

Modern stream ciphers

Prepared by Dr. Ayman Khallel

IT373

Information Security

Lecture 5



Course content

1. Overview of security concepts and definitions
2. Symmetric cryptography; modern encryption algorithms (DES, AES)
3. modern data encryption standards
4. block cipher cryptanalysis and usage
5. modern stream ciphers
6. Asymmetric cryptography; public key cryptography, key certificates and management
7. message authentication and hash functions
8. hash algorithms
9. digital signature and authentication protocols
10. electronic mail security, IP security, web security.

Agenda/Outline

- Confusion and Diffusion
- Stream Cipher vs. Block Cipher
- Understanding Keystream in Stream Ciphers
- Random Numbers in Cryptography
- Pseudorandom Numbers
- Randomness Tests
- Random Number Generation Algorithms
- Stream Cipher Design Considerations
- Modern Stream Ciphers
- Summary



Confusion and Diffusion

- Confusion

- Making the relationship between the encryption key and the ciphertext as complex as possible.
- Relationship between Ciphertext and Plaintext is obscured.
- Given Ciphertext, no info about Plaintext, Key, Encryption algorithm etc.,
- Example: Substitution.

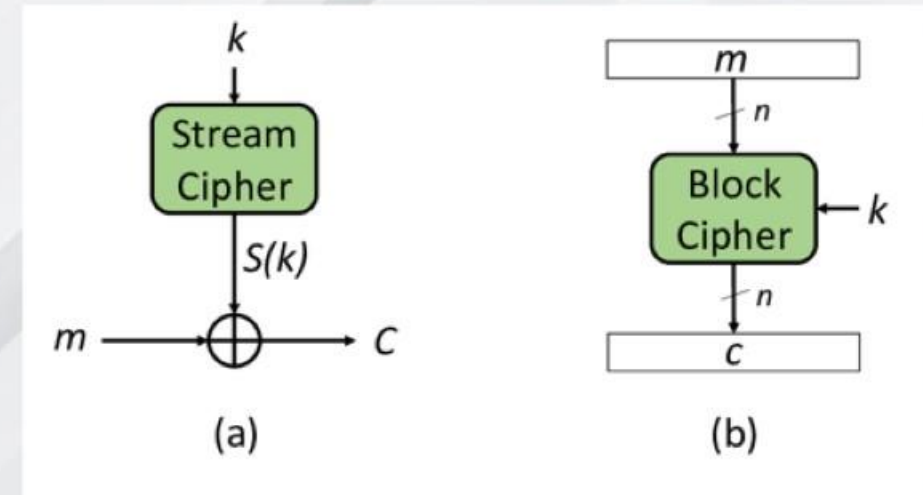
- Diffusion

- Making each plaintext bit affect as many ciphertext bits as possible.
- 1 bit change in Plaintext, significant effect on Ciphertext.
- Example: Transposition or Permutation.



Stream Cipher vs Block Cipher

- Stream Cipher each plaintext digit is encrypted one at a time with the corresponding digit of the keystream, to give a digit of the ciphertext stream.
- Block Cipher is a deterministic algorithm operating on fixed-length groups of bits, called blocks



Key:00101

||
V

keystream: 010001010101110110101



Stream Cipher vs Block Cipher

	Stream Cipher	Block Cipher
Length	Bit or Byte	Block Size 64 bits, 128 bits
Design	Complex	Simple
Principle	Confusion	Confusion and Diffusion
Speed	Faster	Slower
Example	RC4	DES, AES



Understanding Keystream in Stream Ciphers

- A **keystream** is a sequence of bits generated by a pseudorandom number generator or a stream cipher.
- Used to encrypt plaintext by combining it with the keystream to produce ciphertext.
- Key Characteristics:
 - **Pseudorandomness**: Appears random and is difficult to predict.
 - **Length**: Should be the same length as the plaintext for maximum security.
 - **Key Dependency**: Generated based on a secret key, ensuring only authorized parties can reproduce it.



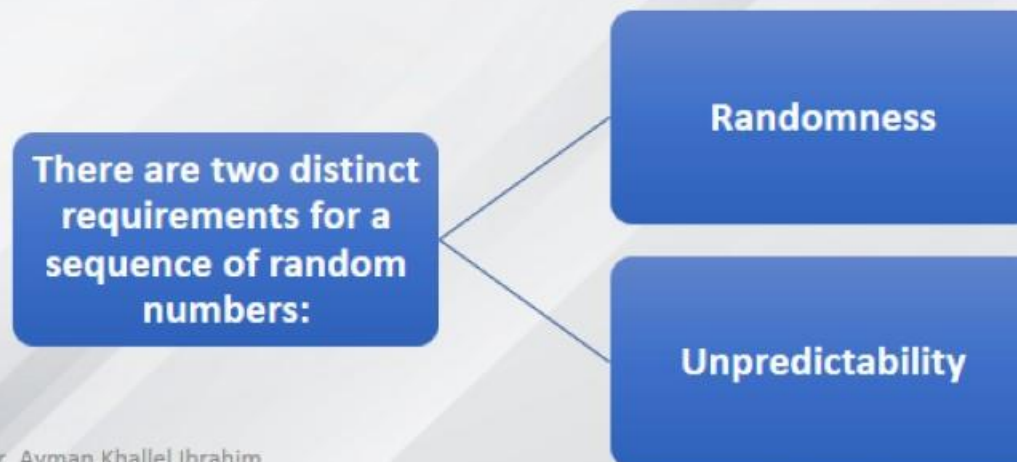
Example: XOR-Based Stream Cipher

- Plaintext (P): "HELLO"
 - Keystream (K): "XMDFL"
 - Ciphertext (C): $C = P \oplus K$
 - Decryption: $P = C \oplus K$
-
- Importance
 - Security relies on the unpredictability of the keystream.
 - Predictable keystreams allow attackers to decrypt ciphertext or alter messages undetected.



Random Numbers

- A number of network security algorithms and protocols based on cryptography make use of random binary numbers:
 - Key distribution and reciprocal authentication schemes
 - Session key generation
 - Generation of keys for the RSA public-key encryption algorithm
 - Generation of a bit stream for symmetric stream encryption



Randomness

- The generation of a sequence of allegedly random numbers being random in some well-defined statistical sense has been a concern
- Two criteria are used to validate that a sequence of numbers is random:
 - **Uniform distribution**
 - The frequency of occurrence of ones and zeros should be approximately equal
 - Example:
 - Consider the following sequence of numbers: 1,0,0,1,0,0
 - Frequencies:
 - 0s: 4 occurrences
 - 1s: 2 occurrences
 - Analysis: The distribution is not uniform, suggesting that the numbers may not be randomly generated.
 - **Independence**
 - No one subsequence in the sequence can be inferred from the others
 - Consider the following sequence: 1,0,0,1,0,0
 - *Analysis:* In this sequence, it appears that each number depends on the one before it.
This means the numbers are not independent, indicating a lack of randomness.



Unpredictability

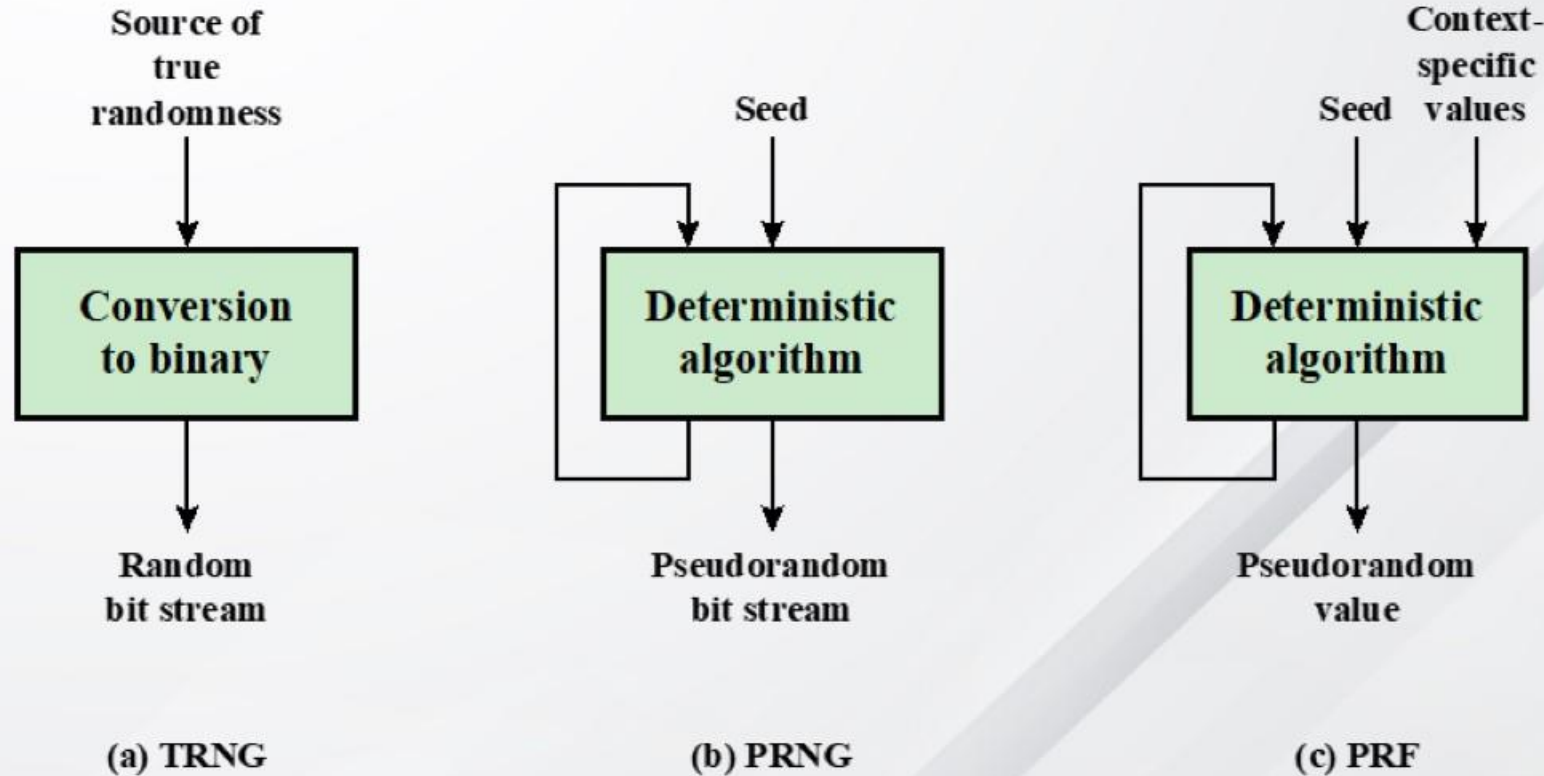
- The requirement is not just that the sequence of numbers be statistically random, but that the successive members of the sequence are unpredictable
- With “true” random sequences each number is statistically independent of other numbers in the sequence and therefore unpredictable
 - True random numbers have their limitations, such as inefficiency, so it is more common to implement algorithms that generate sequences of numbers that appear to be random
 - Care must be taken that an opponent not be able to predict future elements of the sequence on the basis of earlier elements



Pseudorandom Numbers

- Cryptographic applications typically make use of algorithmic techniques for random number generation
- These algorithms are deterministic and therefore produce sequences of numbers that are not statistically random
- If the algorithm is good, the resulting sequences will pass many tests of randomness and are referred to as *pseudorandom numbers*





TRNG = true random number generator
 PRNG = pseudorandom number generator
 PRF = pseudorandom function

Figure 8.1 Random and Pseudorandom Number Generators



True Random Number Generator (TRNG)

- Takes as input a source that is effectively random
- The source is referred to as an *entropy source* and is drawn from the physical environment of the computer
 - Includes things such as keystroke timing patterns, disk electrical activity, mouse movements, and instantaneous values of the system clock
 - The source, or combination of sources, serve as input to an algorithm that produces random binary output
- The TRNG may simply involve conversion of an analog source to a binary output
- The TRNG may involve additional processing to overcome any bias in the source



Pseudorandom Number Generator (PRNG)

- Takes as input a fixed value, called the *seed*, and produces a sequence of output bits using a deterministic algorithm
 - Quite often the seed is generated by a TRNG
- The output bit stream is determined solely by the input value or values, so an adversary who knows the algorithm and the seed can reproduce the entire bit stream
- Other than the number of bits produced there is no difference between a PRNG and a PRF

Two different forms of PRNG

Pseudorandom number generator

- An algorithm that is used to produce an open-ended sequence of bits
- Input to a symmetric stream cipher is a common application for an open-ended sequence of bits

Pseudorandom function (PRF)

- Used to produce a pseudorandom string of bits of some fixed length
- Examples are symmetric encryption keys and nonces



PRNG/PRF Requirements

- The basic requirement when a PRNG or PRF is used for a cryptographic application is that an adversary who does not know the seed is unable to determine the pseudorandom string
- The requirement for secrecy of the output of a PRNG or PRF leads to specific requirements in the areas of:
 - Randomness
 - Unpredictability
 - Characteristics of the seed



Randomness

- The generated bit stream needs to appear random even though it is deterministic
- There is no single test that can determine if a PRNG generates numbers that have the characteristic of randomness
 - If the PRNG exhibits randomness on the basis of multiple tests, then it can be assumed to satisfy the randomness requirement
- NIST SP 800-22 specifies that the tests should seek to establish three characteristics:
 - Uniformity
 - Scalability
 - Consistency



Randomness Tests

- SP 800-22 lists 15 separate tests of randomness

Frequency test

- The most basic test and must be included in any test suite
- Purpose is to determine whether the number of ones and zeros in a sequence is approximately the same as would be expected for a truly random sequence

Runs test

- Focus of this test is the total number of runs in the sequence, where a run is an uninterrupted sequence of identical bits bounded before and after with a bit of the opposite value
- Purpose is to determine whether the number of runs of ones and zeros of various lengths is as expected for a random sequence

Maurer's universal statistical test

- Focus is the number of bits between matching patterns
- Purpose is to detect whether or not the sequence can be significantly compressed without loss of information. A significantly compressible sequence is considered to be non-random

Three
tests



Unpredictability

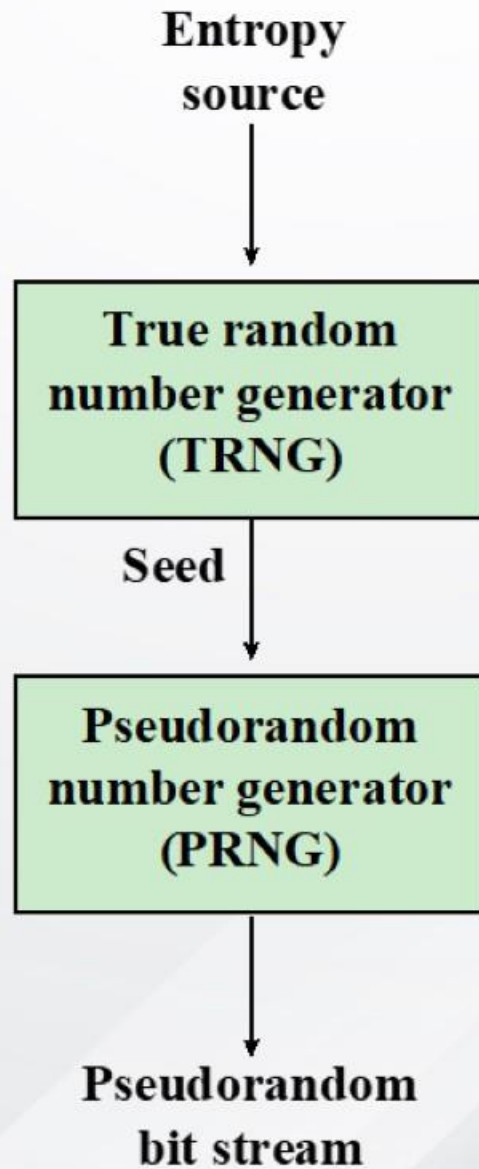
- A stream of pseudorandom numbers should exhibit two forms of unpredictability:
 - Forward unpredictability
 - If the seed is unknown, the next output bit in the sequence should be unpredictable in spite of any knowledge of previous bits in the sequence
 - Backward unpredictability
 - It should not be feasible to determine the seed from knowledge of any generated values
 - No correlation between a seed and any value generated from that seed should be evident
 - Each element of the sequence should appear to be the outcome of an independent random event whose probability is $1/2$
- The same set of tests for randomness also provides a test of unpredictability
 - A random sequence will have no correlation with a fixed value (the seed)



Seed Requirements

- The seed that serves as input to the PRNG must be secure and unpredictable
- The seed itself must be a random or pseudorandom number
- Typically the seed is generated by TRNG





- **Why PRNG over TRNG in Stream Ciphers:**

- Using TRNG directly isn't practical for stream ciphers, as it would require generating and securely transmitting a keystream matching the plaintext's length.
- With PRNG (Pseudorandom Number Generator), only the stream cipher key (128 or 256 bits) needs secure delivery.
- PRNG output is used to reduce possible biases in TRNG output, improving randomness quality.
- TRNG may not always match the application's demand for random bit rate; PRNG compensates by producing bits faster when needed.



Figure 8.2 Generation of Seed Input to PRNG

Algorithm Design

- Algorithms fall into two categories:
 - Purpose-built algorithms
 - Algorithms designed specifically and solely for the purpose of generating pseudorandom bit streams
 - Algorithms based on existing cryptographic algorithms
 - Have the effect of randomizing input data

Three broad categories of cryptographic algorithms are commonly used to create PRNGs:

- Symmetric block ciphers
- Asymmetric ciphers
- Hash functions and message authentication codes



Linear Congruential Generator

- An algorithm first proposed by Lehmer that is parameterized with four numbers:

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

- The sequence of random numbers $\{X_n\}$ is obtained via the following iterative equation:

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

- If m , a , c , and X_0 are integers, then this technique will produce a sequence of integers with each integer in the range $0 \leq X_n < m$
- The selection of values for a , c , and m is critical in developing a good random number generator



Linear Congruential Generator

- Let's use an example with the following parameters:

- Seed (X_0) = 5
- Multiplier (a) = 7
- Increment (c) = 3
- Modulus (m) = 10

- $X_{n+1} = (aX_n + c) \bmod m$
- $X_1 = (aX_0 + c) \bmod m = (7 \times 5 + 3) \bmod 10 = (35 + 3) \bmod 10 = 38 \bmod 10 = 8$
- $X_2 = (aX_1 + c) \bmod m = (7 \times 8 + 3) \bmod 10 = (56 + 3) \bmod 10 = 59 \bmod 10 = 9$
- $X_3 = (aX_2 + c) \bmod m = (7 \times 9 + 3) \bmod 10 = (63 + 3) \bmod 10 = 66 \bmod 10 = 6$



•

•

Blum Blum Shub (BBS) Generator

- Has perhaps the strongest public proof of its cryptographic strength of any purpose-built algorithm
- Referred to as a *cryptographically secure pseudorandom bit generator* (CSPRBG)
 - A CSPRBG is defined as one that passes the *next-bit-test* if there is not a polynomial-time algorithm that, on input of the first k bits of an output sequence, can predict the $(k + 1)$ st bit with probability significantly greater than $1/2$
- The security of BBS is based on the difficulty of factoring n



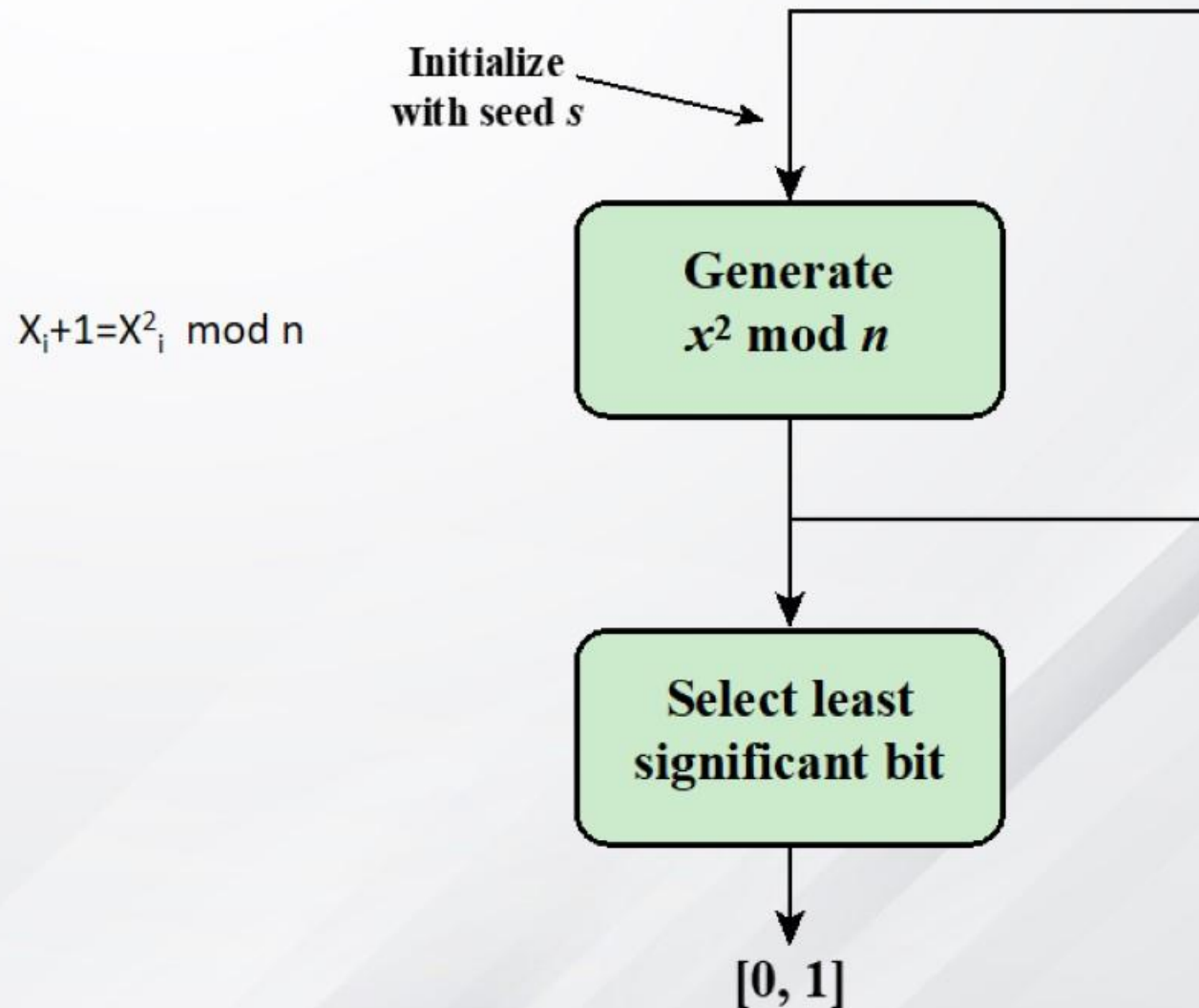


Figure 8.3 Blum Blum Shub Block Diagram



Table 8.1 Example Operation of BBS Generator

$$X_{i+1} = X_i^2 \text{ mod } n$$

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

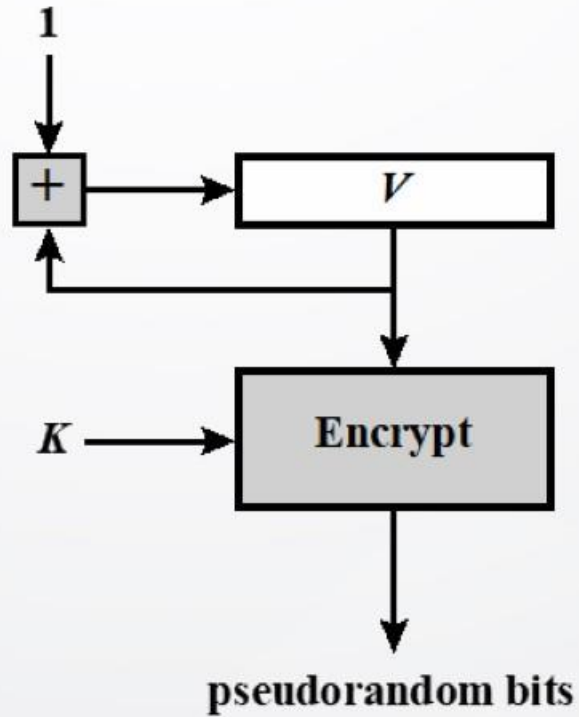
i	X_i	B_i
11	137922	0
12	123175	1
13	8630	0
14	114386	0
15	14863	1
16	133015	1
17	106065	1
18	45870	0
19	137171	1
20	48060	0



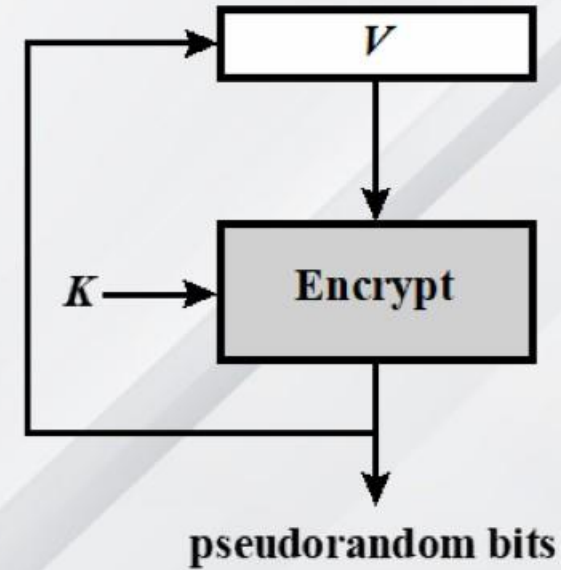
PRNG Using Block Cipher Modes of Operation

- Two approaches that use a block cipher to build a PRNG have gained widespread acceptance:
 - CTR mode
 - Recommended in NIST SP 800-90, ANSI standard X.82, and RFC 4086
 - OFB mode
 - Recommended in X9.82 and RFC 4086





(a) CTR Mode



(b) OFB Mode

Figure 8.4 PRNG Mechanisms Based on Block Ciphers



Table 8.2 Example Results for PRNG Using OFB

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



Table 8.3 Example Results for PRNG Using CTR

Output Block	Fraction of One Bits	Fraction of Bits that Match with Preceding Block
1786f4c7ff6e291dbdfdd90ec3453176	0.57	—
60809669a3e092a01b463472fdcae420	0.41	0.41
d4e6e170b46b0573eedf88ee39bfff33d	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



NIST CTR_DRBG

- **Purpose:** Counter Mode Deterministic Random Bit Generator (CTR_DRBG) provides secure pseudo-random numbers using CTR mode.
- **Entropy Source:** Depends on a True Random Number Generator (TRNG) or equivalent source to ensure sufficient entropy.
- **Algorithm Options:** Uses either 3DES (three keys) or AES (128, 192, or 256 bits) for encryption.
- **Key Parameters:**
 - **Output Block Length (outlen):** Size of each generated output block.
 - **Key Length (keylen):** Encryption key size.
 - **Seed Length (seedlen):** Controls DRBG's internal state (seedlen = outlen + keylen).
 - **Reseed Interval:** Maximum output blocks before reseeding.
- **Application:** Implemented in Intel processors' hardware random number generators.

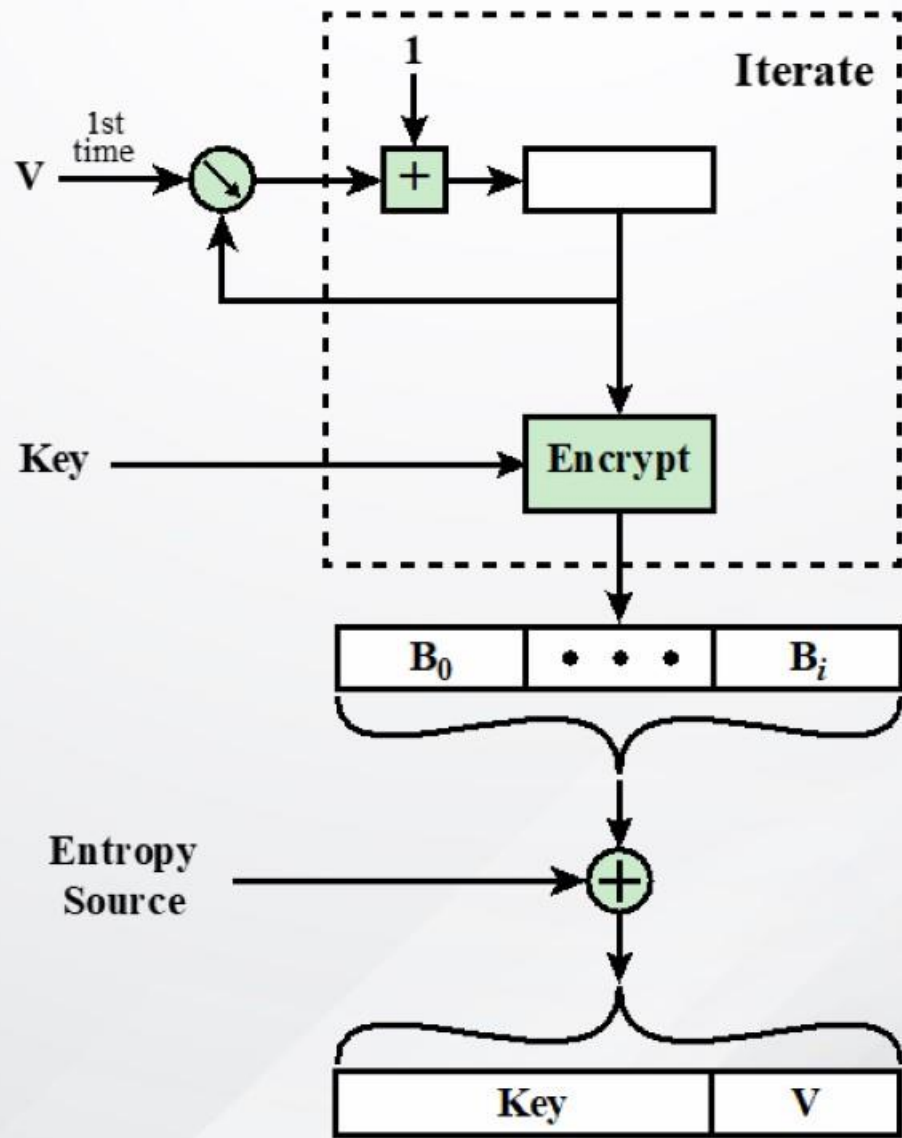


Table 8.4

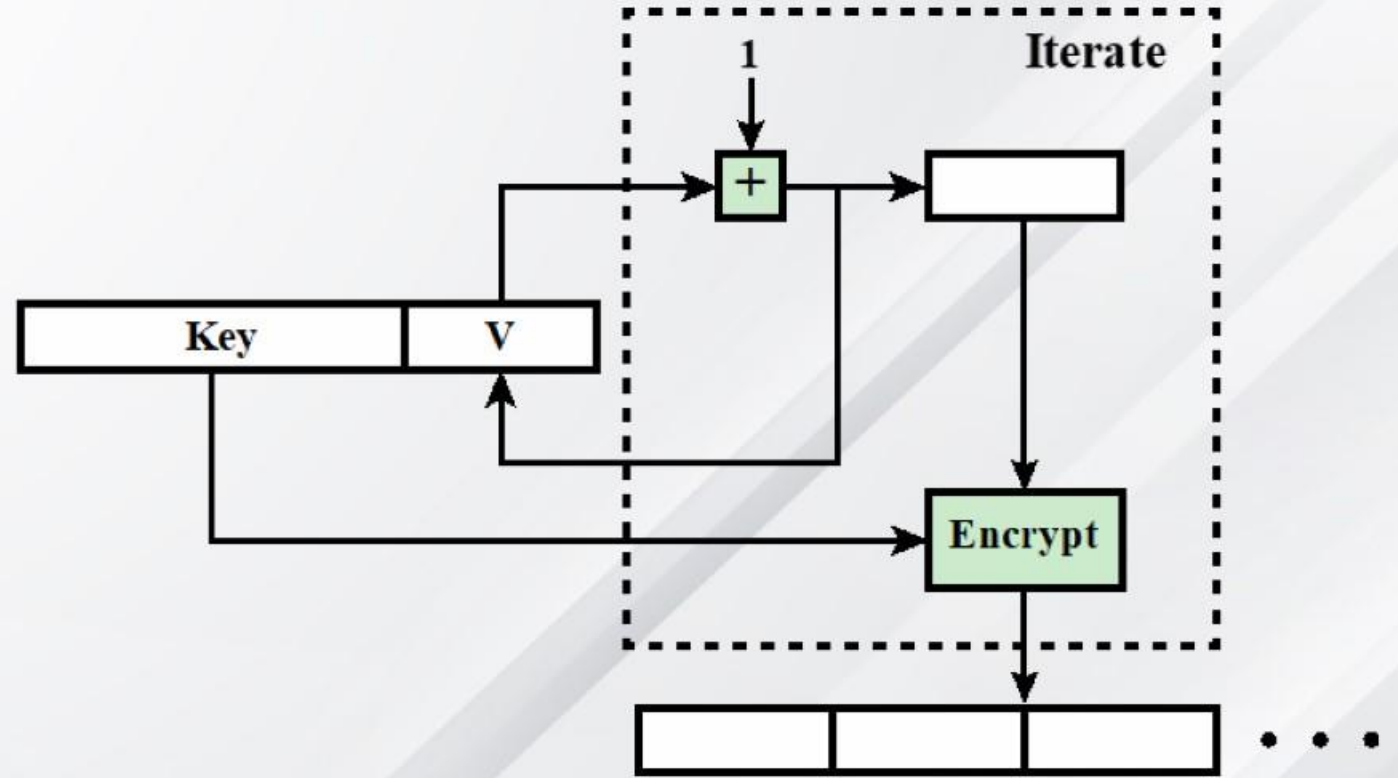
	3DES	AES-128	AES-192	AES-256
<i>outlen</i>	64	128	128	128
<i>keylen</i>	168	128	192	256
<i>seedlen</i>	232	256	320	384
<i>reseed_interval</i>	$\leq 2^{32}$	$\leq 2^{48}$	$\leq 2^{48}$	$\leq 2^{48}$

CTR_DRBG Parameters





(a) Initialize and update function



(b) Generate function



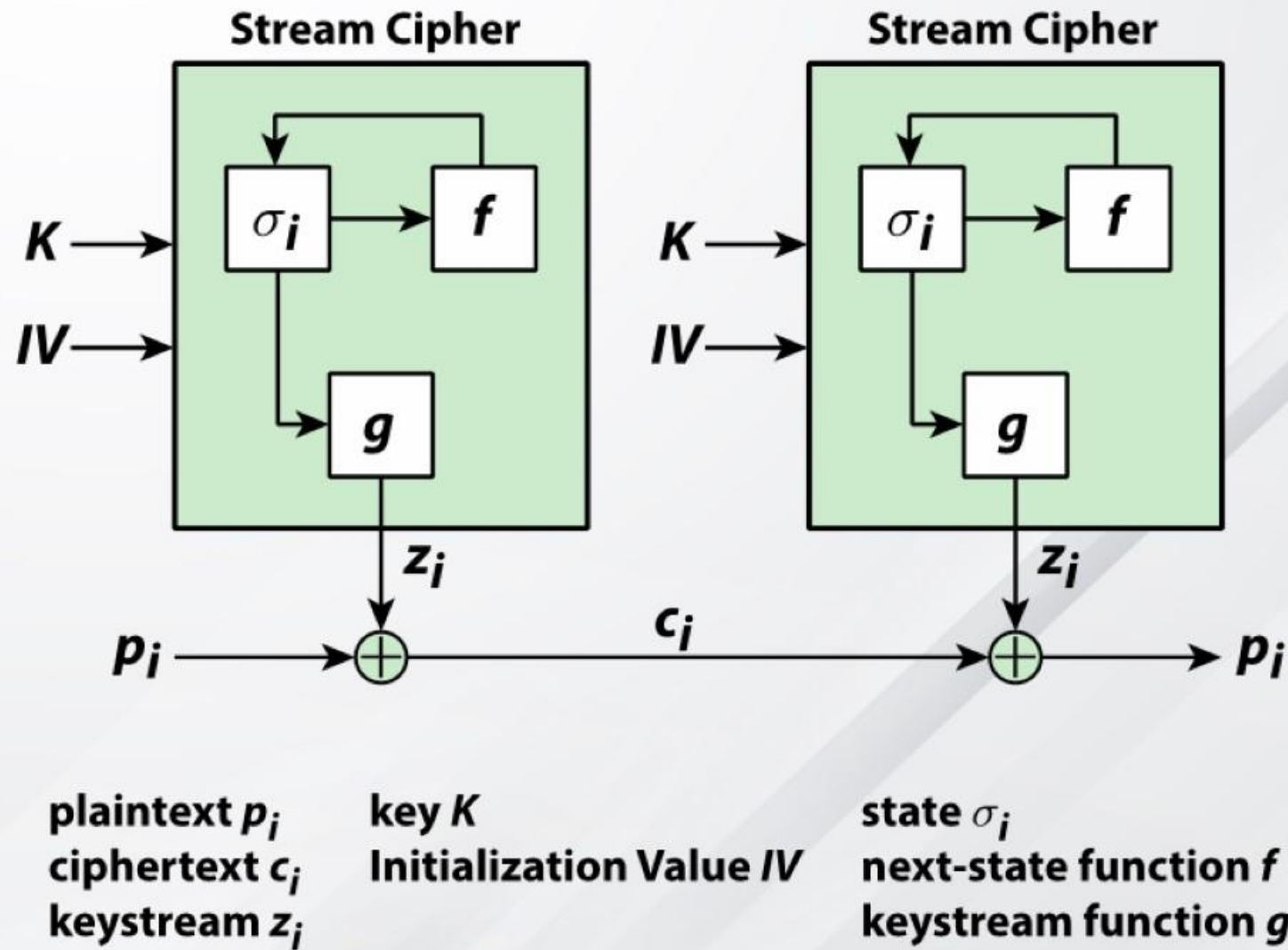


Figure 8.6 Generic Structure of a Typical Stream Cipher



Stream Cipher Design Considerations

The encryption sequence should have a large period

- A pseudorandom number generator uses a function that produces a deterministic stream of bits that eventually repeats; the longer the period of repeat the more difficult it will be to do cryptanalysis

The keystream should approximate the properties of a true random number stream as close as possible

- There should be an approximately equal number of 1s and 0s
- If the keystream is treated as a stream of bytes, then all of the 256 possible byte values should appear approximately equally often

A key length of at least 128 bits is desirable

- The output of the pseudorandom number generator is conditioned on the value of the input key
- The same considerations that apply to block ciphers are valid

With a properly designed pseudorandom number generator a stream cipher can be as secure as a block cipher of comparable key length

- A potential advantage is that stream ciphers that do not use block ciphers as a building block are typically faster and use far less code than block ciphers



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
- RC4 is used in the WiFi Protected Access (WPA) protocol that are part of the IEEE 802.11 wireless LAN standard
- It is optional for use in Secure Shell (SSH) and Kerberos
- RC4 was kept as a trade secret by RSA Security until September 1994 when the RC4 algorithm was anonymously posted on the Internet on the Cypherpunks anonymous remailers list



RC4

RC4 consists of two (three) parts

1. Key-scheduling algorithm (KSA)
2. Pseudo-random generation algorithm (PRGA)
3. (a) Encryption or (b) decryption

Variables: state array S, index pointers i and j, key, output keystream K

//Suppose key = {1, 2, 3, 4, 5, 6, 7, 8}

1. Key-scheduling algorithm (KSA)

for i from 0 to 255

s[i] := i

j := 0

for i from 0 to 255

j := (j + S[i] + key [i mod keylength]) mod 256

swap values of S[i] and S[j]

00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
10	11	12	13	14	15	16	17	18	19	1a	1b	1c	1d	1e	1f
20	21	22	23	24	25	26	27	28	29	2a	2b	2c	2d	2e	2f
30	31	32	33	34	35	36	37	38	39	3a	3b	3c	3d	3e	3f
40	41	42	43	44	45	46	47	48	49	4a	4b	4c	4d	4e	4f
50	51	52	53	54	55	56	57	58	59	5a	5b	5c	5d	5e	5f
60	61	62	63	64	65	66	67	68	69	6a	6b	6c	6d	6e	6f
70	71	72	73	74	75	76	77	78	79	7a	7b	7c	7d	7e	7f
80	81	82	83	84	85	86	87	88	89	8a	8b	8c	8d	8e	8f
90	91	92	93	94	95	96	97	98	99	9a	9b	9c	9d	9e	9f
a0	a1	a2	a3	a4	a5	a6	a7	a8	a9	aa	ab	ac	ad	ae	af
b0	b1	b2	b3	b4	b5	b6	b7	b8	b9	ba	bb	bc	bd	be	bf
c0	c1	c2	c3	c4	c5	c6	c7	c8	c9	ca	cb	cc	cd	ce	cf
d0	d1	d2	d3	d4	d5	d6	d7	d8	d9	da	db	dc	dd	de	df
e0	e1	e2	e3	e4	e5	e6	e7	e8	e9	ea	eb	ec	ed	ee	ef
f0	f1	f2	f3	f4	f5	f6	f7	f8	f9	fa	fb	fc	fd	fe	ff

01	48	b3	0c	15	53	2d	a6	dd	56	27	66	ab	7e	93	d1
7b	ce	e3	51	7d	2b	3a	16	fc	1e	0f	b4	ea	f4	8d	40
3d	49	86	b5	75	bc	33	65	f0	e5	9a	70	63	5f	6a	c0
73	8e	7f	28	96	5e	be	30	a7	e2	3b	41	5d	89	d2	d8
74	fb	18	37	bb	50	21	b8	4d	e7	58	b2	d6	09	90	67
c8	54	a1	1f	98	ac	25	34	2e	0b	2a	6b	6d	1a	08	9c
b7	92	c5	6f	db	c2	55	b0	47	c1	87	0a	a3	20	bf	d5
1d	8f	29	0e	a9	81	a4	88	71	fa	91	44	7a	79	f3	af
f2	4e	13	de	57	da	31	42	e1	77	ad	2c	cd	78	36	bd
9d	cc	fd	ef	d4	97	b1	07	a8	ec	e6	ee	52	83	f9	a0
82	b6	d0	8c	06	fe	3e	2f	d9	9f	a2	ae	e0	16	94	00
12	f7	8a	39	10	e4	85	61	d7	f1	05	d3	dc	ea	a5	eb
26	68	17	11	3c	df	7c	03	5b	ed	76	04	e8	02	c4	24
1c	aa	4f	c9	45	62	f5	14	c3	ca	64	95	4a	19	cf	ba
9b	46	35	cb	5a	b9	6c	4c	8b	c7	c6	32	3f	84	ff	6e
0d	80	43	22	23	4b	38	e9	5c	99	60	59	72	f8	69	1b

RC4

RC4 consists of two (three) parts

1. Key-scheduling algorithm (KSA)
2. Pseudo-random generation algorithm (PRGA)
3. (a) Encryption or (b) decryption

Variables: state array S, index pointers i and j, key, output keystream K

2. Pseudo-random generation algorithm (PRGA)

i := 0 j := 0

while GeneratingOutput:

i := (i + 1) mod 256

j := (j + S(i)) mod 256

swap values of S(i) and S[j]

K := S[(S[i] + S[j]) mod 256]

output K

i = 1 , j = 72 , K = 97

01	48	b3	0c	15	53	2d	a6	dd	56	27	66	ab	7e	93	d1
7b	ce	e3	51	7d	2b	3a	16	fc	1e	0f	b4	ea	f4	8d	40
3d	49	86	b5	75	bc	33	65	f0	e5	9a	70	63	5f	6a	c0
73	8e	7f	28	96	5e	be	30	a7	e2	3b	41	5d	89	d2	d8
74	fb	18	37	bb	50	21	b8	4d	e7	58	b2	d6	09	90	67
c8	54	a1	1f	98	ac	25	34	2e	0b	2a	6b	6d	1a	08	9c
b7	92	c5	6f	db	c2	55	b0	47	c1	87	0a	a3	20	bf	d5
1d	8f	29	0e	a9	81	a4	88	71	fa	91	44	7a	79	f3	af
f2	4e	13	de	57	da	31	42	e1	77	ad	2c	cd	78	36	bd
9d	cc	fd	ef	d4	97	b1	07	a8	ec	e6	ee	52	83	f9	a0
82	b6	d0	8c	06	fe	3e	2f	d9	9f	a2	ae	e0	f6	94	00
12	f7	8a	39	10	e4	85	61	d7	f1	05	d3	dc	9e	a5	eb
26	68	17	11	3c	df	7c	03	5b	ed	76	04	e8	02	c4	24
1c	aa	4f	c9	45	62	f5	14	c3	ca	64	95	4a	19	cf	ba
9b	46	35	cb	5a	b9	6c	4c	8b	c7	c6	32	3f	84	ff	6e
0d	80	43	22	23	4b	38	e9	5c	99	60	59	72	f8	69	1b

01	4d	b3	0c	15	53	2d	a6	dd	56	27	66	ab	7e	93	d1
7b	ce	e3	51	7d	2b	3a	16	fc	1e	0f	b4	ea	f4	8d	40
3d	49	86	b5	75	bc	33	65	f0	e5	9a	70	63	5f	6a	c0
73	8e	7f	28	96	5e	be	30	a7	e2	3b	41	5d	89	d2	d8
74	fb	18	37	bb	50	21	b8	48	e7	58	b2	d6	09	90	67
c8	54	a1	1f	98	ac	25	34	2e	0b	2a	6b	6d	1a	08	9c
b7	92	c5	6f	db	c2	55	b0	47	c1	87	0a	a3	20	bf	d5
1d	8f	29	0e	a9	81	a4	88	71	fa	91	44	7a	79	f3	af
f2	4e	13	de	57	da	31	42	e1	77	ad	2c	cd	78	36	bd
9d	cc	fd	ef	d4	97	b1	07	a8	ec	e6	ee	52	83	f9	a0
82	b6	d0	8c	06	fe	3e	2f	d9	9f	a2	ae	e0	f6	94	00
12	f7	8a	39	10	e4	85	61	d7	f1	05	d3	dc	9e	a5	eb
26	68	17	11	3c	df	7c	03	5b	ed	76	04	e8	02	c4	24
1c	aa	4f	c9	45	62	f5	14	c3	ca	64	95	4a	19	cf	ba
9b	46	35	cb	5a	b9	6c	4c	8b	c7	c6	32	3f	84	ff	6e
0d	80	43	22	23	4b	38	e9	5c	99	60	59	72	f8	69	1b

RC4

RC4 consists of two (three) parts

1. Key-scheduling algorithm (KSA)
2. Pseudo-random generation algorithm (PRGA)
3. (a) Encryption or (b) decryption

Variables: state array S, index pointers i and j, key,
output keystream K

3. (a) Encryption or (b) decryption

Final keystream (10 first bytes) of RC4 with a
key = {1, 2, 3, 4, 5, 6, 7, 8} is

97 ab 8a 1b f0 af b9 61 32 f2

- "HELLOWORLD" converted to numbers is

48 45 4c 4c 4f 57 4f 52 4c 44

- To encrypt (or decrypt), we use the XOR operation

97 ab 8a 1b f0 af b9 61 32 f2

$\oplus$ 48 45 4c 4c 4f 57 4f 52 4c 44

df ee c6 57 bf f8 f6 33 7e b6



Strength of RC4

- A fundamental vulnerability was revealed in the RC4 key scheduling algorithm that reduces the amount of effort to discover the key
- Recent cryptanalysis results exploit biases in the RC4 keystream to recover repeatedly encrypted plaintexts
- As a result of the discovered weaknesses the IETF issued RFC 7465 prohibiting the use of RC4 in TLS
- In its latest TLS guidelines, NIST also prohibited the use of RC4 for government use



Modern Stream Ciphers

- **ChaCha20:**

- Developed by Daniel J. Bernstein.
- Widely used in TLS 1.3, Google Quic, and secure messaging apps like Signal.
- Known for its speed and resistance to cryptanalysis.

- **Salsa20:**

- Predecessor to ChaCha20, also designed by Bernstein.
- Still used in some legacy applications due to its high performance.

- **ZUC:**

- Standardized for use in LTE (4G) networks.
- Efficient and secure, playing a key role in mobile communications.

- **Grain:**

- Lightweight cipher suitable for IoT devices.
- Low resource requirements make it ideal for constrained environments.



Summary

- **Key Concepts:**
 - **Importance of Random Numbers:**
 - Essential for cryptography, simulations, and statistical sampling.
 - **Types of Generators:**
 - **True Random Number Generators (TRNGs):** Rely on physical phenomena.
 - **Pseudorandom Number Generators (PRNGs):** Deterministic sequences that approximate randomness.
 - **PRNG Characteristics:**
 - Speed and efficiency
 - Long periods before repetition
 - Statistical randomness
- **Notable PRNG Types:**
 - **Linear Congruential Generators (LCGs):** Simple but may have poor randomness.
 - **Blum Blum Shub Generator:** Secure, based on prime factorization.
 - **Block Cipher Methods:**
 - **ANSI X9.17 and NIST CTR_DRBG:** Use block ciphers for secure PRNG.
- **Stream Ciphers:**
 - **RC4:** Initialization, stream generation, and vulnerabilities.
- **Modern Stream Ciphers:**
 - **ChaCha20:** Fast and secure, used in TLS 1.3.





Thank You

