CE4024 Lecture Notes 1

# Security Goals/Objectives

### CIA triad

* **Confidentiality** – preserve restrictions on information access and release
* **Integrity** – guard against information being modified or destroyed
* **Availability** – ensure reliable access to/use of data

### Parkerian hexad

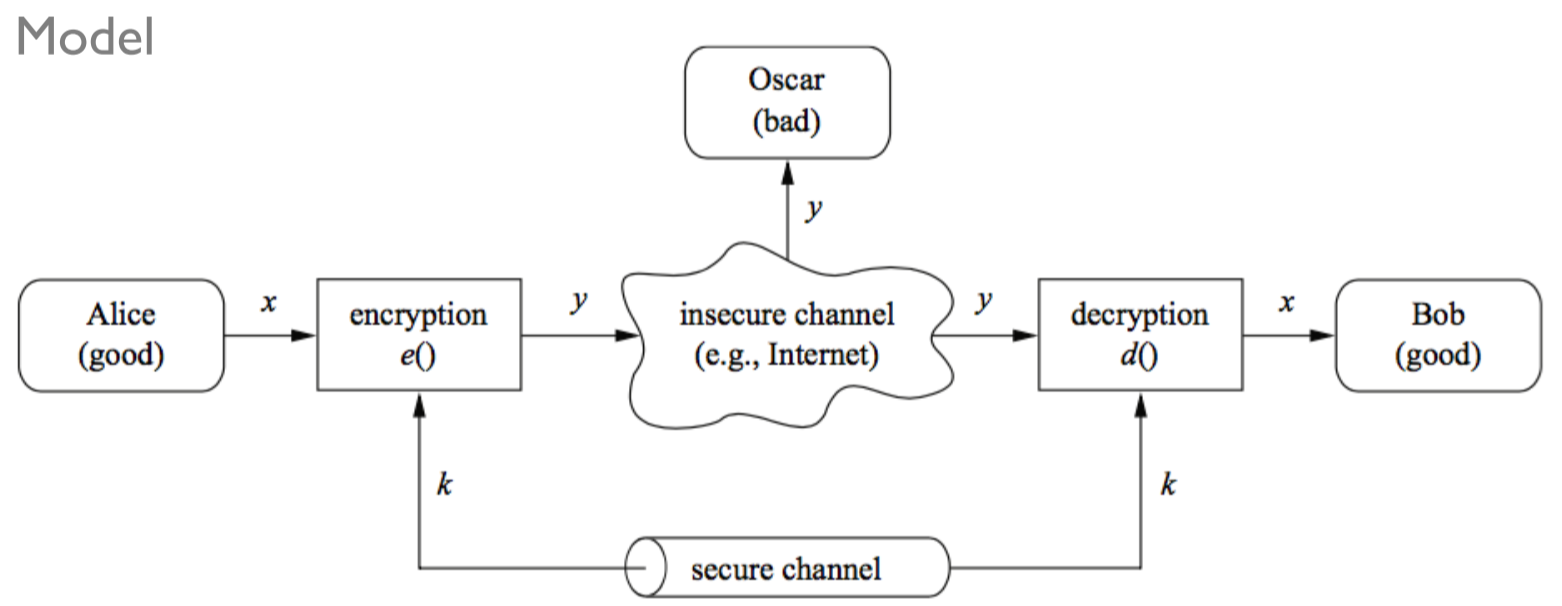
* **CIA**
* **Possession** – control over data that is lost/not in your hands
* **Authenticity** – ensure data comes from verified source
* **Utility** – ability to decrypt data once it’s received

## Types of Attacks

* **Passive** – attacker doesn’t modify network, just listens in on transmitted data
  + Interception
  + Traffic analysis
* **Active** – attacker modifies network/information
  + Impersonation – attacker impersonates another person to send information
  + Replay – attacker sends a previous message again
  + Modification – attacker modifies a message from the original sender
  + DDoS – distributed denial of service

# Private Communication

## Symmetric/Secret Key Cryptography



* Sender and receiver share a common secret key
* Encryption/decryption is done using the same key

# Substitution Ciphers

* Each letter of the alphabet is substituted with another letter to form the cipher

### k-Shift Cipher

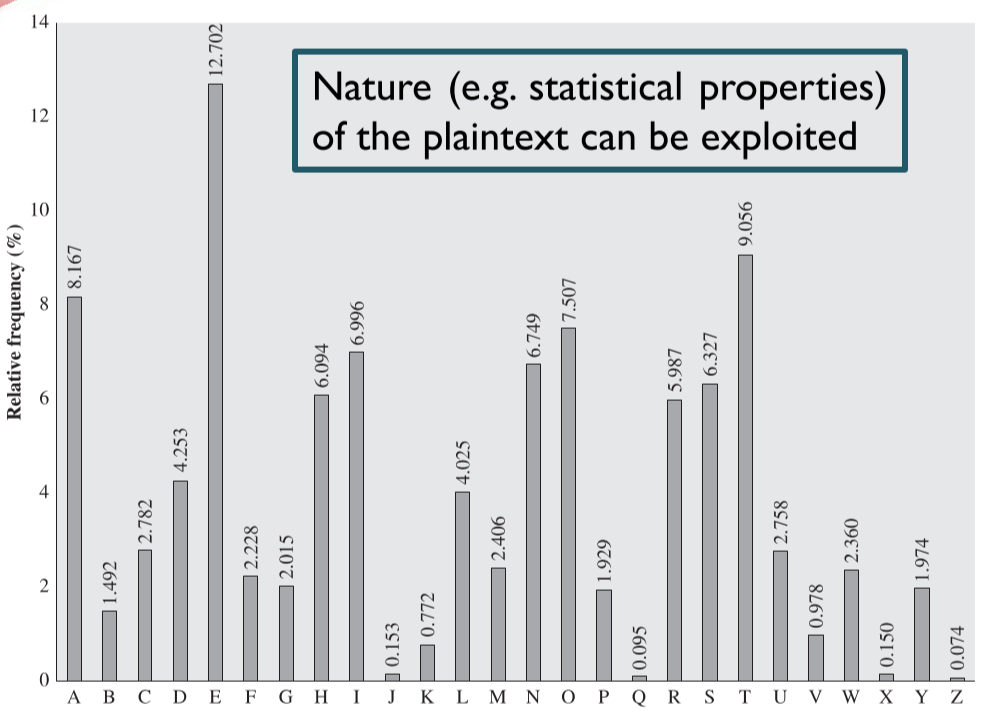
* Each letter is assigned a number and a secret key is selected
* Encryption algorithm:
  + = cipher text
  + = encryption algorithm
  + = secret key
  + = plain text
* Decryption algorithm:
  + = decryption algorithm
* This cipher is easy to determine based on a **brute-force attack** that exhausts all possibilities

## Kerchhoff’s Principle

* A system should be secure even if the attacker knows the encryption and decryption algorithms (but not the secret key)

## Monoalphabetic Ciphers

* Each plaintext symbol is replaced with a unique ciphertext symbol
* If a **random permutation** of substitution symbols is used, then it is a lot harder to determine by brute-force attack
  + However, **frequency analysis** can be done on the cipher text to help determine plain text based on statistical frequencies of letters



### Playfair Ciphers

* **n-gram**: contiguous sequence of symbols from text
* Use **multi-letter encryption** to reduce structural information about ciphertext
  + E.g. aq 🡪 DM, av 🡪 GR, vq 🡪 XM
  + Each digram is considered a single plaintext symbol, so this is still a monoalphabetic cipher

**Initialization**

1. Select a secret key: e.g. CRYPTOCRYO
2. Populate 5x5 matrix with secret key (left to right, top to bottom), omitting duplicate letters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| C | R | Y | P | T |
| O |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

1. Add unused letters to matrix, I/J being equivalent

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| C | R | Y | P | T |
| O | A | B | D | E |
| F | G | H | I/J | K |
| L | M | N | Q | S |
| U | V | W | X | Z |

**Encryption**

1. Preprocess the plaintext by replacing repeating letter pairs with a filler letter (e.g. y)

E.g. yummy 🡪 yu my my

E.g. google 🡪 go og le

1. To get ciphertext, replace letters based on matrix:
   1. If letter pair falls in the **same row**, replace with letter on the right (wrapping)
   2. If letter pair falls in the **same column**, replace with letter beneath (wrapping)
   3. Otherwise, replace plaintext letter with letter in **same row**, but **column of paired letter**

Example:

* Plaintext: cool dude
* Preprocess: co ol du de
* Ciphertext: OF FU OX EO
* Different letters map to the same plaintext letter
  + There are 262 unique digrams
  + The frequency of individual letters doesn’t represent their frequency in the plaintext, protecting against frequency analysis

## Polyalphabetic Substitution

* Use a set of monoalphabetic ciphers, with a key that determines which one to use at each step

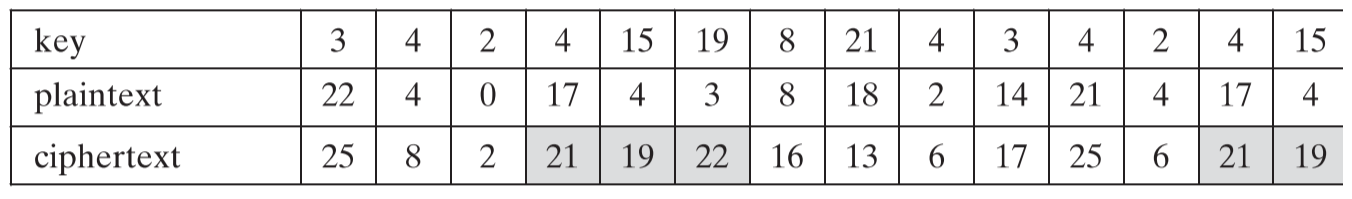
### Vigenère Cipher

A keyword is used to determine which letters the plaintext is encrypted to; choose based on the keyword index mod its length

* **Plaintext**:
* **Keyword**:
* **Encryption:**
* **Decryption**:

Example:

* **Plaintext**: wearediscoveredsaveyourself
* **Keyword**: deceptivedeceptivedeceptive



* **Ciphertext**: ZICVTWQNGRZGVTWAVZHCQYGLMGJ
  + Multiple substitutions for the same letter
  + However, there are periodic repetitions
  + If the attacker figures out the length of the keyword, they can attack individual monoalphabetic ciphers

### Vernam Cipher (One-Time Pad)

* If the keyword is as long as the plaintext and truly random, then the same letter substitution is never systematically repeated – the ciphertext has no relation to the plaintext
* This is mathematically proven to be impossible to break without knowing the key

# Transposition

* Rearrange the letters in the plaintext in some sort of permutation
* **Rail fence technique**: even letters first, then odd letters

### With a Key

Example: Construct ciphertext by column, using a key

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Key** | 4 | 3 | 1 | 2 | 5 | 6 | 7 |
| **Plaintext** | a | t | t | a | c | k | p |
| o | s | t | p | o | n | e |
| d | u | n | t | i | l | t |
| w | o | a | m | x | y | z |

**Ciphertext**: TTNAAPTMTSUOAODWCOIXKNLYPETZ

* Letter frequencies remain the same as plaintext, making it easier to recognize
  + Di/trigrams can help in guessing matrix dimensions and column permutations
* **Reapplication** can make it harder to guess the matrix dimensions/column permutations

Example: Reapply same permutation to first ciphertext

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Key** | 4 | 3 | 1 | 2 | 5 | 6 | 7 |
| **Plaintext** | t | t | n | a | a | p | t |
| m | t | s | u | o | a | o |
| d | w | c | o | i | x | k |
| n | l | y | p | e | t | z |

**Ciphertext**: NSCYAUOPTTWLTMDNAOIEPAXTTOKZ

# Three Cryptographic Ideas

* **Substitution**: replace plaintext symbols with other ones
  + Polyalphabetic substitution is stronger as it resists frequency analysis
* **Transposition**: reorder/permute sequence of symbols
* **Cascade**: (re)-apply multiple smaller units of encryption to get a stronger encryption

## Rotor Machines

* One rotor has polyalphabetic substitution of period 26 – 26 different monoalphabetic ciphers
* Multiple rotors:
  + E.g. with 3 rotors, 263 = 17 576 different monoalphabetic ciphers

# Steganography

* The hiding of messages through embedding them within other messages
* Replaces bits in useless or unused data in computer files with invisible information

# Cryptoanalysis Models

### Cipher Attack

* Cryptanalyst knows **encryption algorithm and ciphertext**

### Known Plaintext

* Cryptanalyst knows encryption algorithm and ciphertext, + **one or more plaintext ciphers** generated with the secret key

### Chosen Plaintext

* Cryptanalyst knows encryption algorithm and ciphertext, + a **cryptanalyst-chosen plaintext** and corresponding ciphertext generated with the secret key

### Chosen Ciphertext

* Cryptanalyst knows encryption algorithm and ciphertext, + a **cryptanalyst-chosen ciphertext message** and corresponding decrypted plaintext generated with the secret key

### Chosen Text

* Cryptanalyst knows encryption algorithm and ciphertext, + **chosen plaintext + chosen ciphertext** examples

# Number Theory

## Divisibility

Given two positive integers and ,

* is the quotient
* is the remainder
* and are unique
* If , then we say divides :

## Prime Numbers

Positive integers , whose only divisors are and

## Greatest Common Divisor (GCD)

The greatest common divisor of two integers is the largest integer that divides both and

* If , then:
  + is a divisor of and
  + All divisors of and are divisors of
* Two integers are **relatively prime** if their

### Euclidean Algorithm for Computing GCD

The GCD of two integers and , where , is equal to the gcd of and :

* Keep going until , then you’ve found the GCD

Example: Find

x

## Bezout’s Identity

The GCD of two integers and is equal to the sum of multiples of and :

* Since , and must have opposite signs

### Extended Euclidian Algorithm for Computing Bezout’s Identity Coefficients

By reversing the steps of the Euclidean algorithm, we can determine the coefficients in Bezout’s identity:

* Replace smaller remainder at each step with previous equation

Example: Find

Working backwards:

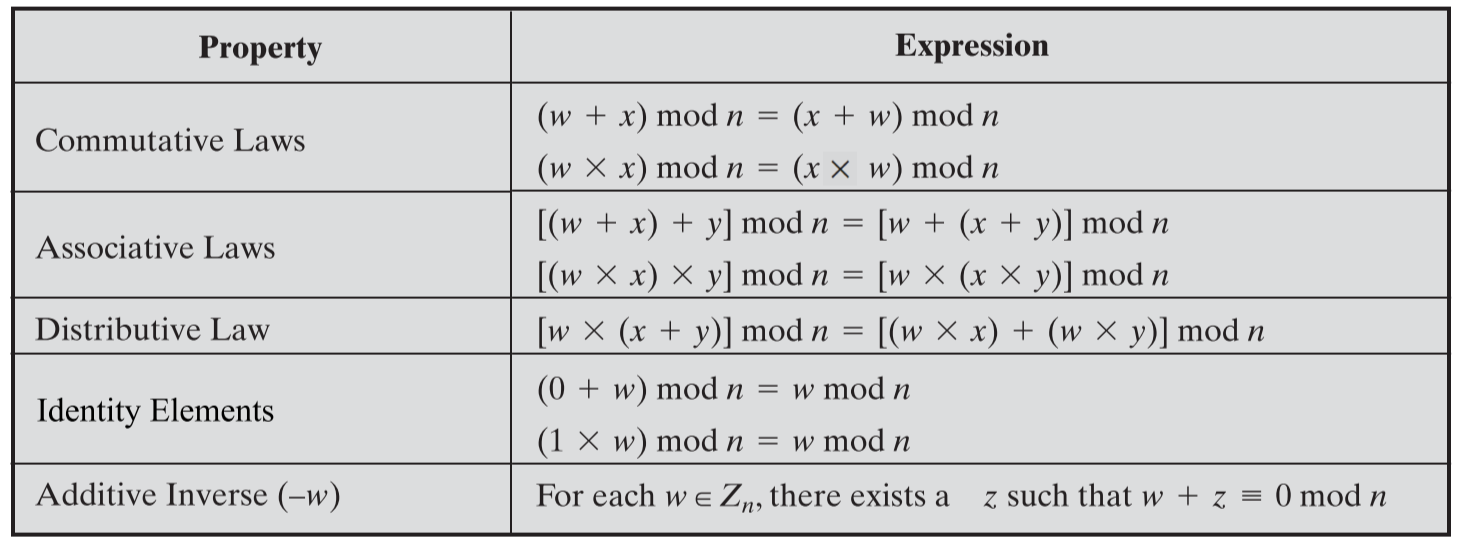
Thus,

# Modular Arithmetic

represents the set of integers

* **Equivalence classes**: for any ,

### Modular Arithmetic Properties



The **additive inverse** of , denoted as , is a number such that

The **multiplicative inverse** of , denoted as , is a number such that

* Only numbers that are relatively prime to have a multiplicative inverse, hence not all members of have a multiplicative inverse

Example: (mod 8)

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 0 | 0 | -- |
| 1 | 7 | 1 |
| 2 | 6 | -- |
| 3 | 5 | 3 |
| 4 | 4 | -- |
| 5 | 3 | 5 |
| 6 | 2 | -- |
| 7 | 1 | 7 |

# Groups, Rings, and Fields

## Groups

A group, denoted , is a set of elements and an operator , that satisfies the following properties:

1. **Closure**: if and are in , then is in
2. **Associativity**: for all elements in
3. **Identity** **element**: there is an element such that in
4. **Inverse** **element**: there is an element for every in , such that

The **order** of the group is the number of elements (finite or infinite) in the group

An **Abelian group** also satisfies:

1. **Commutativity**: for all elements in G

A **cyclic group** is a group where:

* There exists an element in , such that **every element in is a power of**  –
* is the **generator** of the group, and power refers to repeated application of the group’s operator

## Rings

A ring, denoted , is a set of elements , an addition operator , and a multiplication operator , that satisfies the following properties:

1. is an **abelian group** with respect to addition: 0 is the identity element, and the inverse of an element is
2. **Multiplicative closure**: if and are in , then is in
3. **Multiplicative associativity**: for all elements in
4. **Distributive laws**: and for all elements in

A **commutative ring** also satisfies:

1. **Multiplicative commutativity**: for all elements in

An **integral domain** is a commutative ring that also satisfies:

1. **Multiplicative identity element**: there is an element in such that for all elements in
2. **No zero divisors**: if in and , then either or

* doesn’t satisfy this property, since , but neither nor

## Fields

A field, denoted , is a set of elements , an addition operator , and a multiplication operator , that satisfies the following properties:

1. is an **integral domain**
2. **Multiplicative inverse**: for each in except 0, there is an element in such that

* Integers do not satisfy this property, but real numbers do

# Galois (Finite) Fields

A Galois field, denoted , is a finite field of order

is a set of integers with modulo operations, where is a prime number

* is a multiplicative ring
* Since is prime, all non-zero elements in are relatively prime, and hence have a multiplicative inverse - this makes a finite field

Example:

*Addition*

|  |  |  |
| --- | --- | --- |
| **+** | **0** | **1** |
| **0** | 0 | 1 |
| **1** | 1 | 0 |

*Multiplication*

|  |  |  |
| --- | --- | --- |
| **x** | **0** | **1** |
| **0** | 0 | 0 |
| **1** | 0 | 1 |

*Additive and Multiplicative Inverses*

|  |  |  |
| --- | --- | --- |
|  |  |  |
| **0** | 0 | -- |
| **1** | 1 | 1 |

Example:

*Addition*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **+** | **0** | **1** | **2** | **3** | **4** | **5** | **6** |
| **0** | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| **1** | 1 | 2 | 3 | 4 | 5 | 6 | 0 |
| **2** | 2 | 3 | 4 | 5 | 6 | 0 | 1 |
| **3** | 3 | 4 | 5 | 6 | 0 | 1 | 2 |
| **4** | 4 | 5 | 6 | 0 | 1 | 2 | 3 |
| **5** | 5 | 6 | 0 | 1 | 2 | 3 | 4 |
| **6** | 6 | 0 | 1 | 2 | 3 | 4 | 5 |

*Multiplication*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **+** | **0** | **1** | **2** | **3** | **4** | **5** | **6** |
| **0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **1** | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| **2** | 0 | 2 | 4 | 6 | 1 | 3 | 5 |
| **3** | 0 | 3 | 6 | 2 | 5 | 1 | 4 |
| **4** | 0 | 4 | 1 | 5 | 2 | 6 | 3 |
| **5** | 0 | 5 | 3 | 1 | 6 | 4 | 2 |
| **6** | 0 | 6 | 5 | 4 | 3 | 2 | 1 |

*Additive and Multiplicative Inverses*

|  |  |  |
| --- | --- | --- |
|  |  |  |
| **0** | 0 | -- |
| **1** | 6 | 1 |
| **2** | 5 | 4 |
| **3** | 4 | 5 |
| **4** | 3 | 2 |
| **5** | 2 | 3 |
| **6** | 1 | 6 |

## Finding Multiplicative Inverses

**Bezout’s Identity**:

If is a prime number , then and

If and are relatively prime, then   
Since , then , which means that

To find , use the **extended Euclidean algorithm**:

Example: Find

Thus, the multiplicative inverse is

# Polynomial Arithmetic

Consider a polynomial of degree , with integer coefficients defined over a set :

Then the **sum/difference** of two polynomials (possibly of different degrees) is:

The **multiplication** of two polynomials (possibly of different degrees) is:

The **division** of two polynomials (possibly of different degrees) is not always possible:

isn’t possible because isn’t an integer

## Division of Polynomials over a Field

* **Polynomial rings** are a set of polynomials whose coefficients are elements of a field
  + Division is defined, and their **multiplicative inverse** exists
* Given polynomials and of degree and , with : 
  + The degree of is
  + The degree of is smaller than

### Irreducible Polynomials

A “prime polynomial” – a polynomial over a field that **cannot** be expressed as a product of polynomials of lower degrees

* For a polynomial of degree , test polynomials **up to degree**

Example: is irreducible over ?

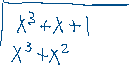
means that the coefficients can be either 0 or 1; so, candidate factors are



Therefore isn’t irreducible over .

Example: is irreducible over ?

Candidate factors are

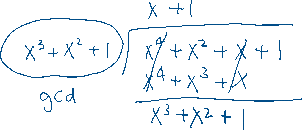
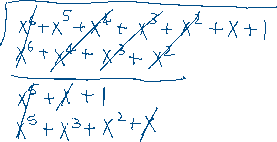


Therefore is irreducible over .

## GCD of Polynomial Rings

* Analogous to the definition of GCD for integers
* **Euclidean algorithm (extended)** and **Bezout’s identity** are both applicable

Example: find ], given and



## Modular Polynomial Arithmetic

Consider the set of polynomials of **degree or less,** **over** : the set is a finite field

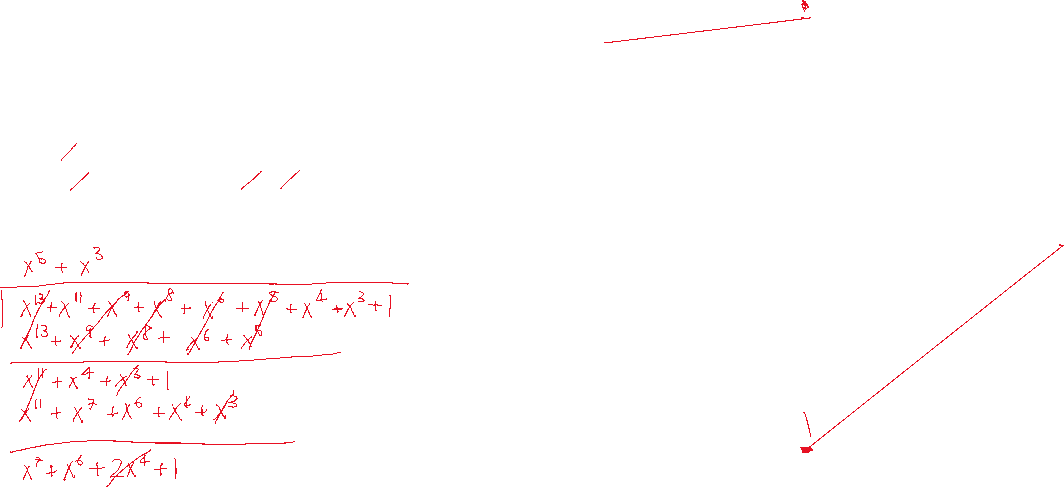
* The arithmetic of coefficients is **modulo**
* If multiplication yields a polynomial of degree , then it should be reduced **modulo an irreducible polynomial of degree**

Example: polynomials over , irreducible polynomial

Find and , given

means , which means coefficients in and degree

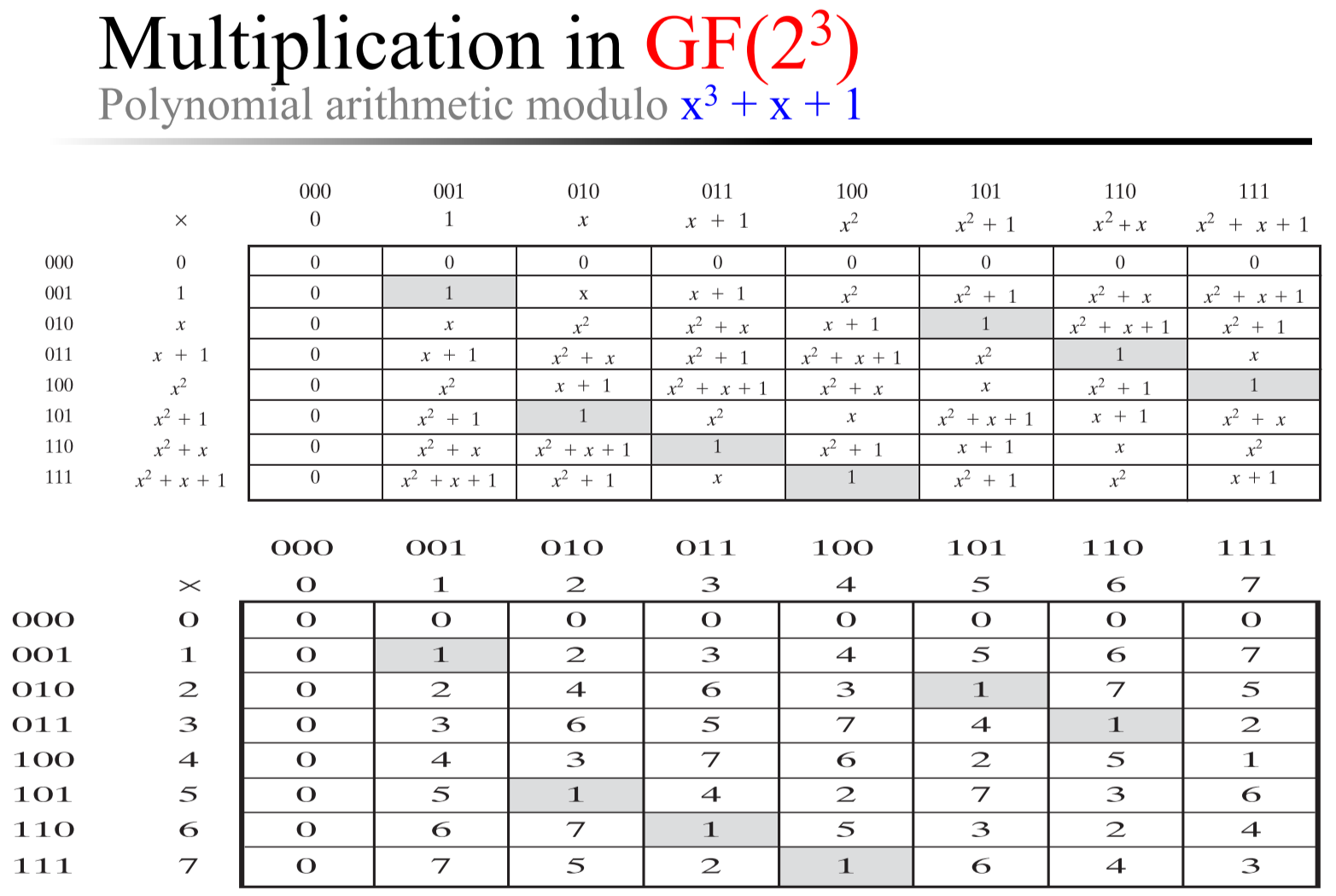
Need to reduce modulo :



Therefore



## GF(2n)



* Polynomials modulo an irreducible nth-degree polynomial form a **finite field**
* The **generator** of a finite field of order is an element whose powers generate all non-zero elements of
* The root of a polynomial is an element where
  + If the root of an irreducible polynomial is a generator of the finite field, then it is a **primitive root**

Example: Consider the root of

Compute .

# Prime Numbers



Any positive integer can be expressed as a product of powers of prime numbers

– the set of prime numbers

Given two integers and

Their product for every

Example:

**Divisibility**: If , then for every

**GCD**: If then for every

Example: GCD of and

## Fermat’s Little Theorem

For a prime number and a positive integer that is not divisible by :

**Proof**:

For values , the following characteristics hold:

* No value = 0, since isn’t divisible by , and values between and aren’t divisible by
* All values are between and
* All values are distinct, since when , so there are values

Hence,

## Euler’s Theorem

For any integers and that are relatively prime:

Where is **Euler’s totient function**, the number of positive integers that are less than and relatively prime with

* for prime number
* if are relatively prime
* for prime number

Hence, if a number is factored as

Then,

# Miller-Rabin Primality Test

**Property 1:**

Given a positive integer and a prime number :

iff:

* or

**Proof**:

If , then

**Property 2:**

If is a positive integer, and is prime, then

for some integer and odd integer

Then, either:

* There exists a positive integer where

**Proof**:

by Fermat’s Little Theorem

* If , then (since )
* If , then – continue squaring, and eventually will get

### Main idea:

If a candidate number is prime, then property 2 will hold

* Hence, if property 2 doesn’t hold, is not prime
  + Then, for any random , the test output is inconclusive with **probability 0.25**
  + Hence, repeat the test with multiple values of
* If it does hold, test is inconclusive

### Algorithm:

1. Find integers , odd , such that
2. Select a random integer , such that
3. If , return “inconclusive”
4. For to :
   1. If return “inconclusive”
5. Return “composite”

Example:

1. Choose
2. For to :
3. Return “composite”

Example:

1. Choose
2. For to :
   1. return “inconclusive”

# Primitive Roots

Recall **Euler’s Theorem**:

For any relatively prime integers and ,

Consider a more general statement:

There exist integers such that

* The smallest value of to satisfy is the **order of**
* If the order of an element is , then is a **primitive root of**

Example:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **1** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| **2** | 4 | 8 | 16 | 13 | 7 | 14 | 9 | 18 | 17 | 15 | 11 | 3 | 6 | 12 | 5 | 10 | 1 |
| **3** | 9 | 8 | 5 | 15 | 7 | 2 | 6 | 18 | 16 | 10 | 11 | 14 | 4 | 12 | 17 | 13 | 1 |
| **4** | 16 | 7 | 9 | 17 | 11 | 6 | 5 | 1 | 4 | 16 | 7 | 9 | 17 | 11 | 6 | 5 | 1 |
| **5** | 6 | 11 | 17 | 9 | 7 | 16 | 4 | 1 | 5 | 6 | 11 | 17 | 9 | 7 | 16 | 4 | 1 |
| **6** | 17 | 7 | 4 | 5 | 11 | 9 | 16 | 1 | 6 | 17 | 7 | 4 | 5 | 11 | 9 | 16 | 1 |
| **7** | 11 | 1 | 7 | 11 | 1 | 7 | 11 | 1 | 7 | 11 | 1 | 7 | 11 | 1 | 7 | 11 | 1 |
| **8** | 7 | 18 | 11 | 12 | 1 | 8 | 7 | 18 | 11 | 12 | 1 | 8 | 7 | 18 | 11 | 12 | 1 |
| **…** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

* The primitive roots are: 2, 3, 10, 13, 14, 15
  + E.g. , no lower should satisfy
* If is a primitive root in , then are all **distinct**
* For a prime , if is a primitive root in , then make up all the **non-zero elements** of
* Elements of lower orders create periodic sequences

# Logarithm

Logarithm is the **inverse of exponentiation**:

## Discrete Logarithm

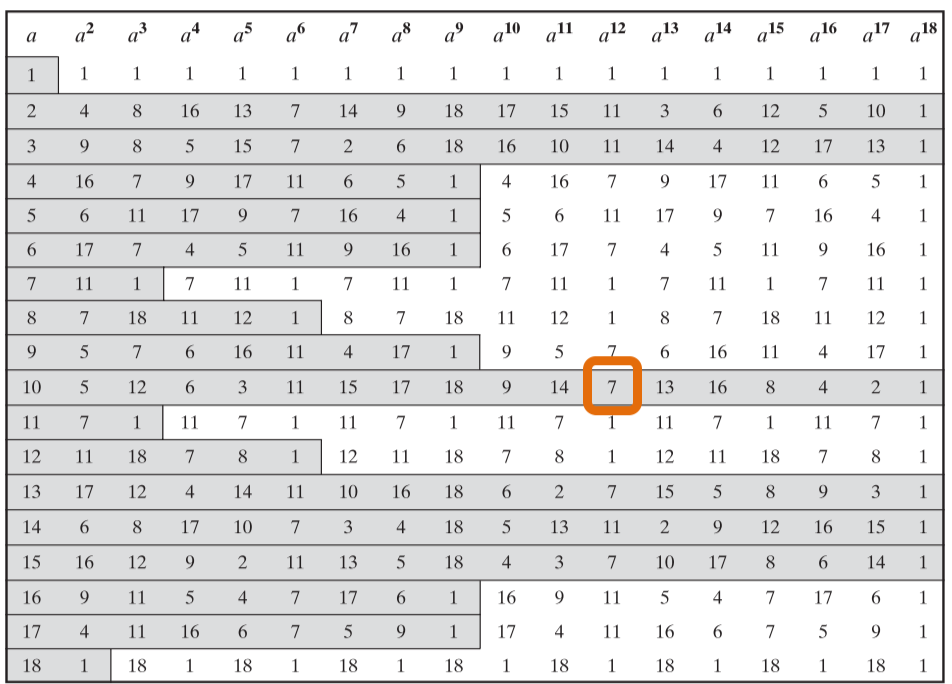
A logarithm in modular arithmetic – has an analogous interpretation as the **inverse of exponentiation**, as with logarithms for real numbers

* Consider a primitive root , for some prime
* For any non-zero integer , we can find a unique where
  + is the discrete logarithm of , for base and prime

### Discrete Logarithm Properties:

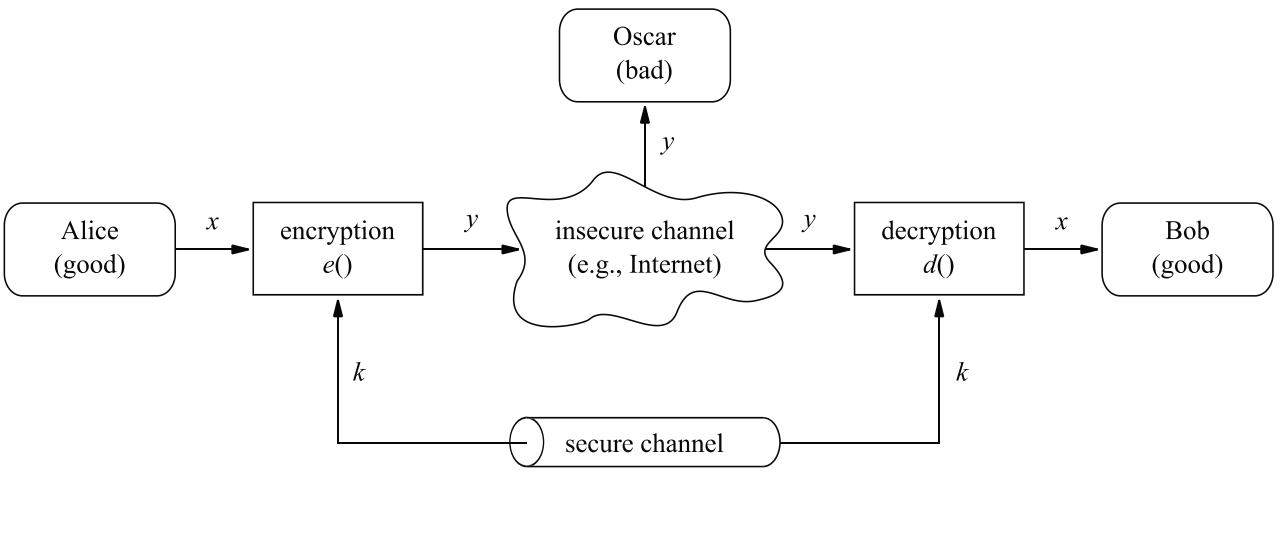
* , since for any
* , since

Example: – discrete logarithm of 7 in base 10, mod 19



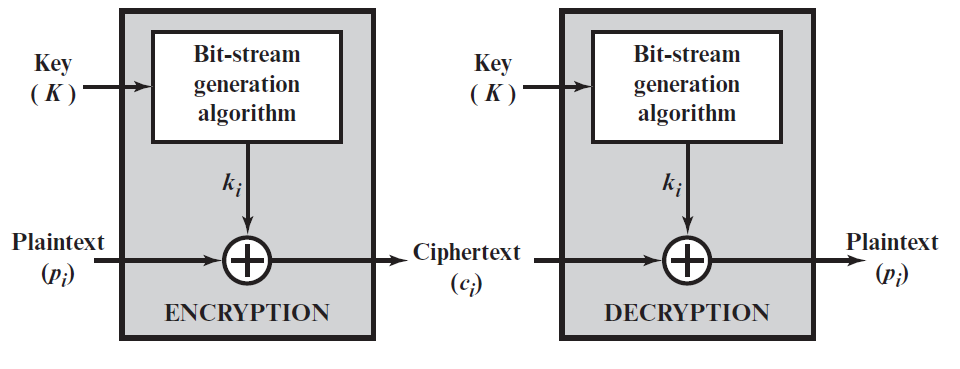
* For , look for **row of** , then **column of power where result =** –answer = power

# Secret Key Cryptography



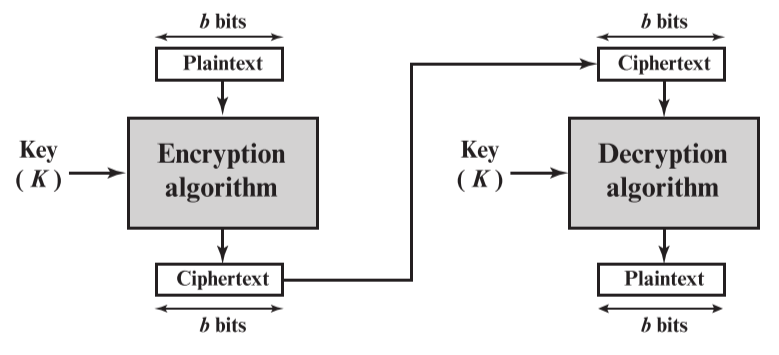
* Alice and Bob share the same secret key

## Stream Ciphers

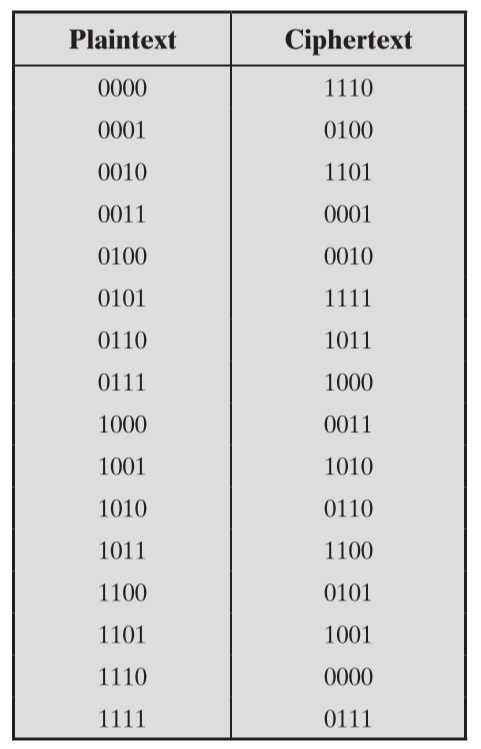
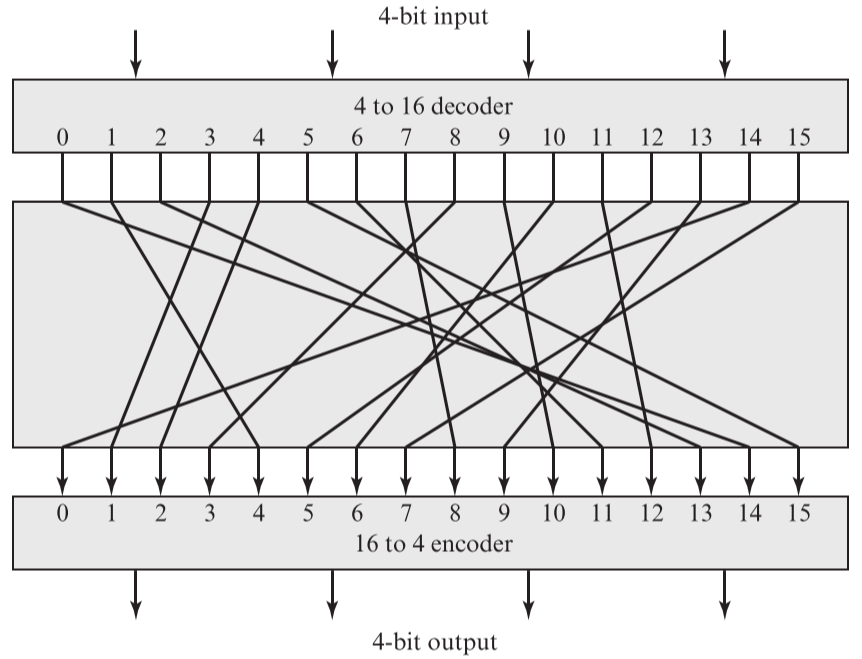


* Process one element (e.g. bit or byte) at a time, continuously
  + One-time pad
  + Vignère and Vernam ciphers
  + ChaCha20 used in TLS (Transport Layer Security)/SSL (Secure Sockets Layer)

## Block Ciphers



* A block of elements (e.g. 64 or 128 bits) is processed together, to create a same-sized block of ciphertext
  + AES
  + DES
* Block ciphers can be used to create stream ciphers
  + Essentially a mapping – for bits plaintext:
    - Input space: possible blocks
    - Output space: possible blocks
    - Number of mappings: reversible mappings, since in order for the mapping to reversible, each mapping must be unique
* Example:

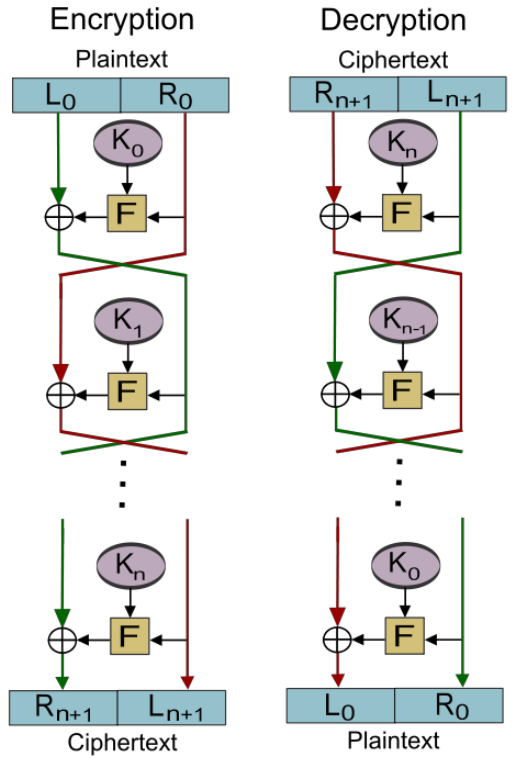


* + 4-bit input produces one of 16 input states
  + Substitution cipher maps input state to one of the 16 output states, then convert back into 4-bit representation
  + **Secret key**: requires bits to represent the mappings
* This is known as the **ideal block cipher**, because it allows the maximum number of encryption mappings from the plaintext block
  + Practical problem: if block size is too small, it becomes a **classical substitution cipher** – vulnerable to plaintext statistical analysis
  + However, ideal block cipher for a large block size is impractical, as it requires bits to be stored to represent the mappings

### In Absence of the Ideal Block Cipher

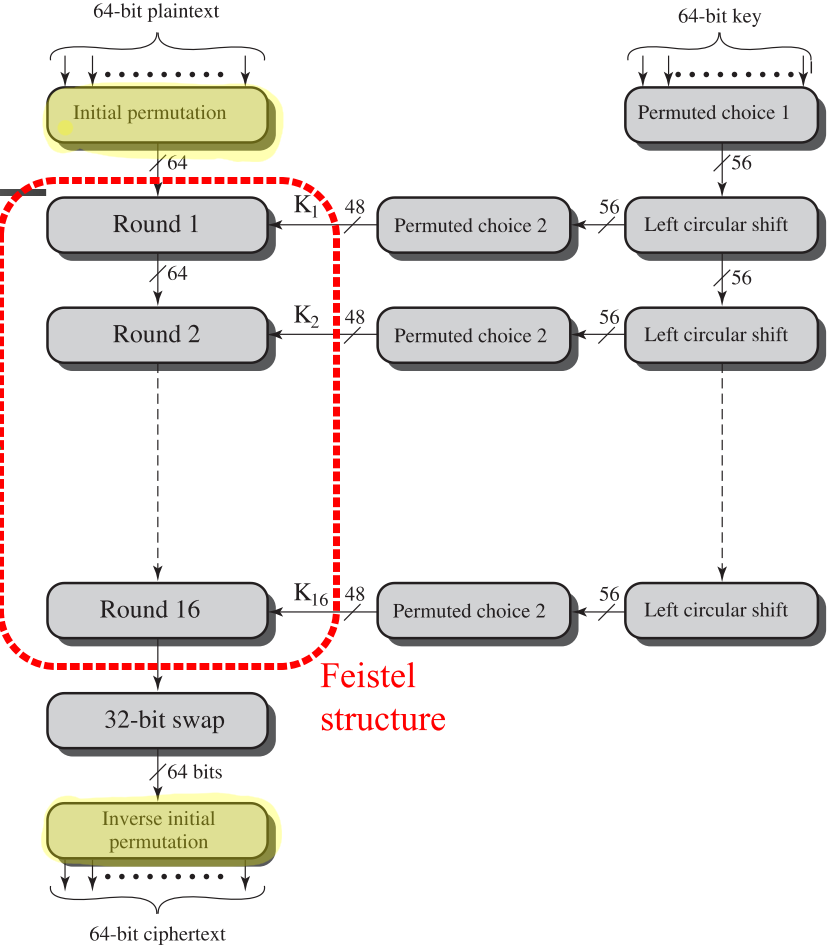
* **Diffusion**: dissipate the statistical structure of the plaintext
* **Confusion**: make the relationship between the ciphertext statistics and the encryption key as complex as possible
* **Cascade**: apply smaller encryption units multiple times, to get stronger encryption

## Feistel Cipher



* Input: plaintext block length and encryption key
* Split input into **two halves**: and
* Alternate and repeat for rounds:
  + Each round has inputs and (from previous round), and subkey derived from K – all subkeys are different from and from each other
  + Substitution: replace left half with round function applied to right half and XOR with left half
  + Permutation: swap two halves
* Design considerations:
  + **Block size**: larger block size means greater security, but reduced encryption/decryption speed; typically use 64 bits or 128 bits (AES)
  + **Key size**: larger key size means greater security, but reduced encryption/decryption speed; typically use 128 bits
  + **Number of rounds**: single round is ineffective, multiple rounds offer increasing security; typically 16 rounds

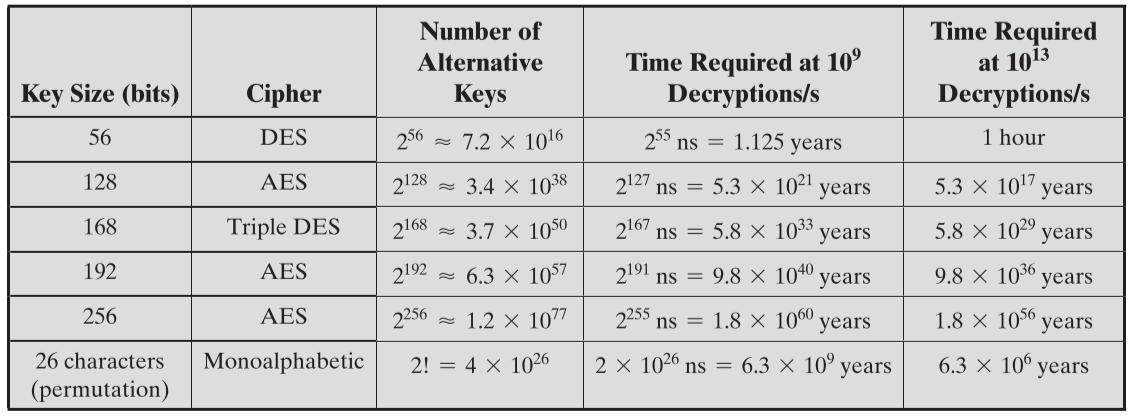
# DES (Data Encryption Standard)



* 64-bit blocks of plaintext input
* 64-bit key – only 56 bits are used, discard last bit from each byte (parity bit)
* 3 phases:
  + Initial Permutation (IP)
  + 16 Feistel rounds (substitution + permutation)
    - At the end, swap left and right halves
  + Inverse of IP
* **Avalanche effect**: small change in plaintext results in large change in ciphertext

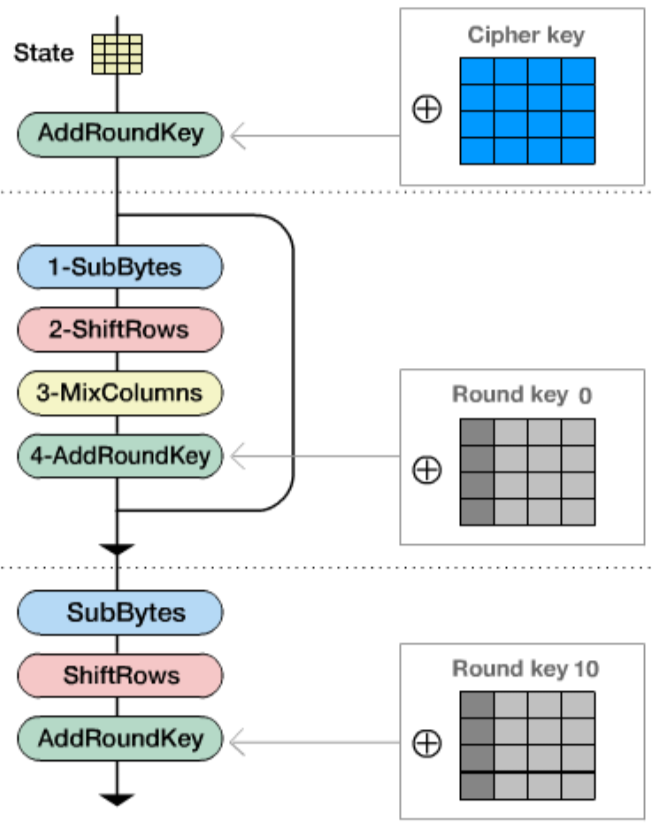
### Strength of DES

* With 56-bit keys, there are possible keys – relatively very fast to break by brute force attack



* Suspicion that S-boxes (used in each Feistel iteration), were designed with a weakness that can be exploited by cryptanalysis if the attacker knows the weakness

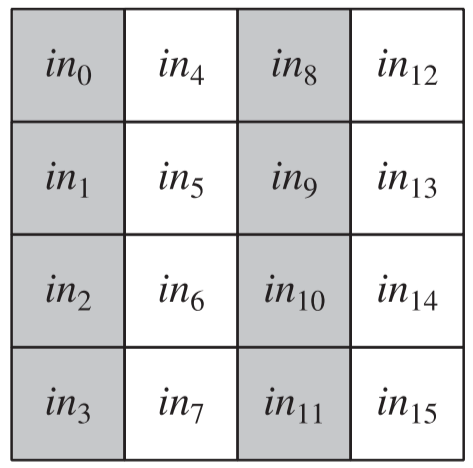
# AES (Advanced Encryption Standard)



* All AES operations are on 8-bit bytes
* Addition, multiplication, and division done in
  + Based on the irreducible polynomial
* 128-bit plaintext input
* 16/32/64-byte keys (AES-128, AES-192, AES-256)
* 10/12/14 rounds – four types of transforms performed per round
  1. Substitute bytes
  2. Shift rows
  3. Mix columns
  4. Add round key

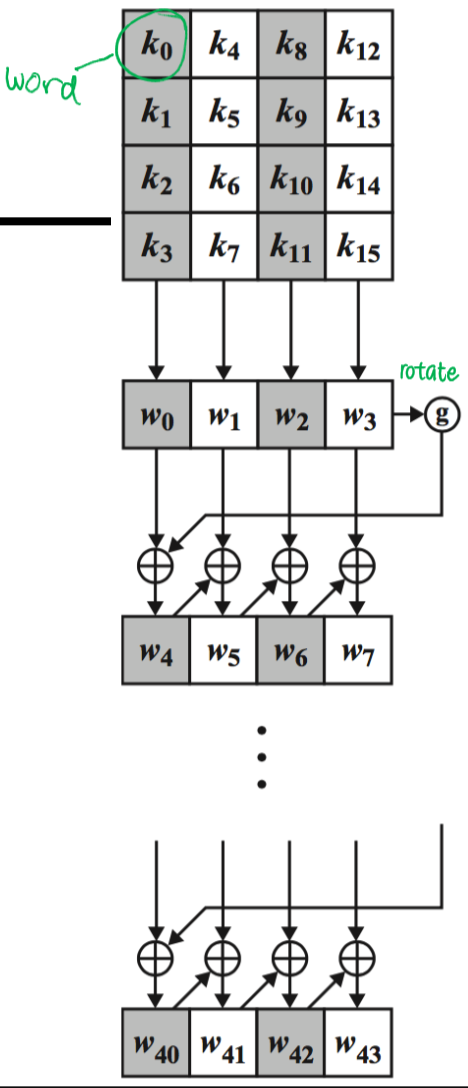
**Inputs**:

* The state: matrix of bytes, ordered by columns



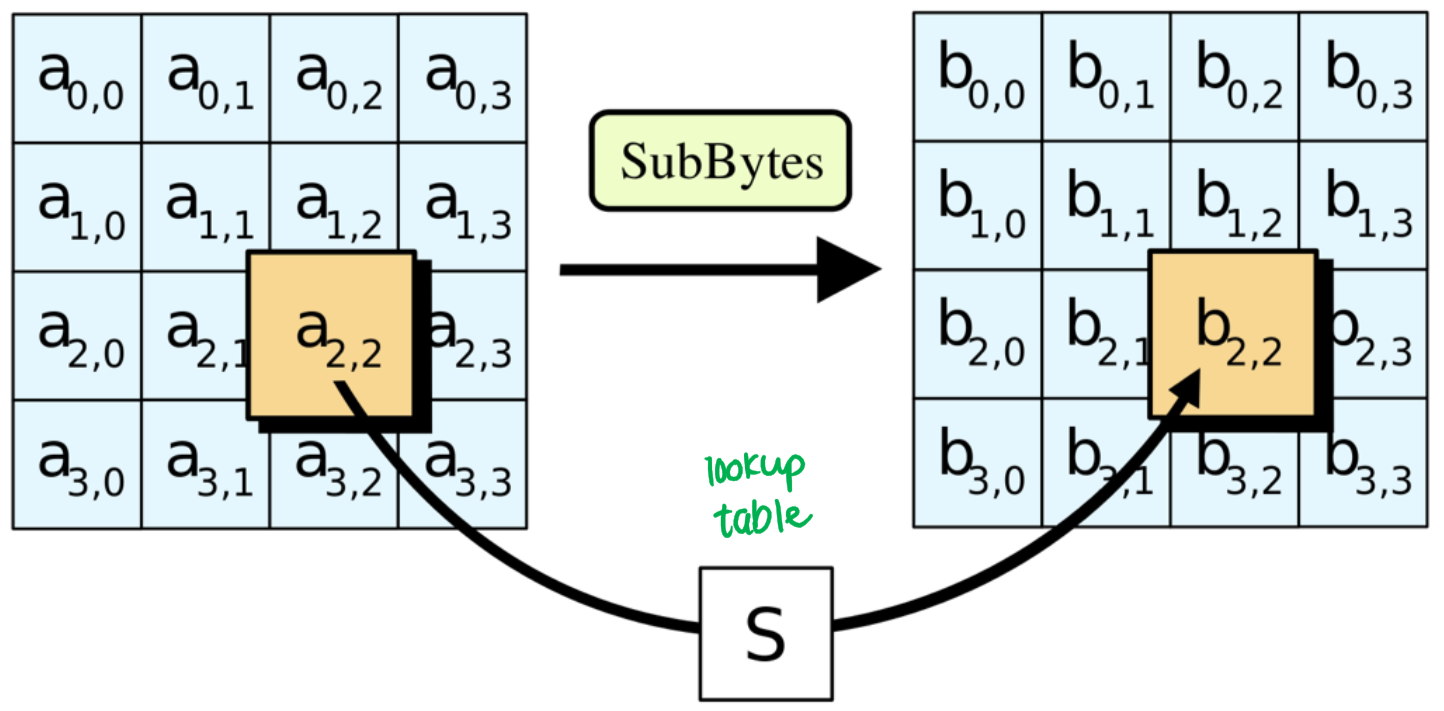
* Cipher key: expanded into round keys, which are the inputs to the *AddRoundKey* transformation each round

### Round Key Generation



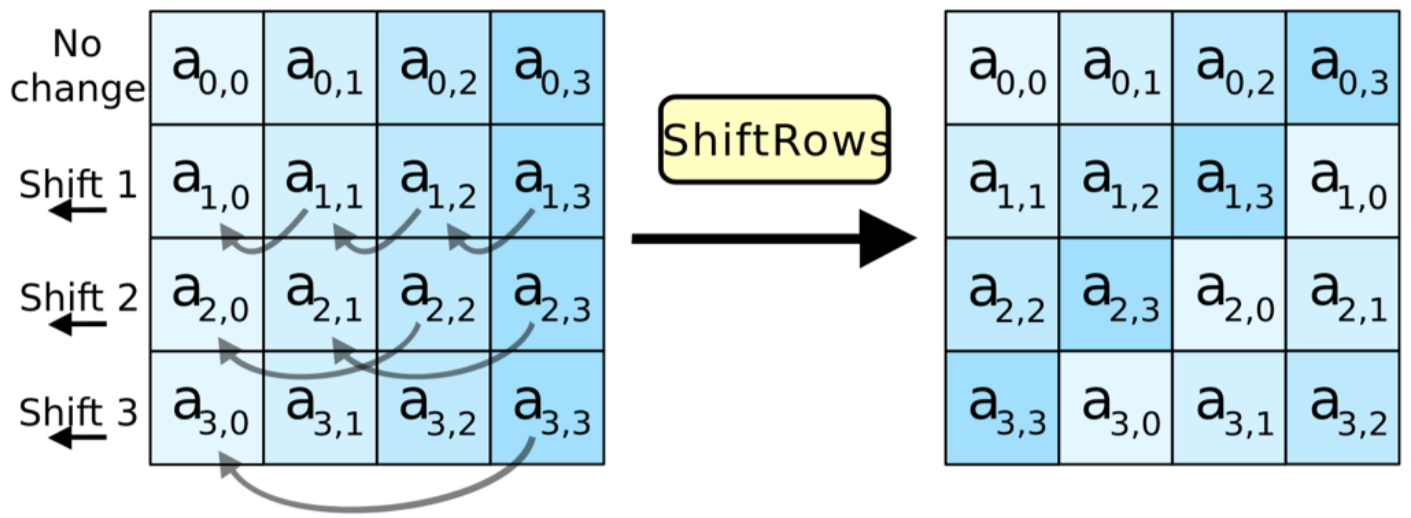
* + 16-byte (four-word) key mapped to a linear array of 44 words (176 bytes)
  + Function rotates bytes, does substitution, and XOR with a round constant
* **Diffusion** (remove statistical structure) of cipher key differences
* **Eliminates symmetries** – creates **non-linearity**

### Substitute Bytes



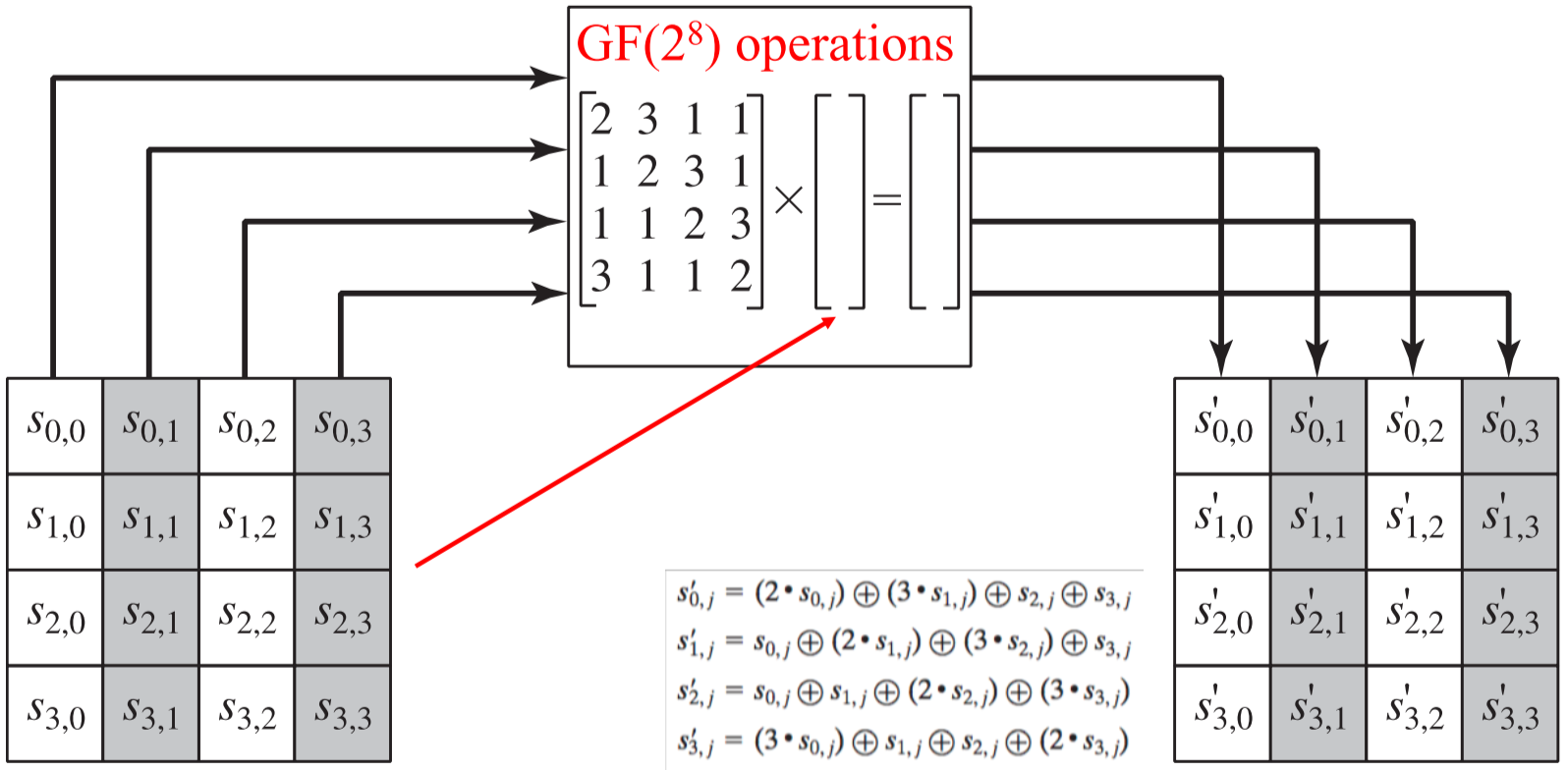
* **S-box** (**look-up table**) contains which value should be substituted, for all 256 possible 8-bit values
  + Take inverse of byte in
  + Take bit representation of inverse, multiply with a matrix, and add to a bit vector
  + Convert back to bytes
* Rationale:
  + Low correlation between input and output bits
  + Outputs a non-linear function of the input
  + Constant is chosen so that:
    - No fixed points, where
    - No opposite fixed points, where (bitwise complement of )
  + The S-box is **invertible**, but there are no **self-inverses** –

### Shift Rows



* The row gets an -left circular shift – this means the four bytes in a column are spread out to different columns

### Mix Columns



* Multiply each column with matrix to produce new column
  + Multiply by 2 = multiplication by = 1-bit left shift + XOR with (0001 1011) if leftmost bit of original value is 1
* Rationale:
  + Mix columns and shift rows operations ensure that, after a few rounds, **all output bits depend on all input bits**

### Add Round Key

* 128 bits of the state are XORed with 128 bits of the round key
* Rationale:
  + Affects every bit of the state – complexity of other stages of AES ensures security

### General Structure

* Cipher begins and ends with an *AddRoundKey* stage – this, in effect, creates a Vernam/One-time pad cipher
* The other stages provide confusion, diffusion, and non-linearity, but no security

# Beyond Block Ciphers

* DES was broken in 1998
* Solution: use DES multiple times in **cascade**

## Double DES

* Use DES twice, with two different keys:
  + Key size becomes bits
* Potential issues:
  + What if ? Turns out to have very low probability
  + Exploit that there must exist – vulnerable to **meet-in-the-middle** attacks using known plaintext/ciphertext pairs

## Meet-in-the-Middle (MITM) Attack

* Encrypt plaintext with all possible keys
* Decrypt ciphertext with all possible keys
* Look for matching
  + One match: chance of false alarm is
  + Two matches: chance of false alarm is – success
* With two pairs of known plaintext/ciphertext pairs, the double DES key can be guessed with high confidence

## 3DES (Triple DES)

* Use DES three times, with two different keys
  + Key size becomes bits
  + MITM cost would be
* 3DES is significantly slower than AES

# Electronic Codebook

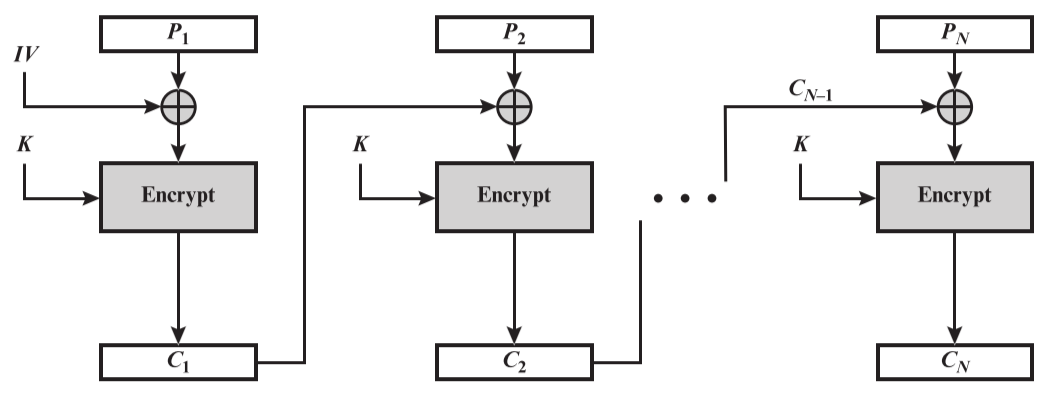
* Block ciphers take a fixed block-size of bits as input and produces a -bit block of ciphertext
* If the plaintext is greater than bits long, block ciphers can still be used by breaking up the plaintext into -bit blocks
  + However, when multiple blocks are encrypted using the same key, security issues arise

## Block Cipher Modes of Operations:

### Electronic Codebook (ECB)

* Plaintext divided into -bit blocks, encrypted individually:
* Good for small messages (size less than one block of underlying cipher) but not large ones, especially if repeats are likely

### Cipher Block Chaining (CBC)



* Input to encryption algorithm is the XOR of the next block of plaintext and the preceding block:
  + For the first block, an **initialization vector (IV)** is XORed with the plaintext
    - IV must be known to sender and receiver, but unpredictable by a third-party
* Limitation: no random access, since decryption must be done in stages

Block ciphers can be converted into stream ciphers

* Eliminates need to pad plaintext to achieve block size
* Encryption happens in real-time

### Cipher Feedback (CFB)

* Plaintext itself isn’t input to the cipher; instead, the bit string that is XORed with the plaintext depends on prior plaintext
* Plaintext divided into -bit segments ( = number of bits for transmission)
  + The leftmost/most significant bits of the encryption function output are XORed with the first segment of plaintext to produce the first segment of ciphertext

### Output Feedback (OFB)

* Output of encryption function is fed back as input to encrypting the next block
* Operates on full blocks of plaintext/ciphertext

Hence,

# Public Key Cryptography

## System Model

* Alice creates a **public/private key pair**
  + Cannot infer private key from public key
  + Data is encrypted with one key, decrypted with the other
    - This means knowing plaintext/ciphertext pairs should not compromise cipher
* Alice keeps private key, shares public key

### Digital Signature (Authentication)

* Sender encrypts with their private key (digital signature), public decrypt with public key
  + Doesn’t keep information confidential, since anyone can decrypt it
* For both authentication and confidentiality: encrypt digitally signed data as plaintext
  + Message
  + Message **hash**
  + Encrypt with sender’s private key – can be decrypted with public key to verify identity
  + Append signed hash with message
  + Generate a symmetric crypto (e.g. AES) **session key**
  + Encrypt with session key – unique to each session
  + Encrypt session key with receiver’s public key – sender can decrypt with its private key
  + Append and transmit

### Public Key Cryptosystems

* Party X generates (private) and (public) keys
* Sender S knows , wants to encrypt plaintext message
* Receiver X knows , receives
* Either key can be used in **either order**:

### Trapdoor functions

* Easy to compute in one direction, difficult to find the inverse (i.e. compute in other direction) unless you have some special information (trapdoor)

# The RSA Algorithm

* Plaintext and ciphertext represented as integers between and , for some
* Block cipher with block size , such that
* Keys:
  + Choose and such that for every
    - for primes
    - – must be relatively prime from
* Encryption/decryption:

Proof:

Need to show that

If and aren’t relatively prime:

If and are relatively prime:

By Euler’s theorem,

Hence,

Thus, divides . The same can be proven for .

If , then   
Similarly, if , then

Then,

Hence, since and divide , then

# Elliptic Curve Cryptography (ECC)

* Prime curves over

Example:

* Claim: finite abelian group if
* **Addition**: for every
  + If then
  + For and , when ,   
     is computed as:
* **Multiplication**: use repeated addition

Example: Consider , with . Determine such that .

## Elliptic Curve PKC

* Based on hardness of discrete logarithm, computing given and
* Known information:
  + – elliptic curve
  + – point on elliptic curve whose order is a large value
* Keys (user X):
  + Private key
  + Public key:
* Encryption/decryption:
  + – is known to sender only

# Cryptographic Hash Functions

* A **hash function** accepts a variable-length block of data and produces a fixed-size hash value
  + **Cryptographic hash functions** – algorithm where it is computationally infeasible to:
    - Find a data value that maps to the hash value (**one-way**)
    - Two data values that map to the same hash value (**collision-free**)

## Applications of Cryptographic Hash Functions

### Message Authentication

* Verify integrity of a message – data is received as sent; no modification, insertion, deletion, etc.
* **Message digest** – hash function value that provides message authentication
  + Sender performs hash function on message, sends hash value along with message
  + Receiver performs same hash function on message, compares with received hash value
  + If there is a mismatch, either message or hash value has been altered
* To prevent attacks, hash value must be transmitted securely
  + **Message and hash** are encrypted using symmetric encryption – provides confidentiality
  + **Only hash** is encrypted using symmetric encryption – does not provide confidentiality
  + If sender and receiver both know a **secret value S**, then hash can be computed over **M+S**; since S is a secret value, attacker cannot modify hash or generate false message
  + Confidentiality can be applied to ^ by encrypting message and hash
* **Message Authentication Code (MAC)**
  + Keyed hash function used between two parties to share a secret key to authenticate information
  + MAC function takes secret key and message as input, produces hash value
  + To check integrity, apply MAC function to message at receiver, compare with MAC value
  + Attacker cannot modify MAC value without knowing secret key

### Digital Signatures

* Hash value of a message encrypted with sender’s private key
* Anyone who wants to verify integrity of message can use public key to decrypt, check hash value
* Attacker cannot modify hash value because they do not know the private key
* If confidentiality is also desired, message and hash code (digital signature) can be encrypted with a symmetric secret key

## Two Simple Hash Functions

### XOR Hash

Given an input of -bit blocks, produces an -bit hash:

* Each hash bit is the XOR of the th bit of each of the blocks in the input
* This operation produces a parity bit for each bit location – since every hash value is equally likely, the probability that a data error will result in the same hash value is

### Rotated XOR Hash

1. Initially set the -bit hash value to zero
2. Process each -bit block of data:
   * Rotate current hash value to the left by one bit (e.g. 101101 🡪 011011)
   * XOR the block with the hash value

* This method provides good data integrity but bad security, because it is easy to produce a message that yields the same hash code

### Cipher Block Chaining Hash

Given a message *M* consisting of a sequence of 64-bit blocks

The hash function is a block-by-block XOR of all blocks, appending the hash code to the message as the final block:

Next, encrypt the hash message using CBC to produce

* Hence, the hash code won’t change if the ciphertext was permuted

## Requirements and Security of Hash Functions

* **Preimage**: for a hash , is the preimage of , a data block with hash value is , using hash function
* **Collision**: when two preimages but
  + Since hash functions are being used for data integrity, collisions are undesirable
* If the length of the hash code is bits, and the hash function takes in -bit blocks ():
  + The total number of message values is
  + The total number of hash values is
  + Hence, on average, each hash value corresponds to message values

### Requirements for Cryptographic Hash Functions

1. **Variable** **input** **size**
2. **Fixed output** **size**
3. **Efficiency**
4. **Preimage resistant (one-way)** – can’t derive from
5. **Second preimage resistant (weak collision resistant)** – for a given block , computationally infeasible to find where and
6. **Collision resistant (strong collision resistant)** – computationally infeasible to find any pair where and
7. **Pseudorandom**