by using a symmetric block cipher.
- A symmetric block cipher takes a block of input PT and produces a block of output CT that seems random.
- The output shows no patterns, making it hard to guess the input from the output.
- This randomness makes symmetric block ciphers good for building PRNGs.
- Using a well-known block cipher like DES or AES helps ensure the PRNG is secure.
- Many applications already use DES or AES, making it easy to integrate them into PRNG systems.

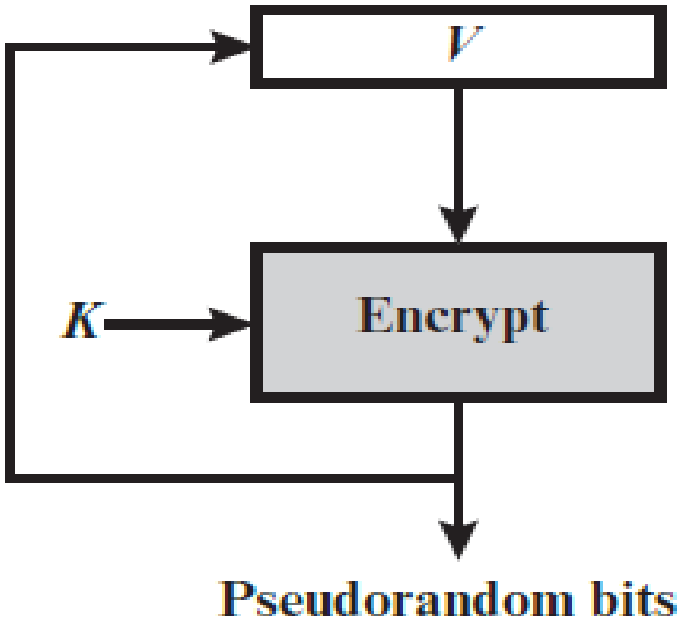
PRNG using Block Cipher (Contd..)

- Two common methods for creating a PNRG use 2 modes: CTR mode and OFB mode (seed consists of encryption key, and a value V)

- CTR mode is suggested in:-
  - NIST SP 800-90A
  - ANSI standard X9.82
  - RFC 4086 (from June 2005)

- OFB mode is recommended in:-
  - ANSI standard X9.82
  - RFC 4086

PRNG based on OFB



- V = Value of the preceding PRNG block

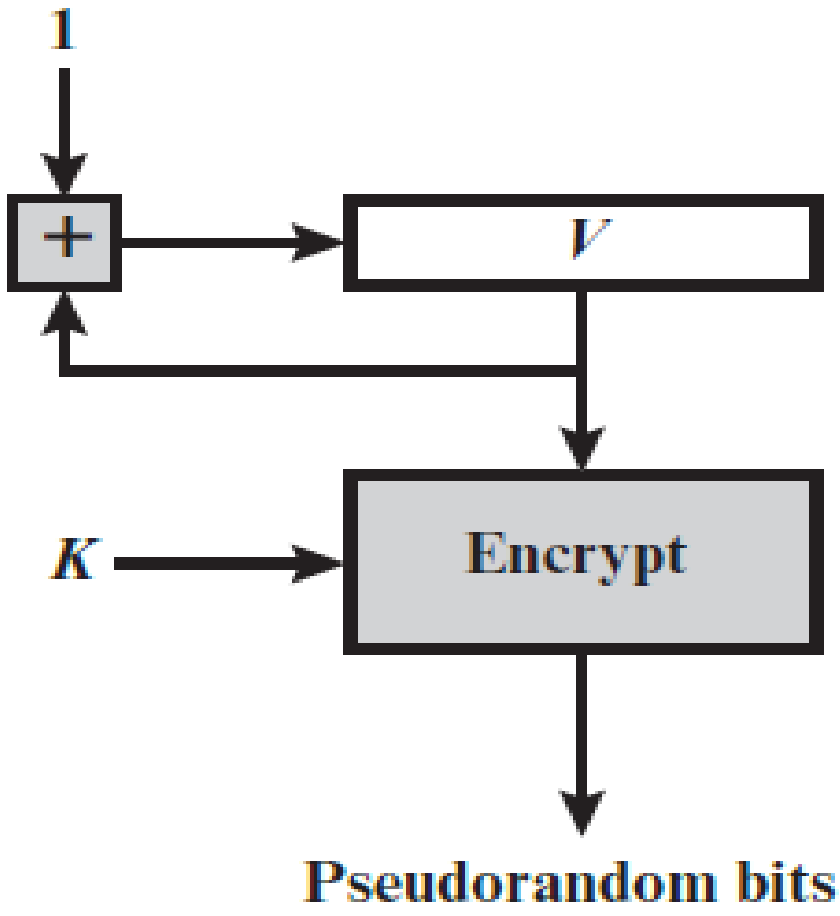
```
while (len (temp) < requested_number_of_bits) do
        V = E(Key, V)
        temp = temp || V
```

PRNG based on OFB (Example)

Key:	cfb0ef3108d49cc4562d5810b0a9af60
V:	4c89af496176b728ed1e2ea8ba27f5a4

Output Block	Fraction of One Bits	Fraction of Bits that Match with Preceding Block
1786f4c7ff6e291dbdfdd90ec3453176	0.57	–
5e17b22b14677a4d66890f87565eae64	0.51	0.52
fd18284ac82251dfb3aa62c326cd46cc	0.47	0.54
c8e545198a758ef5dd86b41946389bd5	0.50	0.44
fe7bae0e23019542962e2c52d215a2e3	0.47	0.48
14fdf5ec99469598ae0379472803accd	0.49	0.52
6aeca972e5a3ef17bd1a1b775fc8b929	0.57	0.48
f7e97badf359d128f00d9b4ae323db64	0.55	0.45

PRNG based on CTR



- ‘V’ is incremented by 1 after each encryption

PRNG based on CTR (Algorithm)

while (len (temp) < requested_number_of_bits)
{
$V = (V + 1) \bmod 2^b$
output_block = E(Key, V)
temp = temp || output_block
}

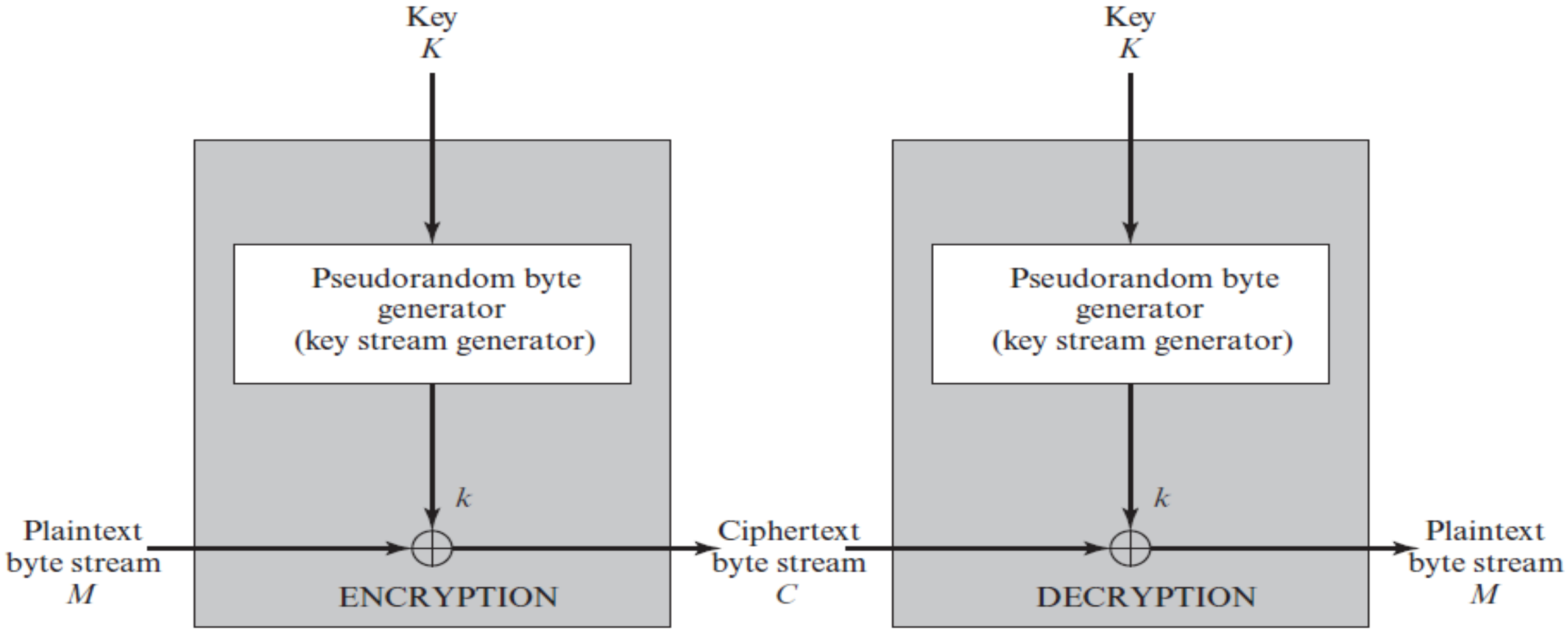
PRNG based on CTR (Example)

Key:	cfb0ef3108d49cc4562d5810b0a9af60
V:	4c89af496176b728ed1e2ea8ba27f5a4

Output Block	Fraction of One Bits	Fraction of Bits that Match with Preceding Block
1786f4c7ff6e291dbdfdd90ec3453176	0.57	–
60809669a3e092a01b463472fdcae420	0.41	0.41
d4e6e170b46b0573eedf88ee39bff33d	0.59	0.45
5f8fcfc5deca18ea246785d7fadc76f8	0.59	0.52
90e63ed27bb07868c753545bdd57ee28	0.53	0.52
0125856fdf4a17f747c7833695c52235	0.50	0.47
f4be2d179b0f2548fd748c8fc7c81990	0.51	0.48
1151fc48f90eebac658a3911515c3c66	0.47	0.45

STREAM CIPHERS

Stream Ciphers



Stream Ciphers

- A stream cipher encrypts data one byte at a time.
- It can also work on one bit at a time or on larger units of bits.
- A key is used as input to a pseudorandom bit generator which produces 8-bit random output called keystream.

- CT stream (1 Byte) = PT stream (1 Byte) $\oplus$ Keystream (1 Byte)
- DT stream (1 Byte) = CT stream (1 Byte) $\oplus$ Keystream (1 Byte)

Stream Ciphers Design Considerations

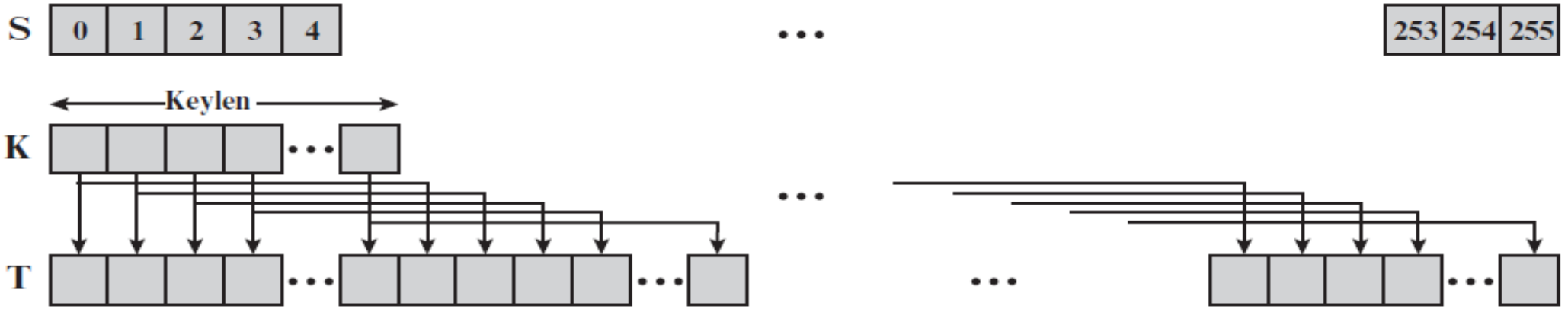
- The encryption sequence should have a large period.
- The keystream should approximate the properties of a true random number stream as close as possible.
- A key length of at least 128 bits is desirable.
- With a properly designed pseudorandom number generator a stream cipher can be as secure as a block cipher of comparable key length.

RIVEST CIPHER (RC) 4

RC4

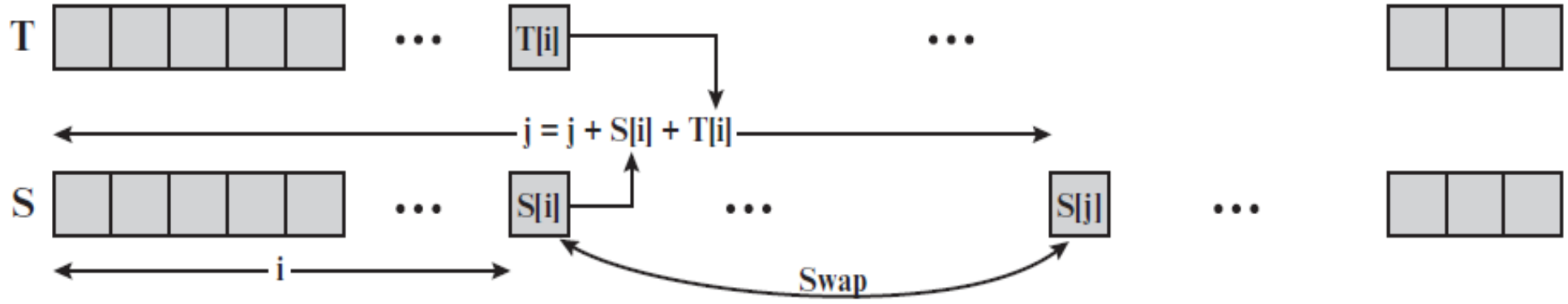
- Designed in 1987 by Ron Rivest for RSA Security
- Variable key size stream cipher with byte-oriented operations
- Based on the use of a random permutation
- Eight to sixteen machine operations are required per output byte and the cipher can be expected to run very quickly in software
- Used in the Secure Sockets Layer/Transport Layer Security (SSL/TLS) standards that have been defined for communication between Web browsers and servers
- Is also used in the Wired Equivalent Privacy (WEP) protocol and the newer WiFi Protected Access (WPA) protocol that are part of the IEEE 802.11 wireless LAN standard

RC4 (Initial State of S and T)



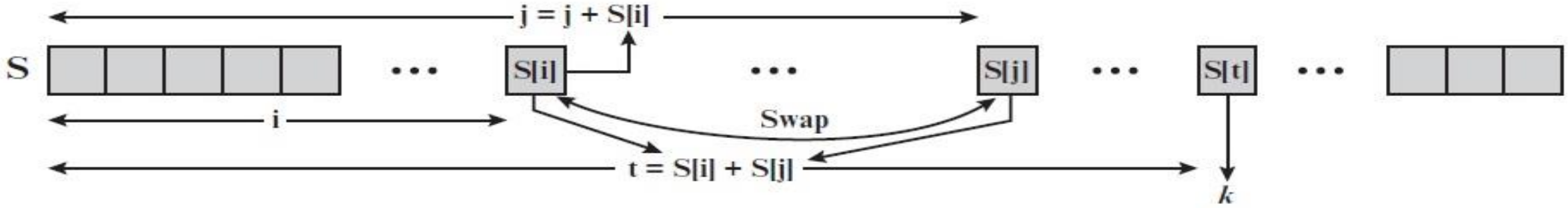

```
for i = 0 to 255
{
S[i] = i;
T[i] = K[i mod keylen];
}
```

RC4 (Initial Permutation of S)




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

RC4 (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]) mod 256;
k = S[t];
}
```

Strength of RC4

- Several papers have studied attacks on RC4 encryption.
- Attacking RC4 with a strong key (like 128 bits) is not practical.
- The WEP protocol, meant to keep data private, has vulnerabilities.
- This key generation issue mainly affects WEP, not other uses of RC4.
- Improving key generation can fix the WEP vulnerability.

Strength of RC4 (Contd..)

- A recent study found a serious problem in the RC4 key scheduling algorithm, which makes it easier to guess the encryption key.
- Another study showed that flaws in the RC4 keystream can help recover the same encrypted messages.
- Due to these weaknesses, the IETF banned RC4 for use in TLS in February 2015.
- NIST also banned RC4 for government use in their guidelines from September 2013.