Introduction to Cryptography
Lecture 3

Monika K. Polak

February 2, 2021

R·I·T

Computer Science @ RIT
B. Thomas Golisano College of Computing & Information Sciences

Content of this Lecture

- Properties of Modular Arithmetic
- Shift (or Caesar) Cipher, Affine Cipher
- Euclidean algorithm
- Hill cipher
- Intro to stream ciphers
- Intro to random number generators (RNGs)
- One-Time Pad (OTP)

Greatest Common Divisor (GCD)

The formal definition of the GCD is:

$$gcd(a, b) = \max[k, \text{ such that } k|a \text{ and } k|b].$$

We stated that two integers a and b are `relatively prime` if their only common positive integer factor is 1. This is equivalent to saying that a and b are relatively prime if

$$gcd(a, b) = 1.$$

Properties of Modular Arithmetic

- How do we perform modular division?

First, note that rather than performing a division, we prefer to multiply by the inverse

$$b/a \equiv b \times a^{-1} \mod m.$$

The inverse a^{-1} of a number a is defined such that:

$$a \times a^{-1} \equiv 1 \mod m.$$

`Example:` What is 5 / 7 mod 9 ?
The inverse of 7 mod 9 is 4 since $7 \times 4 \equiv 28 \equiv 1 \mod 9$, hence:

$$5/7 \equiv 5 \times 4 = 20 \equiv 2 \mod 9.$$

- How is the inverse compute?

The inverse of a number a mod m only exists if and only if:

$$gcd(a, m) = 1$$

(note that $gcd(7, 9) = 1$, so that the inverse of 7 exists modulo 9).

Properties of Modular Arithmetic

- Modular reduction can be performed at any point during a calculation

Example: We want to compute $3^8 \mod 7$.

1. Approach: Exponentiation followed by modular reduction

$$3^8 = 6561 \equiv 2 \mod 7$$

Note that we have the intermediate result 6561 even though we know that the final result can't be larger than 6.

2. Approach: Exponentiation with intermediate modular reduction

$$3^8 = 3^4 \times 3^4 = 81x81 \equiv 4 \times 4 \mod 7 = 16 \mod 7 = 2 \mod 7$$

Note that we can perform all these multiplications without pocket calculator, whereas mentally computing $3^8 = 6561$ is a bit challenging for most of us.

General rule: For most algorithms it is advantageous to reduce intermediate results as soon as possible.

An Algebraic View on Modulo Arithmetic: The Ring $\mathbb{Z}_m$

The integer ring $\mathbb{Z}_m$ has the following properties:

- `Closure:` We can add and multiply any two numbers and the result is always in the ring.
- `Addition and multiplication are associative`, i.e., for all $a, b, c \in \mathbb{Z}_m$

$$a + (b + c) = (a + b) + c, \qquad a \times (b \times c) = (a \times b) \times c$$

  and addition is `commutative:` $a + b = b + a$
- `The distributive law holds:` $a \times (b + c) = (a \times b) + (a \times c)$ for all $a, b, c \in \mathbb{Z}_m$
- There is the `neutral element 0` with respect to addition, i.e., for all $a \in \mathbb{Z}_m$ $a + 0 \equiv a \mod m$
- For all $a \in \mathbb{Z}_m$, there is always an `additive inverse` element $-a$ such that $a + (-a) \equiv 0 \mod m$
- There is the `neutral element 1` with respect to multiplication. For all $a \in \mathbb{Z}_m$ $a \times 1 \equiv a \mod m$
- The `multiplicative inverse` a^{-1}
  $a \times a^{-1} \equiv 1 \mod m$
  exists only for some, but not for all, elements in $\mathbb{Z}_m$.

Modulo Arithmetic: multiplicative inverse

MULTIPLICATION MOD 7

	1	2	3	4	5	6	(inverse)
1	*1	2	3	4	5	6	1
2	2	4	6	*1	3	5	4
3	3	6	2	5	*1	4	5
4	4	*1	5	2	6	3	2
5	5	3	*1	6	4	2	3
6	6	5	4	3	2	*1	6

MULTIPLICATION MOD 8

	1	2	3	4	5	6	7	(inverse)
1	*1	2	3	4	5	6	7	1
2	2	4	6	0	2	4	6	No Inverse
3	3	6	*1	4	7	2	5	3
4	4	0	4	0	4	0	4	No Inverse
5	5	2	7	4	*1	6	3	5
6	6	4	2	0	6	4	2	No Inverse
7	7	6	5	4	3	2	*1	7

More at

https://www.cs.rit.edu/ spr/COURSES/CRYPTO/modtabs.txt

R·I·T

Computer Science @ RIT

An Algebraic View on Modulo Arithmetic: The Ring $\mathbb{Z}_m$

Roughly speaking, a ring is a structure in which we can always add, subtract and multiply, but we can only divide by certain elements (namely by those for which a multiplicative inverse exists).

- We recall from above that an element $a \in \mathbb{Z}_m$ has a multiplicative inverse only if:

$$gcd(a, m) = 1$$

  We say that a is coprime or relatively prime to m.

Example: We consider the ring $\mathbb{Z}_9 = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$. The elements 0, 3, and 6 do not have inverses since they are not coprime to 9. The inverses of the other elements 1, 2, 4, 5, 7, and 8 are:

$$1^{-1} \equiv 1 \mod 9 \quad 2^{-1} \equiv 5 \mod 9 \quad 4^{-1} \equiv 7 \mod 9$$
$$5^{-1} \equiv 2 \mod 9 \quad 7^{-1} \equiv 4 \mod 9 \quad 8^{-1} \equiv 8 \mod 9$$

Shift (or Caesar) Cipher

- Ancient cipher, allegedly used by Julius Caesar.
- Replaces each plaintext letter by another one.
- Replacement rule is very simple: Take letter that follows after k positions in the alphabet.
  Needs mapping from letters to numbers:

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25

`Example` $k = 7$
Plaintext $=$ `ATTACK` $= 0, 19, 19, 0, 2, 10$
Ciphertext $=$ `haahjr` $= 7, 0, 0, 7, 9, 17$
Note that the letters "wrap around" at the end of the alphabet, which can be mathematically be expressed as reduction modulo 26, e.g.,

$$19 + 7 = 26 \equiv 0 \mod 26.$$

Shift (or Caesar) Cipher

- Elegant mathematical (:D) description of the cipher.
  Let $k, x, y \in 0, 1, \ldots, 25 = \mathbb{Z}_{26}$
  - Encryption: $y = e_k(x) \equiv x + k \mod 26$
  - Decryption: $x = d_k(y) \equiv y - k \mod 26$
- The shift cipher is NOT secure.
  - Exhaustive key search (key space is `only 26`).
  - Letter frequency analysis attack.

Affine Cipher

- Extension of the shift cipher: rather than just adding the key to the plaintext, we also multiply by the key.

  We use for this a key consisting of two parts: $k = (a, b)$ Let $k, x, y \in 0, 1, \ldots, 25 = \mathbb{Z}_{26}$

  - Encryption: $y = e_k(x) \equiv ax + b \mod 26$
  - Decryption: $x = d_k(y) \equiv a^{-1}(y - b) \mod 26$
- Since the inverse of a is needed for inversion, we can only use values for a for which:

  $$gcd(a, 26) = 1.$$

  There are 12 values for a that fulfill this condition.
- From this follows that the key space is only $12 \times 26 = 312$ (cf. Sec 1.4 in Understanding Cryptography)
- It is not secure!
  - Exhaustive key search and letter frequency analysis can be used

Why Euclidean Algorithm is cool?

- Compute the greatest common divisor $gcd(a, b)$ of two integers a and b
- gcd is easy for small numbers:
  1. factor a and b
  2. gcd = highest common factor

  Example

  $a = 84 = 2 \cdot 2 \cdot 3 \cdot 7$, $b = 30 = 2 \cdot 3 \cdot 5$

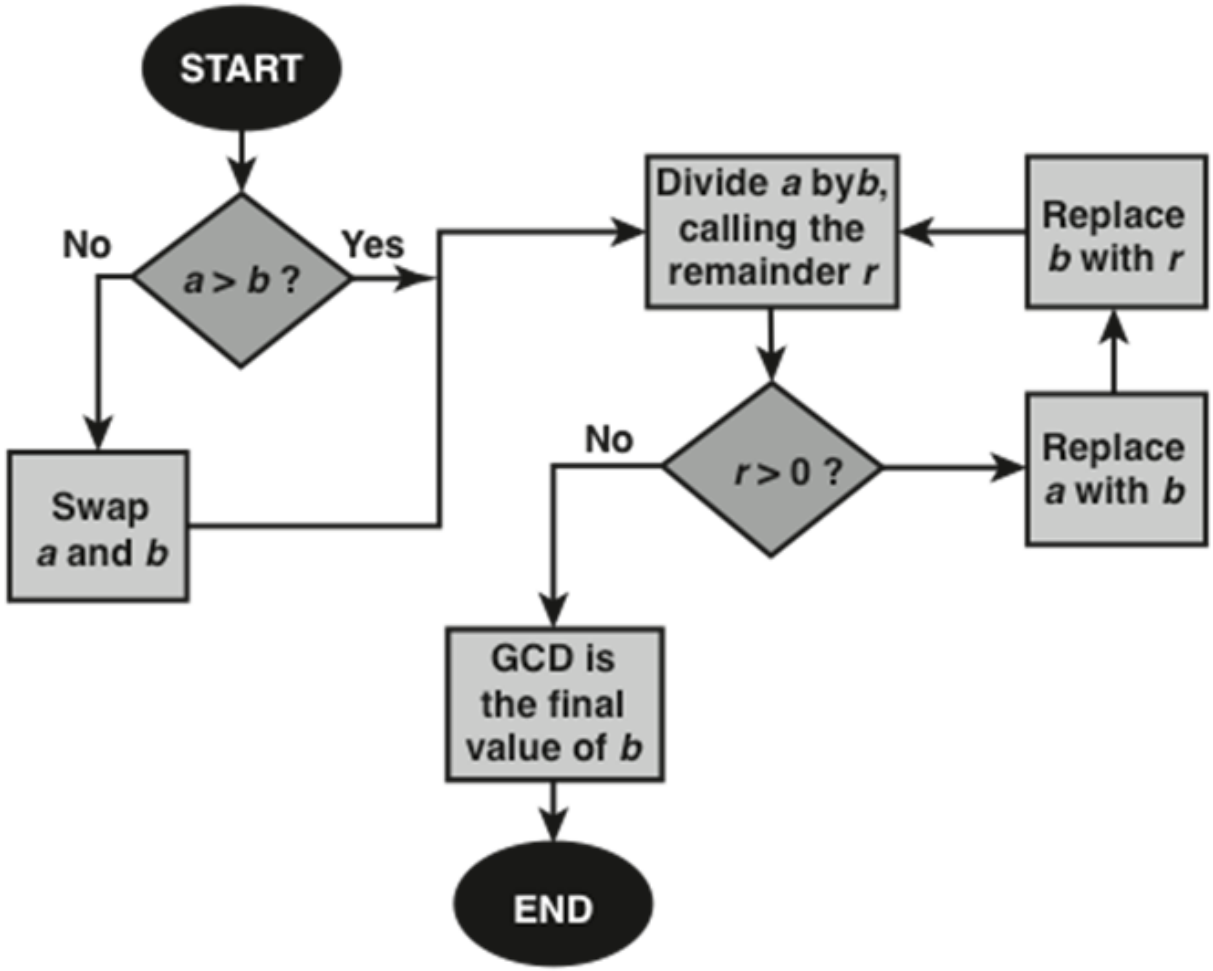
  The gcd is the product of all common prime factors:

  $2 \cdot 3 = 6 = gcd(30, 84)$
- **But**: Factoring (**without** Euclidean Algorithm) is complicated (and often infeasible) for large numbers.

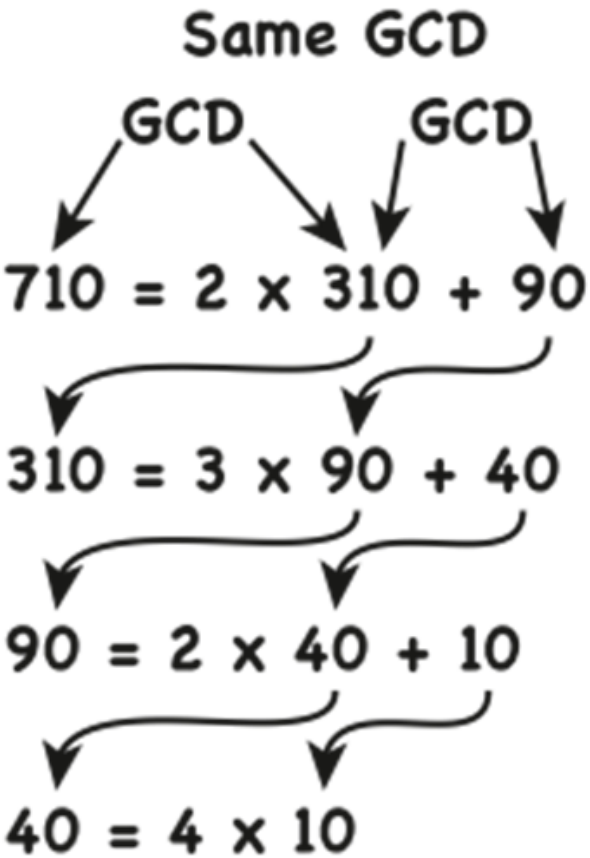
Why Euclidean Algorithm is cool?

- `Observation`: $gcd(a, b) = gcd(a - b, b)$
- `Core idea`:
  - Reduce the problem of finding the gcd of two given numbers to that of the *gcd* of two smaller numbers
  - Repeat process recursively
  - The final $gcd(b, 0) = b$ is the answer to the original problem !
- Very efficient method even for long numbers: the complexity grows `linearly` with the number of bits.

Euclidean Algorithm



Euclidean Algorithm

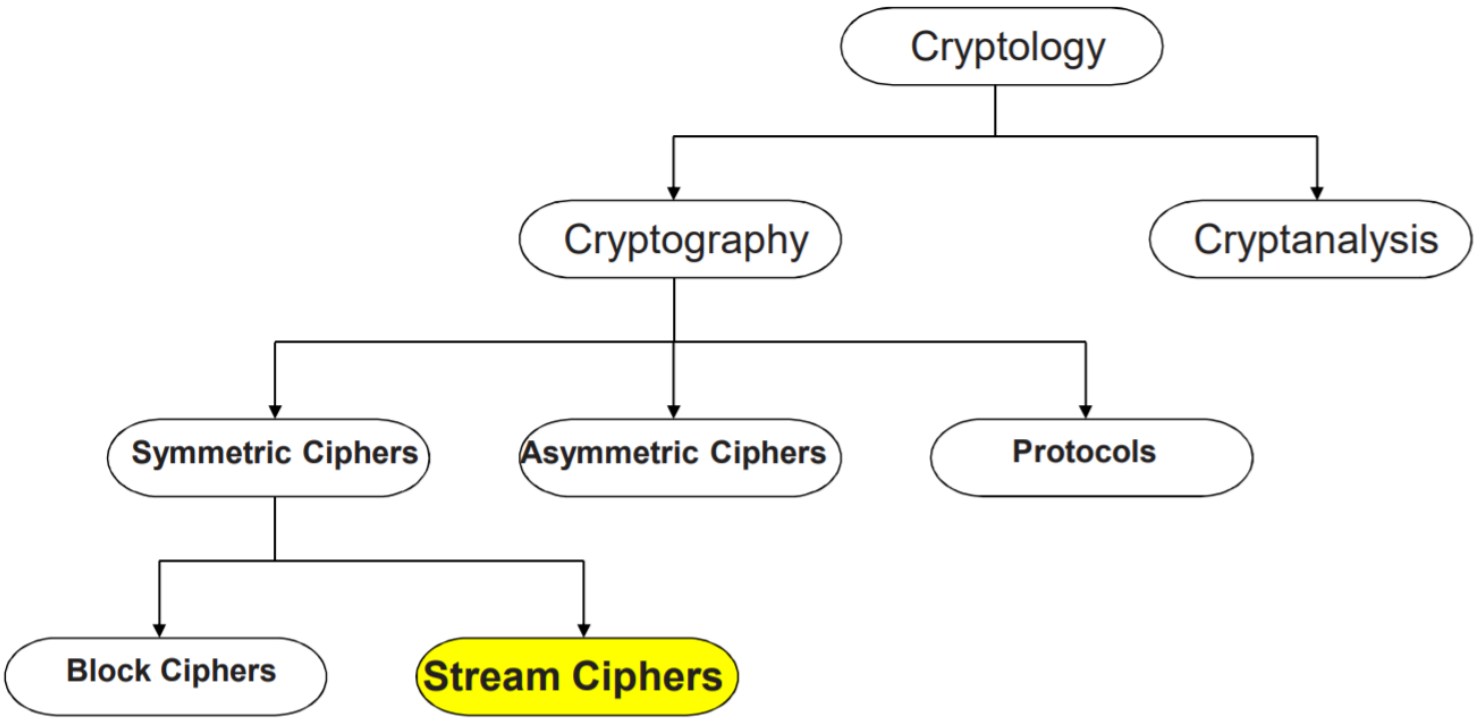


Euclidean Algorithm Example: gcd(710, 310)

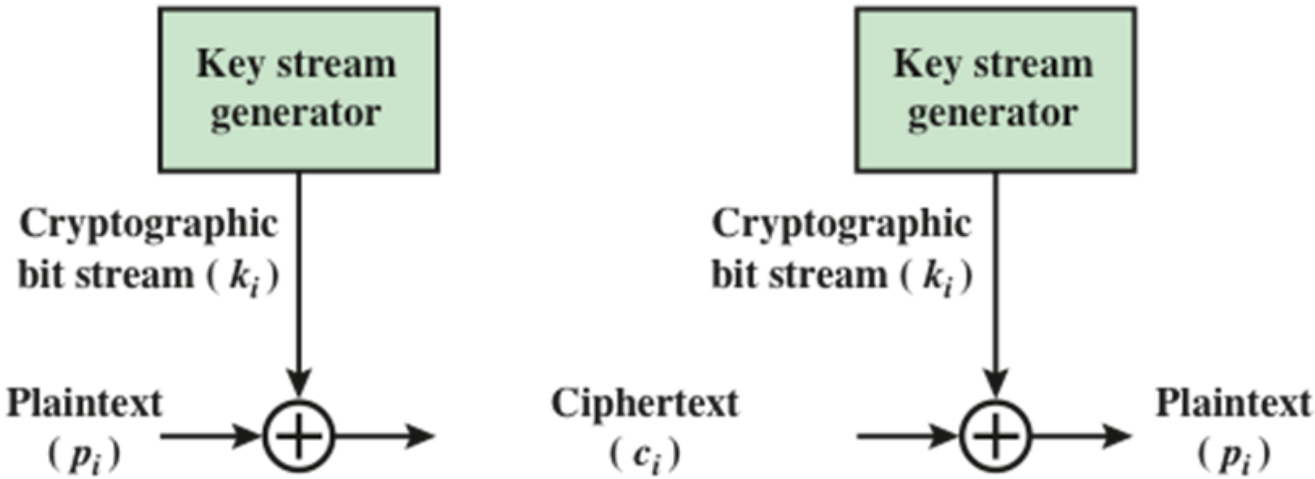
Hill Cipher

- Developed by the mathematician Lester Hill in 1929
- Strength is that it completely hides single-letter frequencies
  - The use of a larger matrix hides more frequency information
  - A 3×3 Hill cipher hides not only single-letter but also two-letter frequency information
- Strong against a ciphertext-only attack but easily broken with a known plaintext attack

Stream Ciphers in the Field of Cryptology

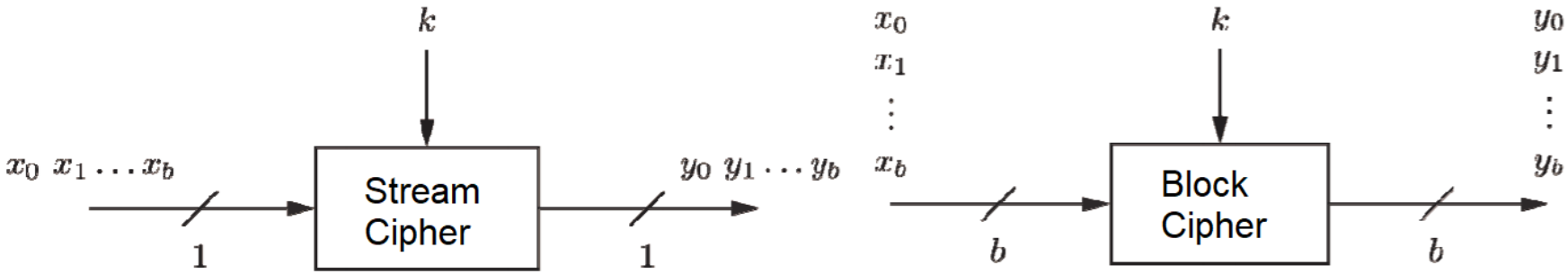


Stream Ciphers (Vernam Ciphers)



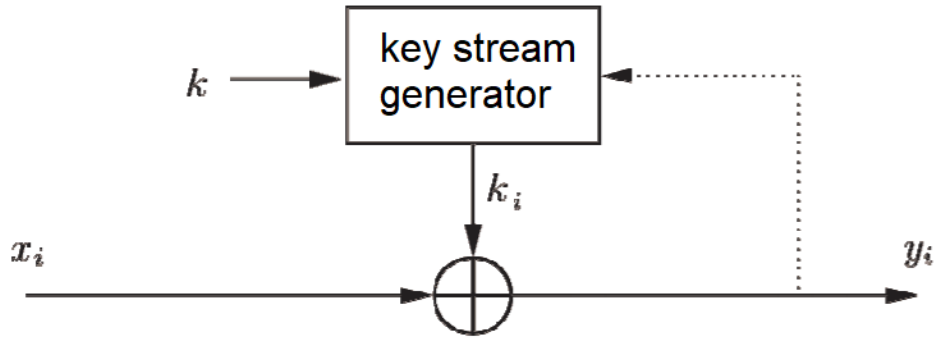
- Stream Ciphers were invented in 1917 by `Gilbert Vernam`
- Plaintext x, ciphertext y and key stream k consist of individual bits x_i , y_i , $k_i \in \{0, 1\}$
- Encryption and decryption are simple additions modulo 2 (XOR)
- Encryption and decryption are the same functions
  - `Encryption`: $y_i = e_{k_i}(x_i) = x_i + k_i \mod 2$
  - `Decryption`: $x_i = d_{k_i}(y_i) = y_i + k_i \mod 2$

Stream Cipher vs. Block Cipher



- Stream Ciphers
  - Encrypt bits individually
  - Usually small and fast → common in embedded devices (e.g., A5/1 for GSM phones)
- Block Ciphers
  - Always encrypt a full block (several bits)
  - Are common for Internet applications

Synchronous vs. Asynchronous Stream Cipher



- Security of stream cipher depends entirely on the key stream k_i:
  - Should be random, i.e., $Pr(k_i = 0) = Pr(k_i = 1) = 0.5$
  - Must be reproducible by sender and receiver
- Synchronous Stream Cipher
  - Key stream depend only on the key (and possibly an initialization vector IV)
- Asynchronous Stream Ciphers
  - Key stream depends also on the ciphertext (dotted feedback enabled)

Why is Modulo 2 Addition a Good Encryption Function?

- Modulo 2 addition is equivalent to XOR operation
- For perfectly random key stream k_i, each ciphertext output bit has a 50% chance to be 0 or 1
  Good statistic property for ciphertext
- `Inverting XOR` is simple, since it is the same XOR operation

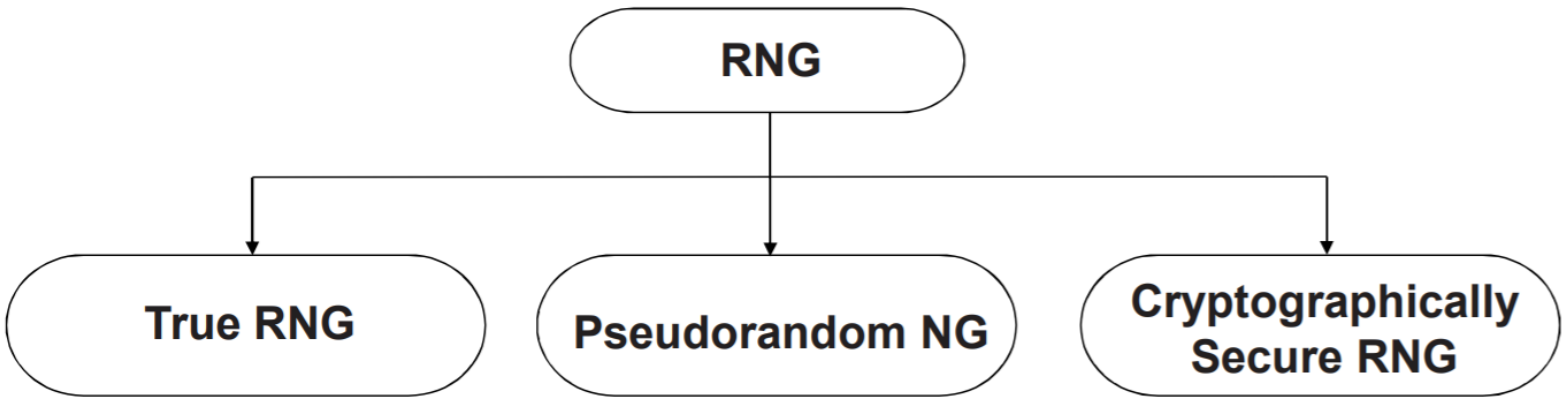
x_i	s_i	y_i
0	0	0
0	1	1
1	0	1
1	1	0

Stream Cipher: Throughput

Performance comparison of symmetric ciphers (Pentium4):

Cipher	Key length	Mbit/s
DES	56	36.95
3DES	112	13.32
AES	128	51.19
RC4 (stream cipher)	(choosable)	211.34

Random number generators (RNGs)



Needed in cryptography, in particular for stream ciphers

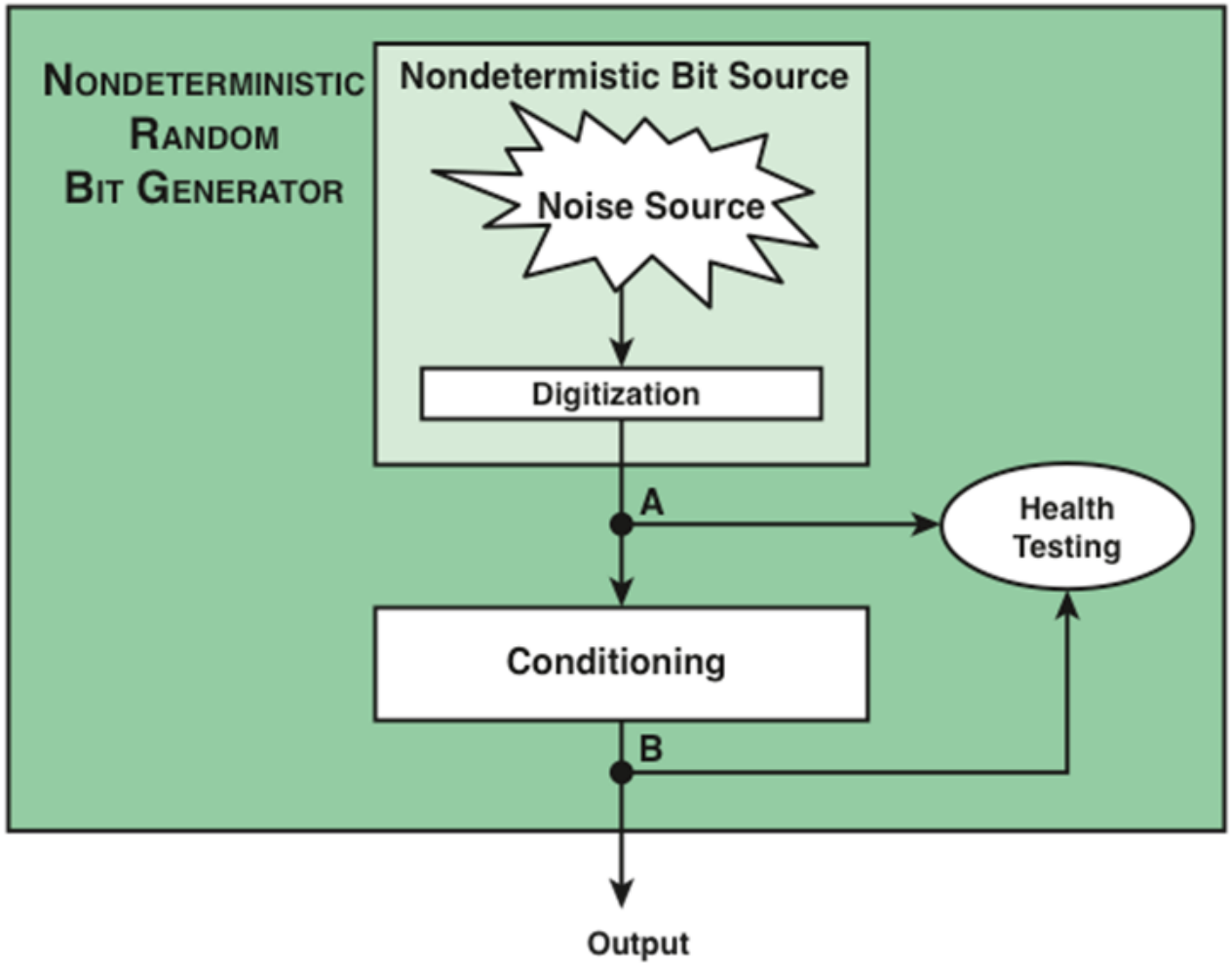
Randomness

- Basic requirements for randomness:
  - Uniform distribution (the frequency of occurrence of ones and zeros should be approximately equal)
  - Independence (each bit uncorrelated with all previous)
- For cryptography, we also need:
  - Compromise of one output must not compromise future or previous output
  - Different output for each use

True Random Number Generators (TRNGs)

- Based on physical random processes: coin flipping, dice rolling, semiconductor noise, radioactive decay, mouse movement, clock jitter of digital circuits
- Output stream k_i should have good statistical properties: $Pr(k_i = 0) = Pr(k_i = 1) = 0.5$ (often achieved by post-processing)
- Output can neither be predicted nor be reproduced
- Typically used for generation of keys, nonces (used only-once values) and for many other purposes

True Random Number Generators (TRNGs)



Possible Sources of Randomness

The following possible sources of randomness can be used on a computer to generate true random sequences:

- Sound/video input
- Disk drives

True Random Number Generators (TRNGs)

"Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin."

John von Neumann

Pseudorandom Number Generator (PRNG)

An algorithm that is used to produce an open-ended sequence of bits

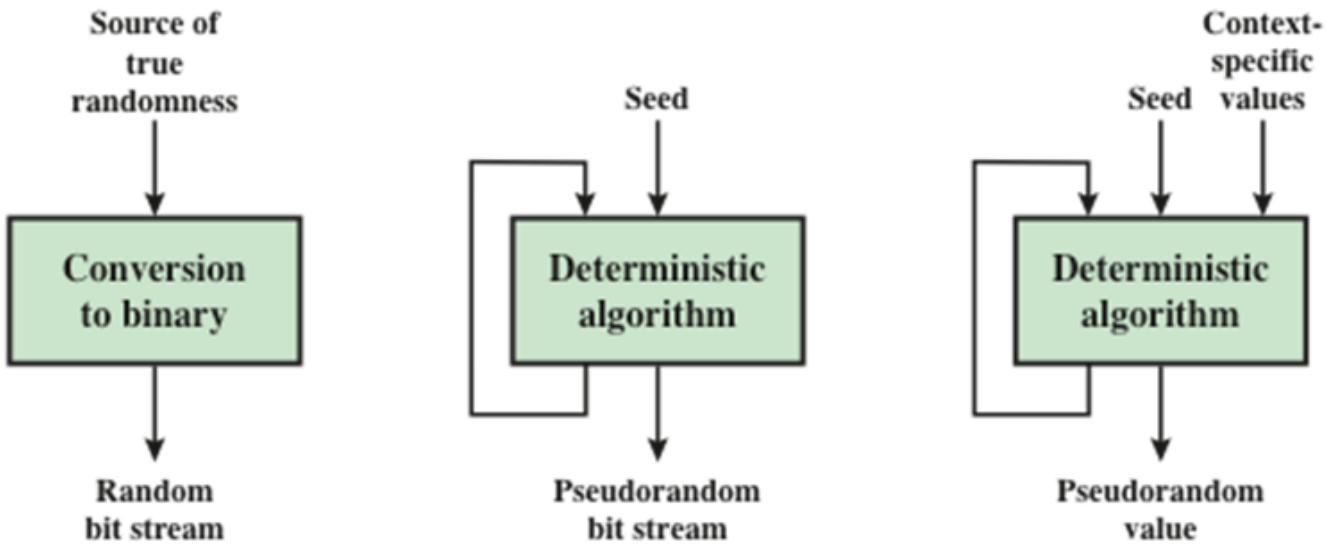
- Generate sequences from initial seed value
- Typically, output stream has good statistical properties
- Output can be reproduced and can be predicted
  Often computed in a recursive way:

$$k_0 = seed$$

$$k_{i+1} = f(k_i, k_{i-1}, \ldots, k_{i-t})$$

Most PRNGs have bad cryptographic properties!

TRNG vs. two forms of PRNG's



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

Other than the number of bits produced there is no difference between a PRNG and a PRF

PRNG 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 (the generated bit stream needs to appear random even though it is deterministic)
  - NIST SP 800-22 Software, the diehard tests developed by George Marsaglia
- Unpredictability (a random sequence will have no correlation with the seed)
- Characteristics of the seed

Random number generators

PRNG Examples: Linear Congruential Generator

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

$$a, b, m, seed$$

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

$$x_{n+1} = (ax_n + b) \mod m$$

- `Bad cryptographic properties due to the linearity of most PRNGs`
- The selection of values for a, b, and m is critical in developing a good random number generator

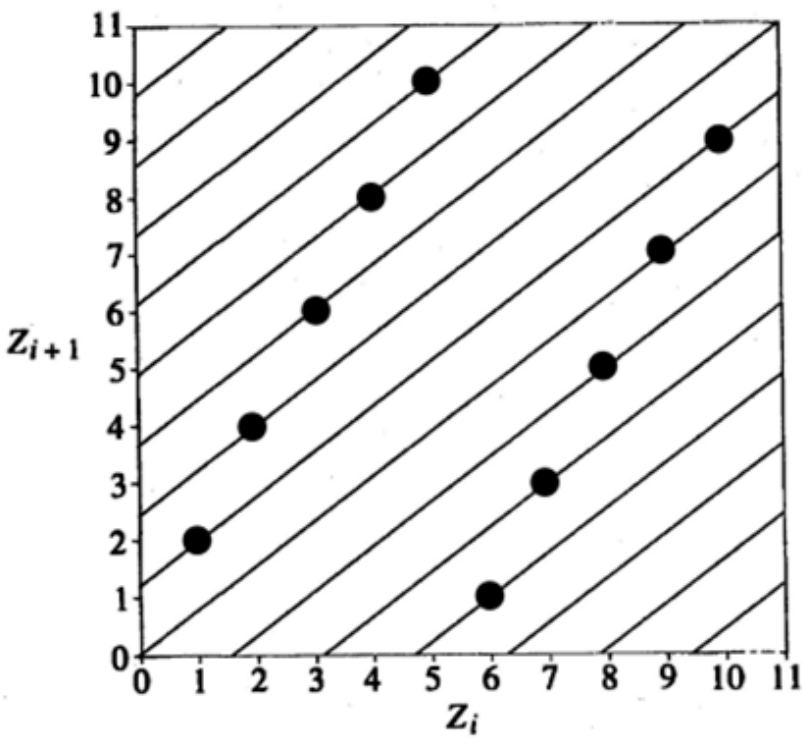
PRNG Examples: Linear Congruential Generator

$$X_i = aX_{i-1} \ mod \ 11, \quad X_0 = 1$$

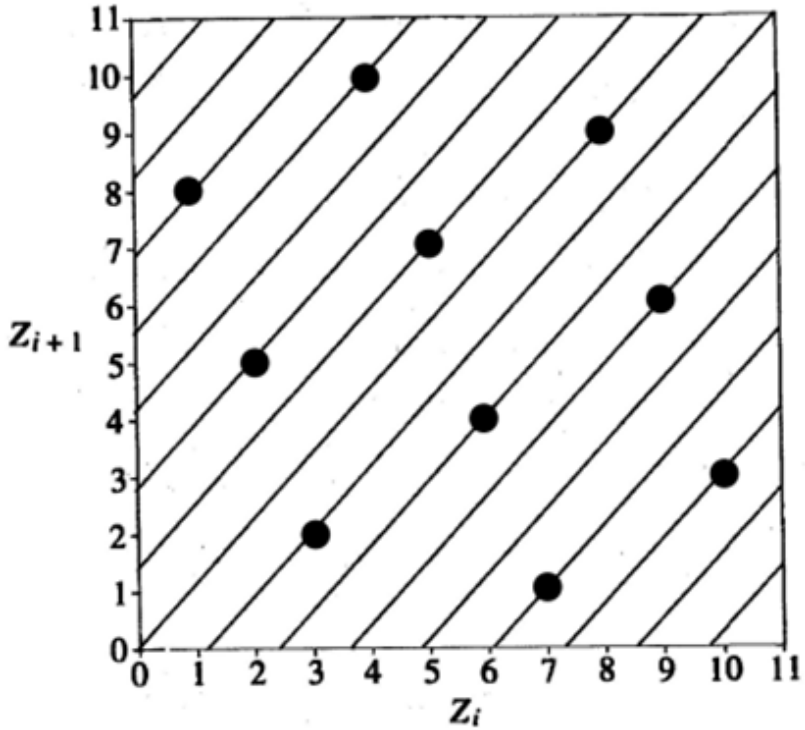
					X_i					
	a=1	2	3	4	5	6	7	8	9	10
i=0	1	**1**	1	1	1	**1**	**1**	**1**	1	1
1	1	**2**	3	4	5	**6**	**7**	**8**	9	10
2		**4**	9	5	3	**3**	**5**	**9**	4	1
3		**8**	5	9	4	**7**	**2**	**6**	3	
4		**5**	4	3	9	**9**	**3**	**4**	5	
5		**10**	1	1	1	**10**	**10**	**10**	1	
6		**9**				**5**	**4**	**3**		
7		**7**				**8**	**6**	**2**		
8		**3**				**4**	**9**	**5**		
9		**6**				**2**	**8**	**7**		
10		**1**				**1**	**1**	**1**		

PRNG Examples: Linear Congruential Generator

$$X_i = aX_{i-1} \ mod \ 11$$



a) X_0=1, a=2

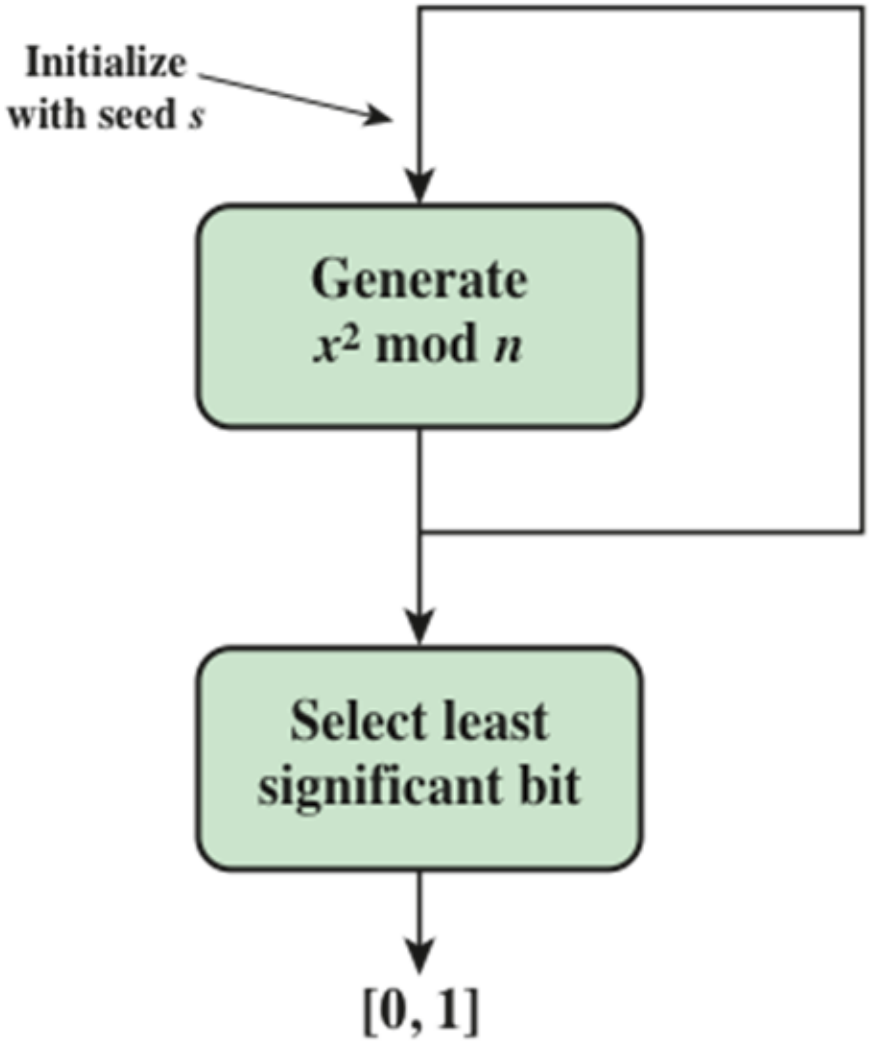


b) X_0=1, a=8

PRNG Examples: 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

PRNG Examples: Blum Blum Shub (BBS) Generator



PRNG Examples: Blum Blum Shub (BBS) Generator

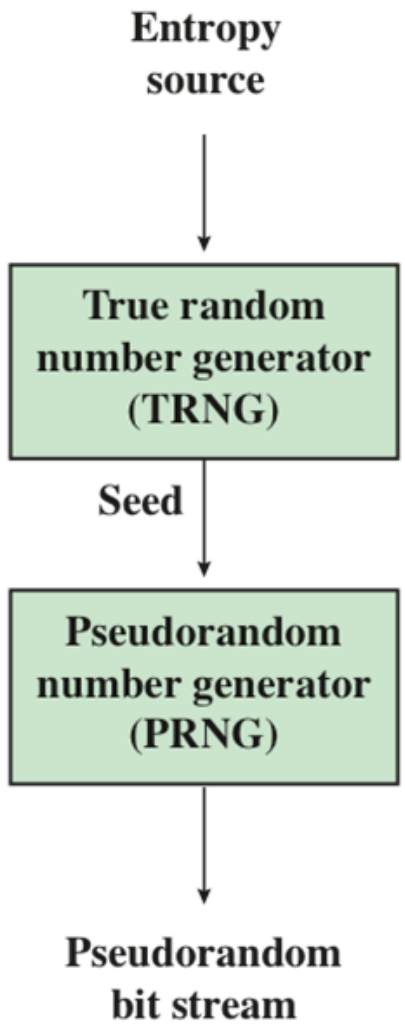
i	X_i	B_i
0	20749	
1	143135	1
2	177671	1
3	97048	0
4	89992	0
5	174051	1
6	80649	1
7	45663	1
8	69442	0
9	186894	0
10	177046	0

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

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

Typically, the seed is generated by a TRNG



Comparison of PRNGs and TRNGs

	PRNG	TRNG
Efficiency	Very Efficient	Generally inefficient
Determinism	Deterministic	Nondeterministic
Periodicity	Periodic	Aperiodic

One-Time Pad

One-Time Pad

- Improvement to Vernam cipher proposed by an Army Signal Corp officer, Joseph Mauborgne
- Use a random key that is as long as the message so that the key need not be repeated
- Key is used to encrypt and decrypt a single message and then is discarded
- Each new message requires a new key of the same length as the new message
- Scheme is `unbreakable`
  - Produces random output that bears no statistical relationship to the plaintext
  - Because the ciphertext contains no information whatsoever about the plaintext, there is simply no way to break the code

One-Time Pad – Difficulties

- The one-time pad offers complete security but, in practice, has two fundamental difficulties:
  - There is the practical problem of making large quantities of random keys
    Any heavily used system might require millions of random characters on a regular basis
  - Key distribution problem
    For every message to be sent, a key of equal length is needed by both sender and receiver
- Because of these difficulties, the one-time pad is of limited utility
  - Useful primarily for low-bandwidth channels requiring very high security
- The one-time pad is the `only cryptosystem` that exhibits `perfect secrecy`

Thanks for Your attention.