Introduction - 04/09

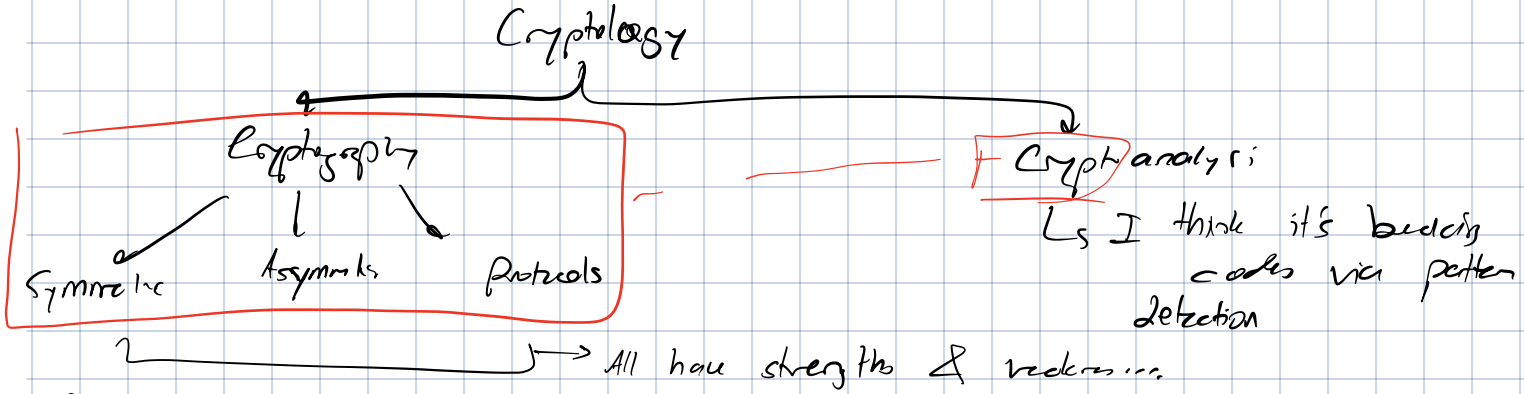
Logistics

- Monday: 2.3.B01
- Wed: 2.3.C04 (this Wed → lab) } Double session sometimes
  - Lab: Lab.2.2.C.05
- Python labs
- Group activity on crypto research → Dec.
- Assessment:
  - 2 midterms → each 25%
  - Lab exam: 40%. This is our final exam. Need to get 40% to pass (Focus)
  - Research & tell: 10%.
- Midterm I: Nov. 13 (Topics 0-5)
- Midterm II: Dec. 4th (mostly real, but 0-5 mostly)
- Research & tell: Dec. 11th & 18th
- TODO: request Lab Account (in slides)

Intro

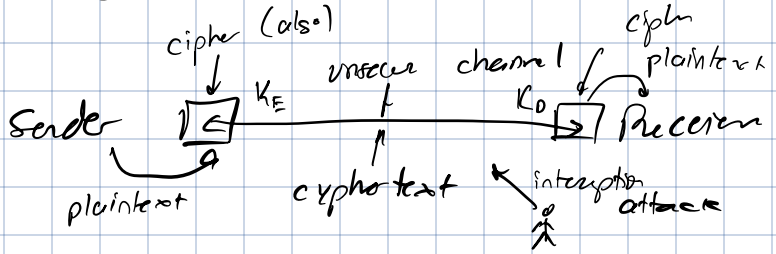
- Defn (2000 B.C → 1948): hide info
- " (1948 →): ", establish authenticity, protect from unauth mod. & repudiation
  - Confidentiality + data integ.
- Comp. sec: protect info from confidentiality, integ., availability (DDoS) (cryptography doesn't have this)
- Security aspects:
  - Prevention
  - Detection
  - Reaction

} Needs cryptography



Cryptanalysis: I think it's breaking codes via pattern detection

- Cryptosystem model

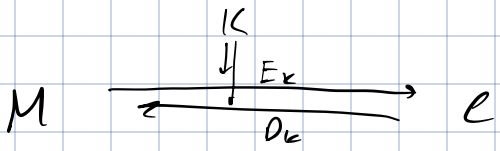


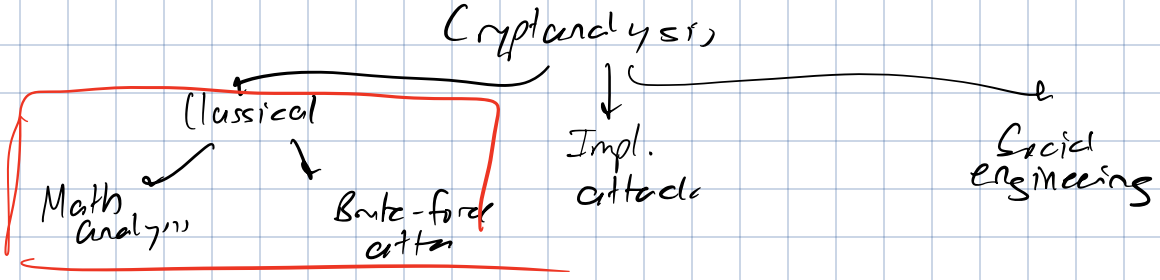
$\Rightarrow K_E \neq K_D$ all the time!

- Math:
  1. Message space: $M = \{m_1, m_2, \ldots\}$
  2. Cipher text space: $C = \{c_1, c_2, \ldots\}$
  3. Key space: $K = \{k_1, k_2, \ldots\}$

} encryption functions (E_k)

} decryption function (D_k)





MATHEMATICAL BACKGROUND

- Congruence: $a \equiv b \bmod n$ (n = modulus) if $a - b$ is int. multiple of n
  - Alt: $a \,\%\, n = b \,\%\, n$
  - Ex: // $24 \equiv -27 \bmod 6$
  - $\bmod n: \mathbb{Z} \to \{0, \ldots, n-1\}$

Set Z_n

- $Z_n = \{[a] \,/\, [a] \in \mathbb{Z}\} = \{0, \ldots, n-1\}$
  - $Z_8 = \{0, \ldots, 7\}$
- Operations:
  - $+_n$: $[a] +_n [b] = (a+b) \bmod n$
    - $[6] +_8 [7] = (6+7) \bmod 8 = 13 \bmod 8 = 5 \bmod 8$
  - $\cdot_n$: $[a] \cdot_n [b] = (a \cdot b) \bmod n$
    - $[3] \cdot_{31} [11] = (3 \cdot 11) \bmod 31 = 2 \bmod 31$
  - Follows regular addition & multiplication properties (can prove)
- Properties:
  - $(a+b) \bmod n = a \bmod n + b \bmod n$
  - $(a \cdot b) \bmod n = a \bmod n \cdot b \bmod n$
    - $7^4 \bmod 5 = (7 \bmod 5)^4 = 2^4 \bmod 5 = 1$
    - V. USEFUL IN EXAM
  - $[a \cdot (b+c)] = [a] \cdot_n ([b] +_n [c])$
- Look @ homework on Math Background lecture

Set Z_n^* & Euler's Totient Function

- Z_n^* is subset of Z_n s.t. members not coprime (no common factors) w/ n
- Ex: //

Z_8	Inv. add. $(-w)$	Inv. mult. (w^{-1})
0	0	—
1	7	1
2	6	—
3	5	3
4	4	—
5	3	5
6	2	—
7	1	7

$+_n$ w/ $Z_n \to 0$

$\cdot_n$ w/ $Z_n \to 1$

- All elements of Z_n^* have multiplicative inverse. Z_n^* also called reduced set of residuals
- Ex: //

$$Z_{10}^* = \{1, 3, 7, 9\}$$

- Totient function:

$$\Phi(n) = \text{len}(Z_n^*)$$

- Rules:
  1. If p is prime: $\Phi(p) = p - 1$
  2. If p is prime & $k \in Z^+$: $\Phi(p^k) = p^k - p^{k-1} = p^{k-1}(p-1)$ (Z^+: +ve)
  3. If p & q is coprime: $\Phi(p \cdot q) = \Phi(p) \cdot \Phi(q)$
  4. $\forall i$, if $n = \prod p_i^{k_i}$ where p_i is prime & $k_i \in Z^+$:
     $\Phi(n) = \prod p_i^{k_i - 1} \cdot (p_i - 1)$ (prime factorize)
- Ex: //

$\Phi(7) = 6$ (7 is prime)

$\Phi(11^4) = 11^4 - 11^3 = 11^3(11-1) = 10 \cdot 11^3$

$\Phi(4) = \Phi(2^2) = 2^2 - 2 = 4 - 2 = 2$

$\Phi(33) = \Phi(3) \cdot \Phi(11) = 2 \cdot 10 = 20$

$\Phi(2^3 \cdot 5^7) = (2^2 \cdot 1) \cdot (5^6 \cdot 4)$

$\Phi(172) = \Phi(43 \cdot 2^2) = 42 \cdot (2^2 - 2)$
$= 42 \cdot 2$
$= 84$

Fermat's Theorem

- Theorem:

If p is prime, $\forall a \in Z$, $\gcd(a, p) = 1$ (coprime) $\rightarrow a^{p-1} \bmod p = 1$

- Useful if calculating extremely large modulus
- Note:

$\times a$ ($a^{p-1} \bmod p = 1$) $\times a$

$a^p \bmod p = a$

↳ Another way

- Note that:

$a^{p-1} \bmod p = 1$

$a \cdot a^{p-2} \bmod p = 1$

a^{p-2} is mult. inv. of $a \bmod p$

a^{-1}

Find mult. inv. of #

Euler's Theorem

- Theorem:

$\forall a, n \in Z$ $(n \neq 0)$. $\gcd(a, n) = 1$

$a^{\Phi(n)} \bmod n = 1$

→ n doesn't have to be prime

So:

$$a \cdot a^{\Phi(n)-1} \bmod n = 1 \Rightarrow a^{-1} = a^{\Phi(n)-1} \bmod n$$

- Ex:// Solve: $3x \bmod 10 = 1$

Cannot use Fermat since 10 isn't prime. Use Euler

$a=3,\ n=10 \rightarrow \Phi(10) = \Phi(5) \cdot \Phi(2)$

$= 4 \cdot 1$

$= 4$

Now:

$$x = 3^{\Phi(10)-1} \bmod 10$$
$$= 3^{4-1} \bmod 10$$
$$= 3^3 \bmod 10$$
$$= 7$$

- Ex:// Solv $2x \bmod 11 = 1$

11 is prime → Fermat

$$\therefore x = 2^9 \bmod 11$$
$$= (2^3)^3 \bmod 11$$
$$= 3^3 \bmod 11$$
$$= 5$$

- Ex:// Solve $17x \bmod 12 = 1$

$5x \bmod 12 = 1$

Euler:

$$x = 5^{\Phi(12)-1} \bmod 12$$
$$= 5^{4-1} \bmod 12$$
$$= 5^3 \bmod 12$$
$$= \ldots$$

Linear Congruence Equations

- Recall: $a \cdot x = b \bmod n \rightarrow ax + nk = b \quad \exists k$
- If a & n are coprim, unique solution

$$x = b \cdot y \bmod n$$

where $y = a^{-1} \bmod n$ (mult. inv. of a)

↳ $a \cdot y \bmod n = 1$ (use prev. recipe)

- If a & n aren't coprime but $\gcd(a, n) = m$ & $m \mid b$, $\exists$ m solns btw 0 & m-1 → $x = (b/m) \cdot y + j(n/m) \bmod n$
- O.U.: no soln. ↳ $(a/m) \cdot y \bmod (n/m) = 1, \quad j \in \{0, \ldots, m-1\}$
- Ex:// $10x = 15 \bmod 25$

① Classify:

$\gcd(10, 25) = 5$

Now: $15 \mid 5$ so 5 solutions

② Find b:

$2y \bmod 5 = 1$

S is Prime, so Fermat

$b = 2^{\phi(5)-1} \bmod 5$

$= 2^3 \bmod 5$

$= 3$

③ $x_s = (3 \cdot 3 + 5j) \bmod 25$

$= \{9, 14, 19, 24, 4\}$ ⇒ Soln. on Acla

- Ex: // $15x = 14 \bmod 21$

① Classify:

$a = 15, \quad b = 14, \quad n = 21$

$\gcd(15, 21) = 3$

Since $b \nmid m$, no solution

Extended Euclidean Algorithm

- Euclidean Algorithm: find $\gcd(a,b)$ given $a > b$
  1. Write out:
     $a = bq + r$ (r is remainder of a/b, $q = a // b$)
  2. While $r \neq 0$:
     a) $a := b, \quad b := r$
- Extended Euclidean Algorithm:
  1. Do Euclidean
  2. Reverse & sub to write out in original #s
- Ex:// Solve $17x = 1 \bmod 43$

① Do Euclidean to find $\gcd(17, 43) = 1$

$43 = 17 \times 2 + 9$

$17 = 9 \times 1 + 8$

$9 = 8 \times 1 + 1$

② Extend:

$1 = 9 - 8$

$= 9 - (17 - 9)$

$= 2 \times 9 - 17$

$1 = 2(43 - 17 \times 2) - 17$

$1 = 2 \times 43 - 17 \times 5$

③ Mod w 43

$1 \bmod 43 = -5 \times 17 \bmod 43$

④ Get x:

$x = -5 \bmod 43$

$= 38$

Primitive Roots

- Consider elements g such that $a^g = 1 \bmod n$ & $GCD(a, n) = 1$
  - At least $g = \phi(n)$ but can be smaller / greater
- Primitive root generator: if at least on a s.t. smallest g satisfying eq. is $\phi(n)$, then a is root generator
  - Not all n have primitive roots.
  - Ex: 3 is primitive root gen. for 19.

- Defn: $ord_n a$:= smallest number m s.t. $a^m \equiv 1 \bmod n$ (given $\gcd(a, n) = 1$)
  - Important: $ord_n a \mid \Phi(n)$
- Defn: r is primitive root mod n if $ord_n r = \Phi(n)$
- Ex:// Find primitive root of $n = 13$

  ① Totient:

  $$\Phi(n) = 12$$

  ② Limit:

  Possible orders: 1, 2, 3, 4, 6, 12 (since ord | Φ)

  Possible primitive roots: 0 → n-1 that is coprime

  ③ Check if any number coprime w/ n has order of $\Phi(n)$

  2: $2^1 = 2$
  $2^2 = 4$
  $2^3 = 8$ (mod 13)
  $2^4 = 16 = 3$
  $2^6 = -1$

  Now can do it, must be 12

  $\Rightarrow ord_{13} 2 = 12$ ✓
- Primitive root raised to successive powers mod n generates group $Z_{\Phi(n)}^*$
  - In other words, if a is primitive root of p if $a^b \bmod p$ $\forall$ b, $0 \le b \le p-1$ is distinct

Galois Fields

- Defn: set of polynomials $a(x)$ of degree $n-1$ or less w/ coef. in Z_q (q is prime)
  - Math: $a(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \dots + a_1 x + a_0$, $a_i \in Z_q$
- Total # of polynomials = q^n
- $a(x)$ is remainder of dividing polynomials via reducing polynomial $p(x)$ w/ deg. of n
  - $p(x)$ is irreducible & primitive (generate of a) (irreducible = cannot factor)
- +, · → finite field $GF(q^n)$
- Ex:// $GF(2^5)$

  $a(x) = a_4 x^4 + a_3 x^3 + \dots + a_0$, $a_i \in Z_2 = \{0, 1\}$

  ↳ $x^4 + x^2 + 1 \rightarrow x^4 + 0x^3 + x^2 + 0x + 1$
  $= 10101$ (bin)
  $= 21$ (dec)
  $= 15$ (hex)

  ↳ $x^4 + x^3 + x^2 \rightarrow 11100$ (bin) . . .

  Primitive polynomial: $p(x) = x^5 + x^2 + 1$
- Operations:
  - Addition & subtraction:

    $$c_i = (a_i \pm b_i) \bmod 2 = a_i \oplus b_i$$

    Ex:// $a = (10110)$, $b = (10101)$ in $GF(2^5)$. What is c?

    $10110 \oplus 10101 = 00011$
  - Multiplication

$$c(x) = a(x) \cdot b(x) \bmod p(x)$$

- Coefficients of same degree need mod 2 reduction
- If $c(x)$ degree $> n-1$, reduce by $p(x)$
- Ex: // $a = (101)$ in $GF(2^3)$, $p(x) = (1011)$. Compute $c = a \cdot a$

① Multiply:

$$101 \rightarrow x^2 + 1$$

$$\therefore (101)(101) = (x^2+1)(x^2+1)$$

$$= x^4 + 2x^2 + 1$$

② Mod 2:

$$x^4 + \overset{0}{\cancel{2}}x^2 + 1 \rightarrow x^4 + 1$$

③ Reduce by $p(x)$:

$(1011) \rightarrow x^3 + x + 1$

$$\begin{array}{r|l} & x \\ \hline x^4 + 0x^3 + 0x^2 + 0x + 1 & \\ -x^4 + 0x^3 + x^2 + x + 0 & \\ \hline -x^2 - x + 1 & \end{array}$$

$$-x^2 - x + 1 \rightarrow \boxed{x^2 + x + 1} \sim (111)$$

$\uparrow \bmod 2$

- Computing multiplicative inverses: $a^{-1} = a^{\Phi(GF(2^n))-1} \bmod p(x) = a^{2n-2}$
  - $\Phi(GF(2^n)) = 2^n - 1$
  - Ex: // Compute the inverse of $a(x) = x^2 = (100)$ in $GF(2^3)$ w/ $p(x) = x^3 + x + 1$ (1011).

① Calculate Φ

$$\Phi(GF(2^n)) = 2^n - 1 = 7$$

② Inverse:

$$a^{-1} = (100)^{\Phi(GF(2^n))-1} \bmod 1011$$

$$= (100)^6 \bmod 1011$$

$$= (x^2)^6 \bmod 1011$$

③ Break into polynomials w/ $\deg < \deg(p(x))$

$$(x^2)^6 = x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x^2 \bmod \ldots$$

④ Multiply from right to left & reduce by mod $p(x)$ if need.

$$x^2 \cdot x = 1000 \oplus 1011 = 011 \sim x^9$$

$$011 \cdot x = 011 \cdot 010 = 110$$

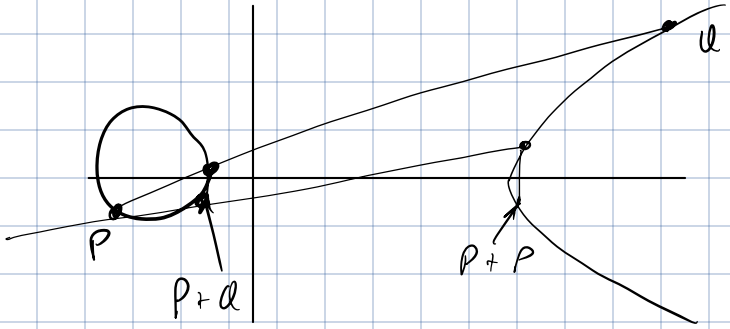
$x^4 \cdot x$: :

$$\rightarrow \boxed{111}$$

Elliptic Curves

- General formula: $E(a,b): y^2 = x^3 + ax + b$
  - To plot, we need square root of RHS
  - Symmetric along x-axis (not always at $y=0$)
  - Also includes point @ infinity / zero point O
- Operations:
  - $O = -O$
  - $P = (x,y) \rightarrow -P = (x,-y)$
  - To add 2 points:
    1. Draw line
    2. Find 3rd pt. of intersection
    3. $P + Q = -R$

    } If line is tgt at P/Q, $R = P$ or $R = Q$
  - To double pt: Draw tgent line & find intersection ($P+P = 2P = -R$)
- Ex:



- Cryptography uses finite field elliptic curves where variables/coef. $\in$ set
  - Prime curves over Z_p: $E_p(a,b) = y^2 \bmod p = (x^3 + ax + b) \bmod p$
  - Binary curves over $GF(2^n)$: $E_{2^n}(a,b) = (y^2 + xy) \bmod p = (x^3 + ax^2 + b) \bmod p$
  - Not all curves are good for cryptography ← need adequate # of elem.

CLASSICAL CRYPTOGRAPHY

Introduction

Started in 5th century

Based on key & symmetric cipher algo (both parties use same key for encrypt & decrypt)

Techniques

Substitution

Each character/letter in plain text is modified/replaced

Transposition (permutation)

All chars reallocated in ciphertext w/out modification

Transposition

Recall: characters only change posn in ciphertext
↳ Frequency of letters is not modified

① Rail fence transposition

Method:
1: Decide on # of lines
2: Write msg in zigzag going up & down + seq. across lines.
3: Compose ciphertext by reading line by line

Ex:// Do 3-line cipher of "Civil war field cipher"

C L F D H
I I W R I L C P E
V A E I R

C L F D H I I W R I L C P E V A E I R

② Group transposition

Π_M : describes order for a group of p letters.

Ex:// $\Pi_M = 2\,4\,3\,5\,1$ → 2nd letter → 1st
4th letter → 2nd
⋮

If message is M & C is cipher:

M = MANOS ARRIB AESTO
C = AOSNM RIBRA ETOSA

Increasing # of letters (p) makes this less vulnerable
As p ↑, transposition by series

③ Transposition by series

$M' = M_{S_1} M_{S_2} M_{S_3} \ldots M_{S_\theta}$

Group transposition

Ensure that each M_{S_i} is mut. excl.

④ Transposition by columns/rows

Method
1. Symbols placed in some geometric patterns (usually matrix)
2. Read row by row OR col by col

Alternately:
2. Set key for message
3. Rearrange col/row & tell key to message so they know how to unscramble

Ex:// Key = ESPIA. M = EJEMPLO DE TRANSPOSICION CON CLAVE

Rearrange key w/ Alphabetical order

① Matrix

E	S	P	I	A
E	J	E	M	P
L	O	D	E	T
R	A	N	S	P
O	S	I	C	I
O	N	C	O	L
U	M	N	A	R
C	O	N	C	L
A	V	E	X	X

Empty

② Rearrange cols alphabetically via key

A	E	I	P	S
P	E	M	E	.
T	L	E	D	/
P	R	S	N	/
I	O	C	I	/
L	O	O	C	/
R	U	A	N	

L C N
X A C E

③ Read column by column:

C = PTPILRLX ELR . . .

Substitution

Defn: substitute alphabet of messag w/ alphabet of cipher

↳ Ex: // A-Z → 0-25

A-Z 0-9 → 0-35

① Simple monoalphabetic substitution (monographic)

Sub 1 character of plaintext alphabet for 1 character in ciphertext alphabet

Ex: //

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
b	c	d	f	h	i	n	r	t	x	e	s	j	k	m	o	q

R	S	T	U	V	W	X	Y	Z
s	u	w	y	a	g	p	v	z

M: THIS IS A SUPER SECRET MESSAGE

C: wrtu tu b vyohs uhdshw jhuu bnh

n! possible ciphers!

Substitution might follow following relationship:

$$E(m_i) \equiv (a \cdot m_i + b) \bmod n$$

- $E(m_i)$: encrypted letter #
- a: decimation constant
- m_i: letter # (0-n-1)
- b: shift constant
- n: # of letters in alphabet

∴ Key: (a, b)

- Condition: a, n must be coprime

Ex: // Caesar cipher is $E(m_i) = (m_i + 3) \bmod n$

A B C D E F

x y z a b c

② Polyalphabetic substitution (Vignére)

$$E(m_i) = (m_i + K_{(i \bmod m)}) \bmod 26$$

- $E(m_i)$: encrypted char
- m_i: letter in i^{th} posn
- K_j: shift of alphabet j
- m: length of key

Vignére utilizes a table:

	A	B	C	...	Z
A	A	B	C	...	Z
B	B	C	D	...	A
C	C	D	E	...	B
D	D	E	F	...	C
⋮					
Z	Z	A	B	...	Y

(cleartext → columns; encryption alphabet → rows)

cell is encryption of cleartext letter via encryption alphabet (D in this case)

Key defines shift for each letter. Periodic

Ex:// Message: HEL LOM ATE
Key: SOL SOL SOL

Encrypt:

① 1st letter: Go to "S" encrypt alphabet & "H" column → encrypt letter

∴ H → Z

② 2nd letter: Go to "O" encrypt alph. & "E" col.

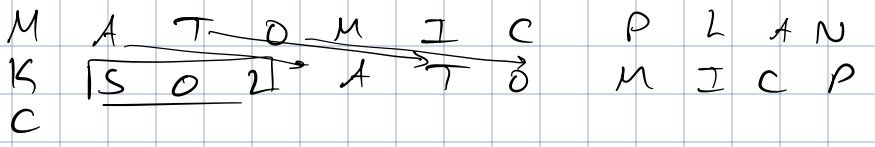
∴ E → S

⋮

Ciphertext: ZSW DCX SHP

Autokey: use key once & then use plain text to encrypt

↳ Ex: // M: ATOMICPLAN. Key: SOL



③ Non-periodic polyalphabetic substitution (Vernam)

$$E(m_i) = m_i \oplus k_j \quad , \quad D(m_i) = c_i \oplus k_j$$

Perfect secrecy if key is:
1. Random
2. ≥ plaintext
3. Never reuse
4. Secret

Pros: perfect secrecy, hard to cryptoanalyze

Cons: key size ↑, randomness, key can only be use once

Fun fact: Enigma use polyalphabetic subn.

Cryptanalysis

Used to break encrypted msgs, even if key is unknown

① Breaking monoalphabetic substitution ciphers

A: Brute force

Go through all possible shifts in alphabet

$$E_n(x) = (x + n) \mod 26$$
$$D_n(x) = (x - n) \mod 26$$

↑ vary n from 0 – len(alphabet)

B: Frequency analysis

Characters will have own frequencies in lang. Look @ freq. of ciphertext chars to figure out most likely shift.

If it's affine/linear encryption, then

$$E(m_i) = (a \cdot m_i + b) \mod n$$

& used freq. to find a & b

Ex:// Suppose in ciphertext D & V are most frequent characters. Find a & b

① Possible encryption:

$$\underset{(3)}{E} \longrightarrow \underset{(4)}{D}, \quad \underset{(19)}{T} \longrightarrow \underset{(21)}{V}$$

② Simultaneous eqns:

$$3 \equiv 4a + b \mod 26 \quad - A$$
$$21 \equiv 19a + b \mod 26 \quad - B$$

③ Solve

$$A: \quad b = 3 - 4a \mod 26$$
$$\text{Sub in } B: \quad 21 \equiv 19a - 4a + 3 \mod 26$$
$$18 \equiv 15a \mod 26$$

Since $\gcd(15, 26) = 1$, we can apply Euler

$$\boxed{a = 22, \quad b = 19}$$

Pure shifting: $a = 1$
Affine " : $a > 1$ & $b > 0$

② Breaking polyalphabetic substitution ciphers

Usually, distance of repetitions of substring is some multiple of the key length

Can use Kasiski Method:

① Estimate key length

i) Look for substring repetitions

ii) Find distances b/w substrings

iii) Compute GCD of distances → key length estimate (L)

② Extract sub cryptograms of length L

i) Take ciphertext & split into substrings of length L

ii) Take i^{th} char in each → i^{th} sub-cryptogram

③ Frequency attack

i) Look at most common letter in each sub-cryptogram

ii) Substitute w English char freq.

Fundamental Cryptography Concepts

Defining Security

Two models of security:

① Informational/unconditional security

A: Ciphertext leaks no info on plain text

B: Ciphertext is secure if cannot be broken given unlimited time & memory

Known as perfect secrecy

Ciphers that are informationally secure:

- OTP: relies on key randomness, long key length ($\geq$ len(plain)), only use once key
  - Not practical b/c true randomness hard to achieve
- Vernam

All other ciphers are not this secure b/c ciphertext leaks info

② Computational security

Best-known algo to break requires unreasonable comp. resources

Security goals:

① Indistinguishability (IND): ciphertext $\approx$ random string

② Nonmalleable (NM): given $C_1 = E(K, P_1)$, shouldn't create C_2 whose Plaintext P_2 is close to P_1

Attack models:

Set of assumptions about how attacker interacts w cipher & capabilities

Ideally, system should be secure if attacker knows everything but key

Models:

strength ↓

A: Ciphertext-only attack (COA)

B: known-plaintext attack (KPA): adv knows plaintext

C: Chosen-plaintext attack (CPA): can encrypt any plaintext

D: Chosen-ciphertext " (CCA): can do both encrypt/decrypt for any ciphertext/encryptiontext

E: Gray-box: has access to cipher impl.

We want IND-CPA usually

↳ This requires diff. encryptions of same plaintext → randomized encrypt

Authentication security properties:

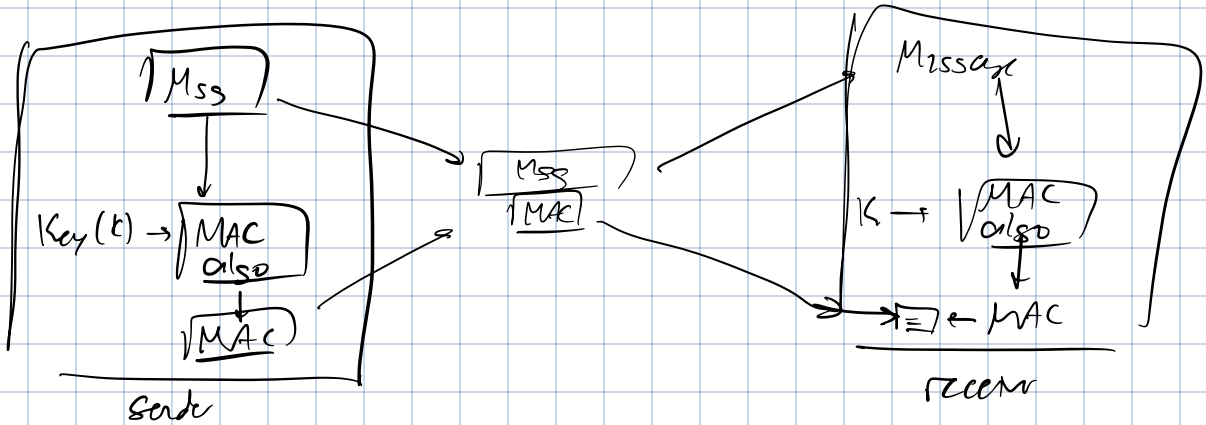
① Message integrity: msg not altered in transit

② Message orgn auth: msg sent by Alice

③ Non-repudiation: Alice cannot claim she didn't send msg.

MACs & digital signatures

MAC:



If $MAC_{sender} = MAC_{receiver}$ → integrity ✓

MAC security goals:

① Total break: forger can produce MAC as if he has key → key leaked

② Selective forgery: can forge MAC on single msg given to adversary

③ Existential " : " of adversary's choosing

Attack types

Mathematical analysis
- Uses algo + plaintext + ciphertext to find key

Brute-force
Tries every possible key until intelligent translation found

n-bit security: attacker needs 2^n operations to compromise security
- Ex:// Takes 1M ops to break security.
  - $\log_2 1000000 \sim 20 \rightarrow$ 20-bit security
  - Upper-bound is brute-force ciphers. Upper bound != upper-bound from key size
  - Doesn't account for time/mem. of operations

Proving system is secure:
1. Provable: exists math proof that shows breaking security $\in$ NP
2. Heuristic: evidence of failed attempts from experts

Randomness

Process that gen. random keys should:
- A: Uniform distr.
- B: Independence (S_1 cannot be deduced by S_2 from process, or v.v.)

Entropy: uncertainty observer has over possible outcomes from source
- Expected amt. of info provided by obs. from process
- Formula to calculate entropy of distr.

$$H(M) = -\sum_{n \text{ outcomes}} p(m_i) \log_2 p(m_i) \quad \text{bits}$$

($p(m_i)$: prob. of outcome)

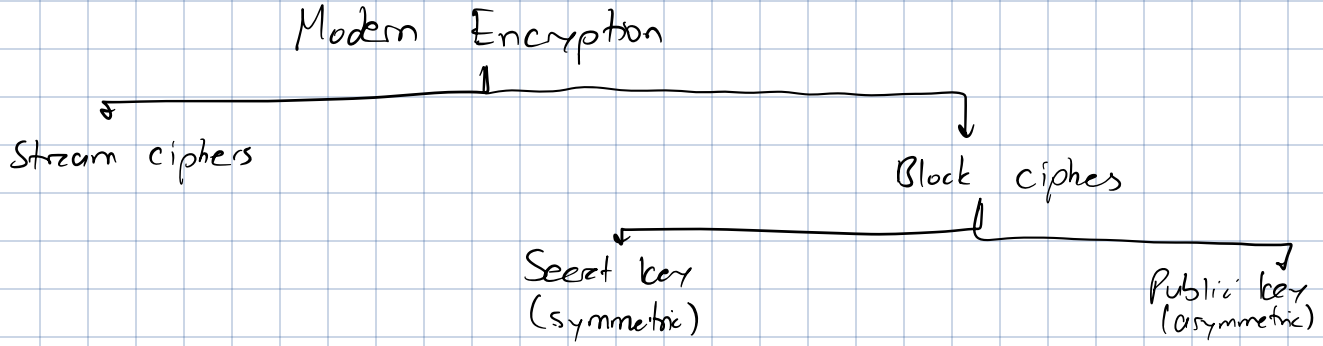
→ $\max(H(M))$ if $M \sim$ Uniform

Tests exist to det. if distr. $\sim$ Uniform, no tests for indep.
- Tests pass: cannot guarantee randomness

↳ Tests fail: discard randomness

SYMMETRIC ENCRYPTION

Classification



Stream Ciphers

Process:

① Divide message into submessages

$$m = m_1 m_2 \ldots m_n$$

② Encrypt m_i w/ corresponding key symbol k_i from keystream (ideally infinite + random)

$$k = k_1 k_2 \ldots k_n k_{n+1} \ldots$$

③ Combine:

$$E_k(M) = E_{k_1}(m_1)\, E_{k_2}(m_2) \ldots$$

Ex:// Vernam cipher

$$E(M) = M \oplus K = m_1 \oplus k_1,\ m_2 \oplus k_2,\ \ldots\ m_n \oplus k_n$$

$$D(M) = M = E(M) \oplus K$$

Shannon showed this cipher is uncond. secure if K is:

① Random

② Used once

③ len(K) ≥ len(M)

For words: ASCII → hex → ciphe

PRNG: uses entropy sources to create unpredictable bits

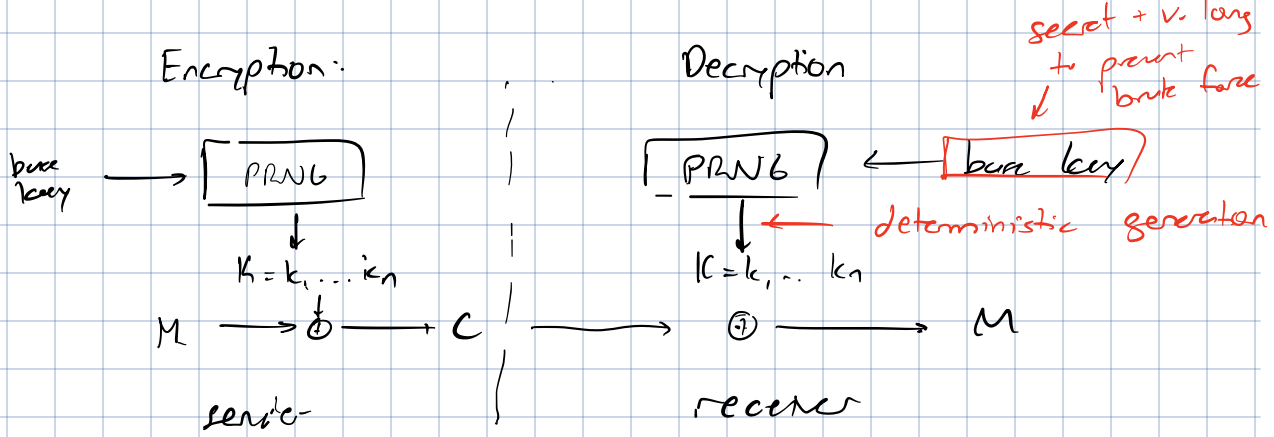
↓

sensors, I/O, network/disk activity, user activity, ...

Stream Ciphers - Keystream

Problem: true random keystream gen. from PRNG is not practical

Solution: create pseudo random keystream via pseudo-random num. gen. (PRNG) w/ a base key (nonce)



Desirable keystream properties:

① Long period ($\sim 10^{50}$)

② Uniform distr.

③ Indep./unpredictability: knowing k_i will not give clue on k_{i+1}

Keystream measured by linear complexity (LC):

↳ LC: # of bits needed to predict remainder of keystream

↳ If I calculate & 22 bits known, remainder of keystream can be predicted

↳ Obj: max (LC)

Stream ciphers - cryptographic PRNG

Many PRNG's specifically designed as keystream generators

① LFSR (linear feedback shift register)

Process:

① Generate next part of seed by XORing taps bits

② Left shift to get S_i

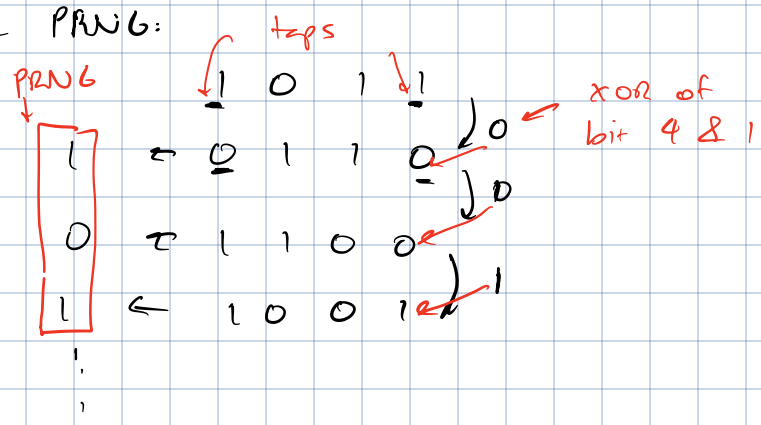
③ Create new seed via output of ① & left-shifted key

Period: $T = 2^n - 1$

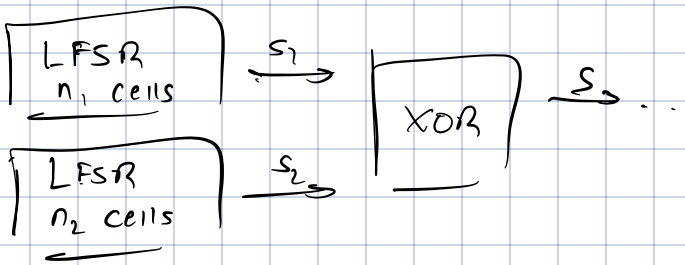
Ex: // Seed = 1011

$f(x) = x^4 + x + 1$ (taps are bit 4 & bit 1)

Generic PRNG:



Long period by low LC. Can improve by doing linear ops. on PRN sequences or non-linear ops (eg. AND)



Stream ciphers - pros & cons

Pros:

① High encryption rates ← character by character or bit by bit transformation.
↳ i.e. faster & less mem. footprint

② Error resistance: b/c encrypting bit-bit, 1 error will not cause issues w entirety of encrypted msg.

Cons:

① Poor info diffusion:

We want 1 change in plaintext should affect entire ciphertext, but bit-by-bit method is not good for this

② Keystreams are not purely random

③ Key reuse

↳ Plaintext attack: knowing M & C yields K:

$$K = M \oplus C$$

↳ Ciphertext attack: if C_i & C_j, M_j predictable:

$$C_i \oplus C_j = M_i \oplus K \oplus M_j \oplus K$$
$$= M_i \oplus M_j$$

Block ciphers

Process:

① M is divided into sub-M in blocks of equal length:

$$M = M_1 M_2 \ldots M_n$$

↳ 64, 128, 256 bits

② Each block encrypted w/ same key

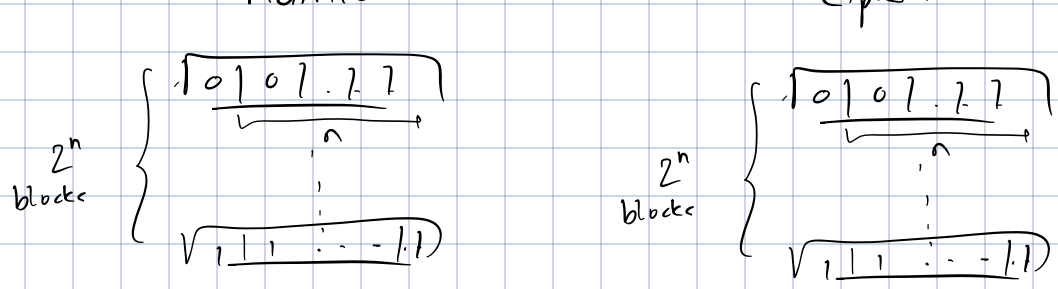
$$C = E_k(M) = E_k(M_1)\, E_k(M_2) \ldots E_k(M_n)$$

Same process to decrypt

Q: How many keys could we have in an ideal scenario?

Keys are effectively mapping plaintext ↔ ciphertext, so count mappings.

Assume 2^n bits in cleartext to 2^n bits to ciphertext (n being # of bits in block).



# of mappings: $\frac{2^n}{1^{st}\text{ block}} \cdot \frac{2^n - 1}{2^{nd}\text{ block}} \cdot \frac{2^n - 2}{3^{rd}\text{ block}} \ldots$

$$= \sqrt{2^n!}$$

This is not ideal, bc key is the substitution table, which is $2^n \cdot n$ length!

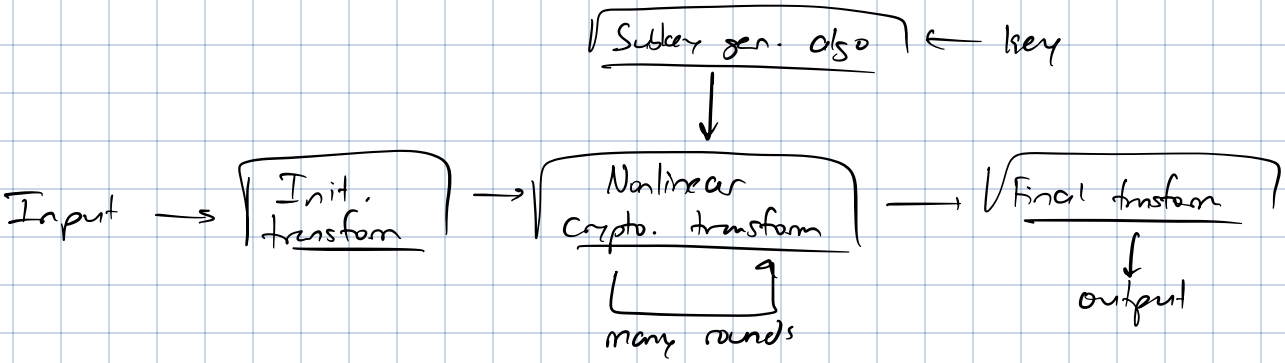
We need to reduce the # of possible keys

We also want:

① High diffusion: stat. struct. of M gone in C
- ↳ each C bit affected by many M bits
- ↳ Achieved via permuting $M_i \rightarrow$ encrypt

② High confusion: C ↔ k relationship should be complicated
- ↳ Achieved by substitution algo.

Overall scheme:



Nonlinear cryptographic transform options:

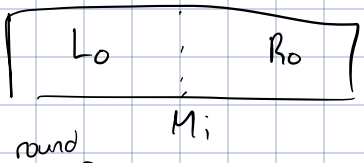
① Feistel scheme

② Substitution-perm. network scheme.

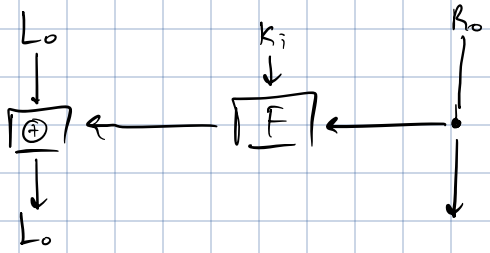
Block ciphers - Feistel Scheme

Encryption process:

① Divide M_i into 2 halves: L_0 & R_0



② Apply non-linear round function F to R_0 & XOR output w L_0 to get new L_0. Usually needs some key



③ Permute 2 halves

④ Repeat for n rounds

Decryption process: same circuit but uses subkey in reverse order & 1 final permutation

This requires:

① Good subkey gen. also

② Develop good round function F

Properties:

Ⓐ Block size: block size ↑ → security ↑, speed ↓

↳ Typically 64 bits or more

Ⓑ Key size: key size ↑ → security ↑, speed ↓

↳ Typically 128 bits ≤

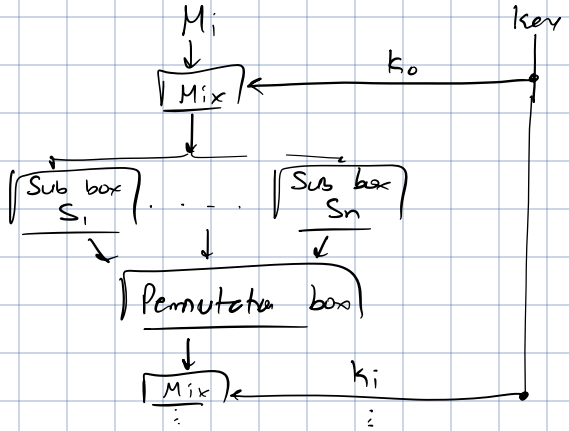
Ⓒ Number of rounds: n ↑ → security ↑, speed ↓

↳ Typically n = 16

Ⓓ Subkey gen. also + F: more complex → greater resistance to cryptanalysis

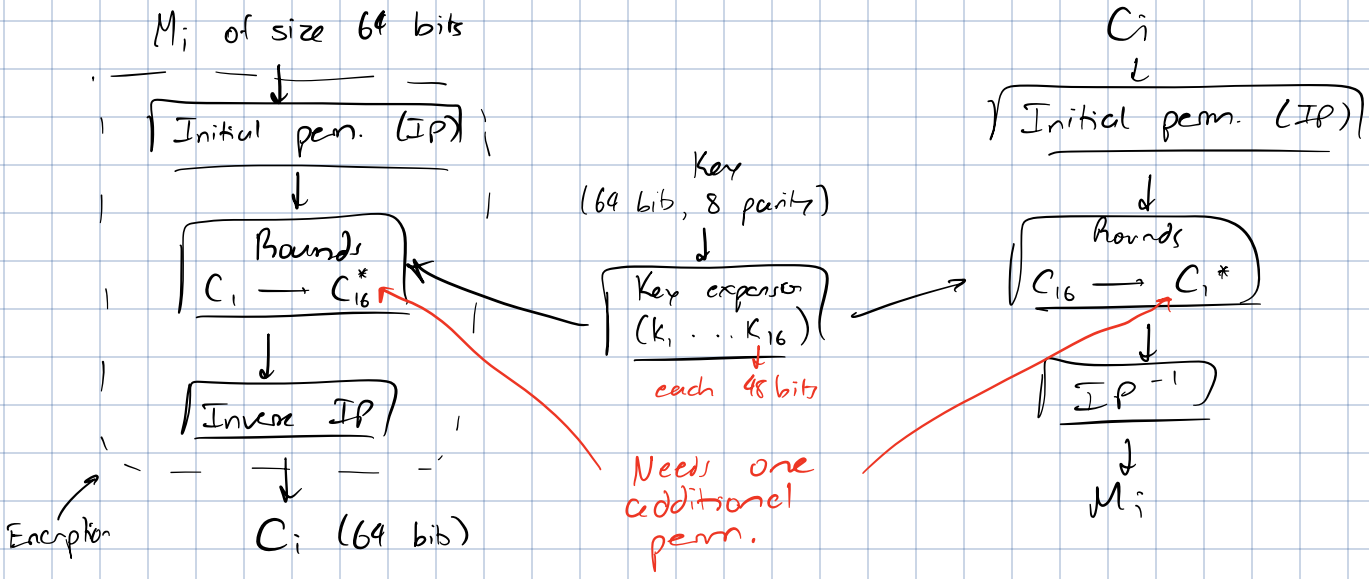
Block ciphers - substitution permutation network (SPN) scheme

Encryption process:

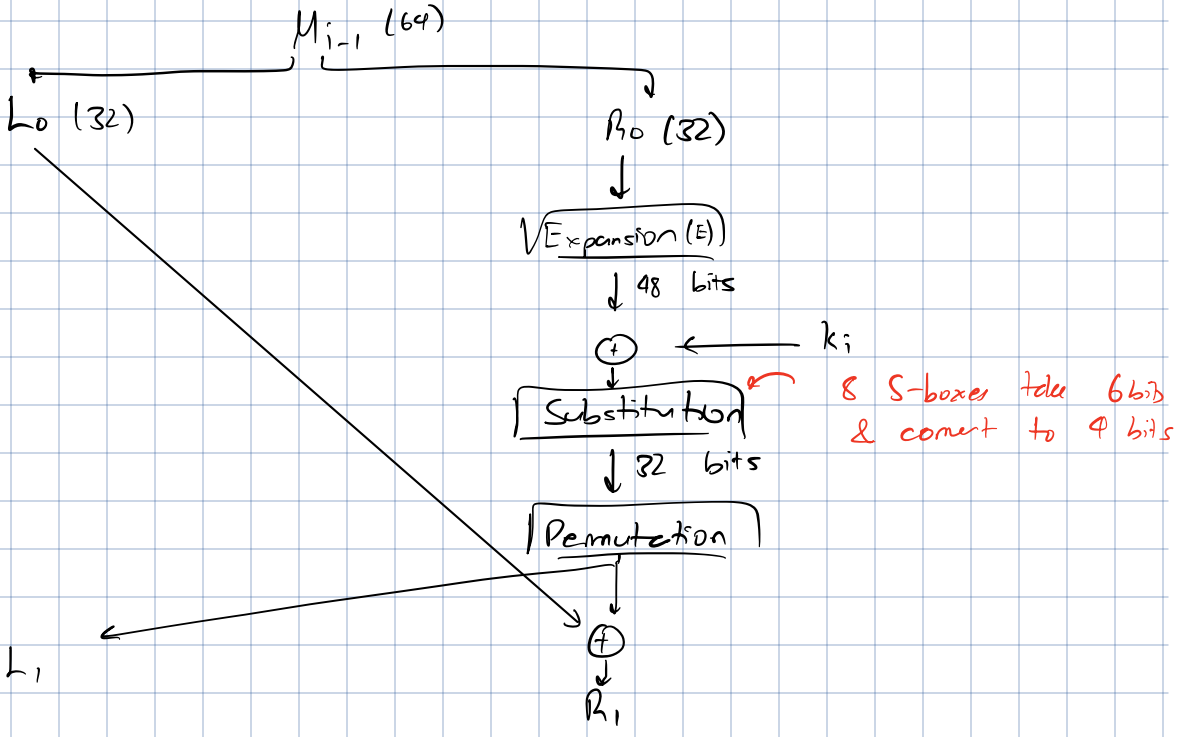


Block ciphers - Data Encryption Standard (DES)

Process:



Round scheme: Feistel!



Improvement: 3DES

↳ Apply DES 3 times in series w/ different keys

↳ If 2 keys: 112 bit length. 3 keys → 168 bit length

↳ Much slower but stronger

Vulnerable to brute force attacks

Block ciphers - Advanced Encryption Standard (AES)

Standard block cipher

Good security, cheap & eff. to impl., flexible & simple

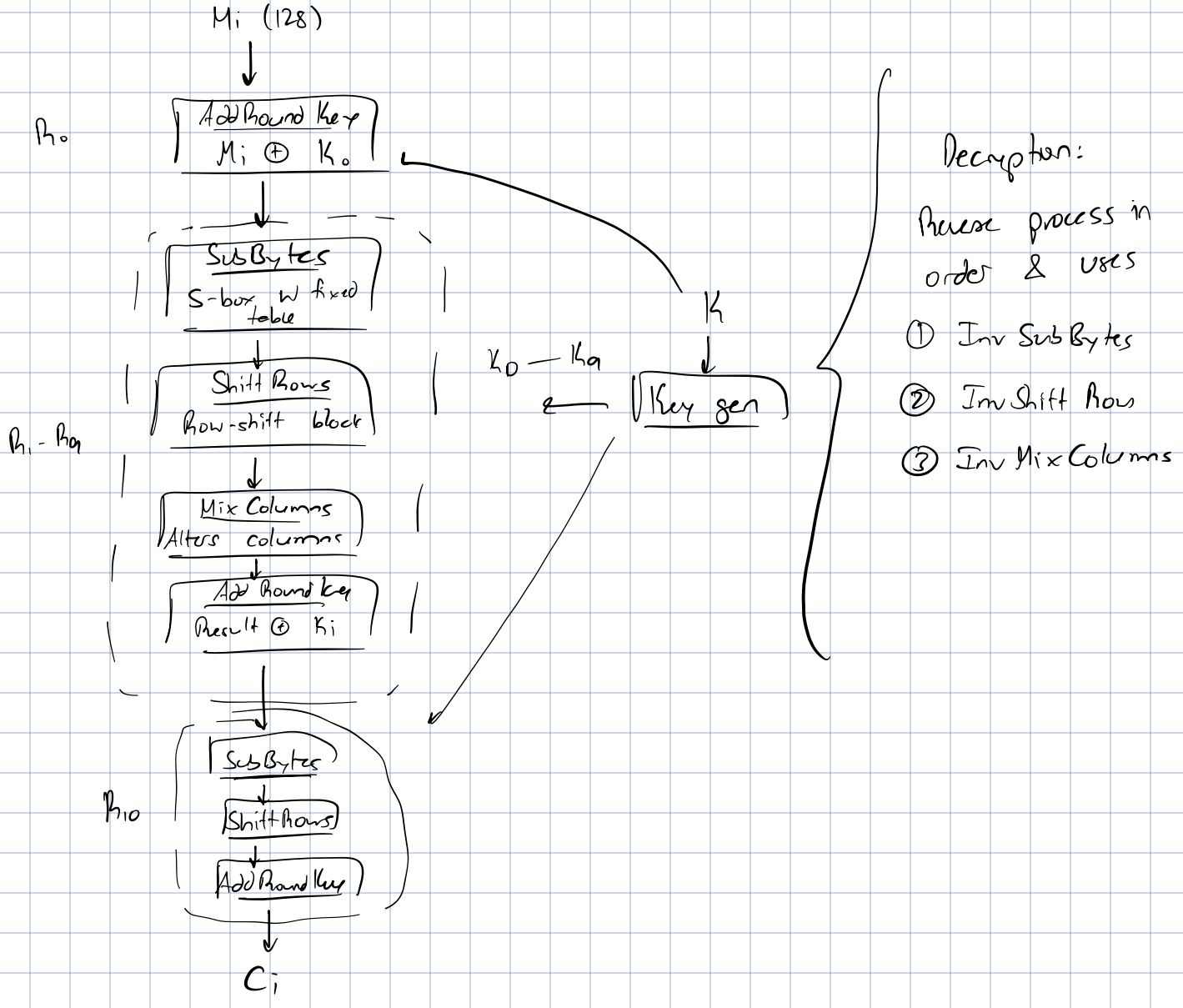
Properties:

① Block size: 128 bits

② Key size: 128, 192 or 256 bits

③ Round scheme: SPN

Scheme:

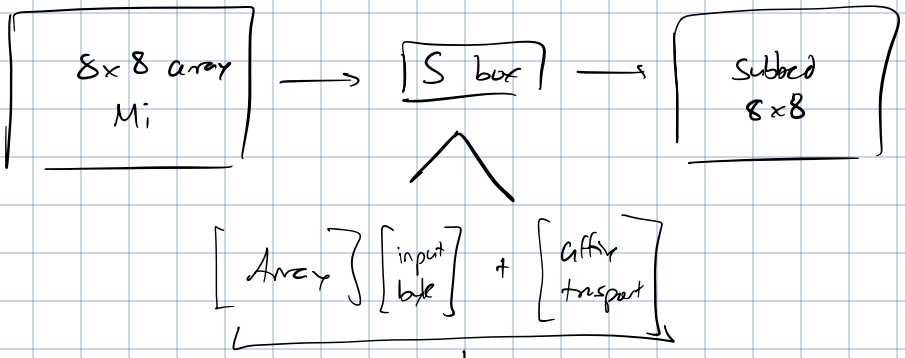


Keys & blocks are in arrays (hence mention of "rows" & "cols")

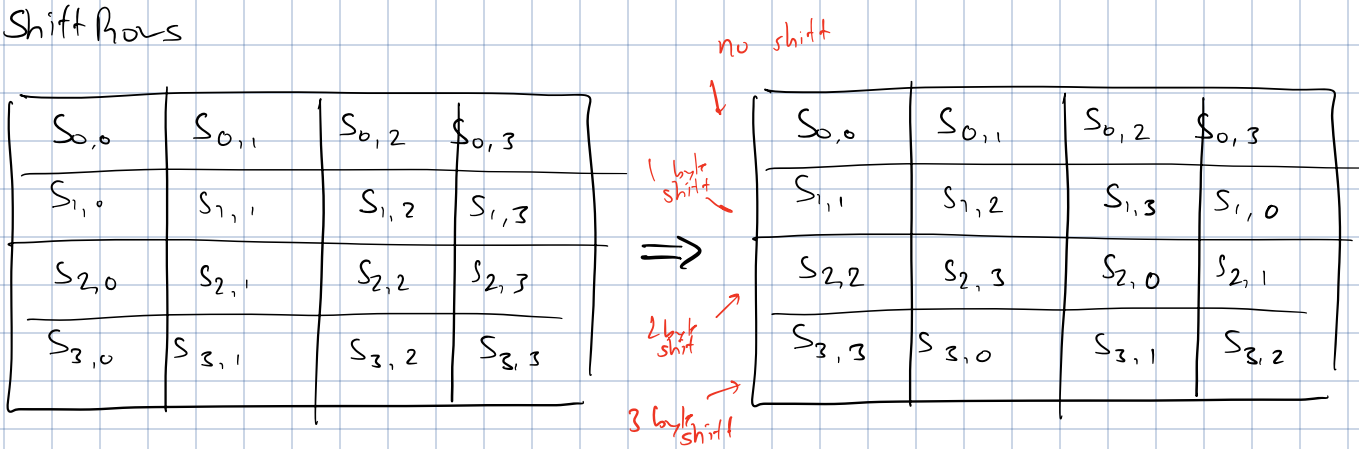
In Galois Field of $GF(2^8)$ w/ $p(x) = x^8 + x^4 + x^3 + x + 1$

Going through each function:

① SubBytes:



② ShiftRows



$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$

$\Rightarrow$

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,1}$	$S_{1,2}$	$S_{1,3}$	$S_{1,0}$
$S_{2,2}$	$S_{2,3}$	$S_{2,0}$	$S_{2,1}$
$S_{3,3}$	$S_{3,0}$	$S_{3,1}$	$S_{3,2}$

③ Mix Columns

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix}$$

$03 \rightarrow x+1$
$02 \rightarrow x$
$01 \rightarrow 1$

Multiply like matrix but add is actually ⊕

Ex:// Assume column is $[el \quad fb \quad 96 \quad 7c]^T$. Find output of MixColumns

$s'_{0,c} = \{02\}(el) \oplus \{03\} fb \oplus 96 \oplus 7c$

$\downarrow$

$x(x^7 + x^6 + x^5 + 1) = x^8 + x^7 + x^6 + x$

BUT mod: $(x^8 + x^7 + x^6 + x) \bmod p(x) = \boxed{d9}$

Do this for all 4 XOR

④ AddRoundKey

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$

⇒

$S_{0,0} \oplus K_0$	$S_{0,1} \oplus K_4$	$S_{0,2}$	$S_{0,3} \oplus K_{12}$
$S_{1,0} \oplus K_1$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$
$S_{3,0} \oplus K_3$	$S_{3,1}$	$S_{3,2}$	$S_{3,3} \oplus K_{15}$

Much more secure against brute-force attacks

Block ciphers - pros & cons

Pros:

① Encryption & decryption are almost the same

② High diffusion

③ Efficient

Cons:

① Slower than stream

② Error propagation 1/in block

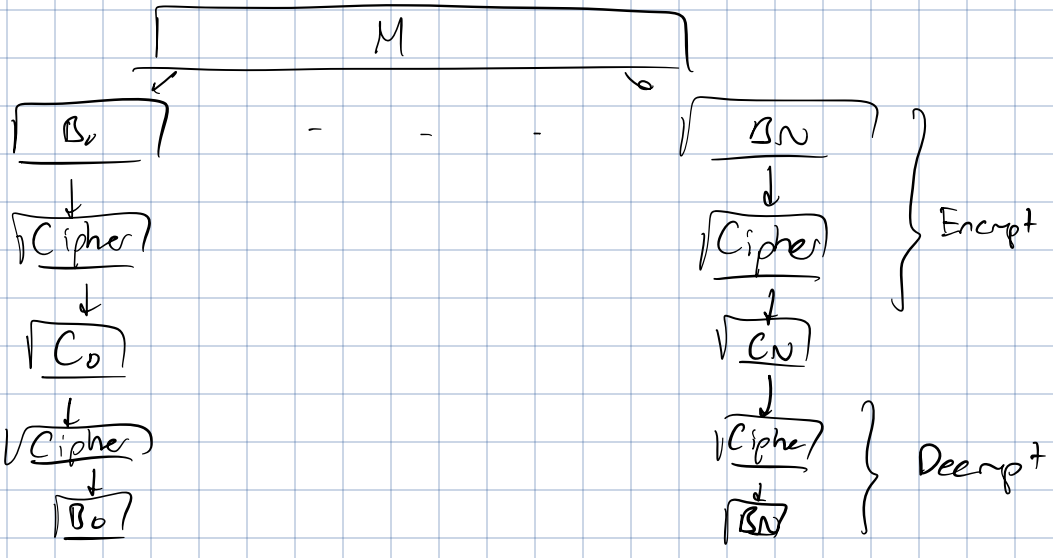
③ Padding to ensure M is multiple of block size

④ Needs operation modes to mess w/ blocks.

AES, Camellia & Serpent are secure block ciphers

Block ciphers - operation modes

① Electronic code book (ECB)



Pros:

① Parallelization

② Bit error in transmission only affect 1 block

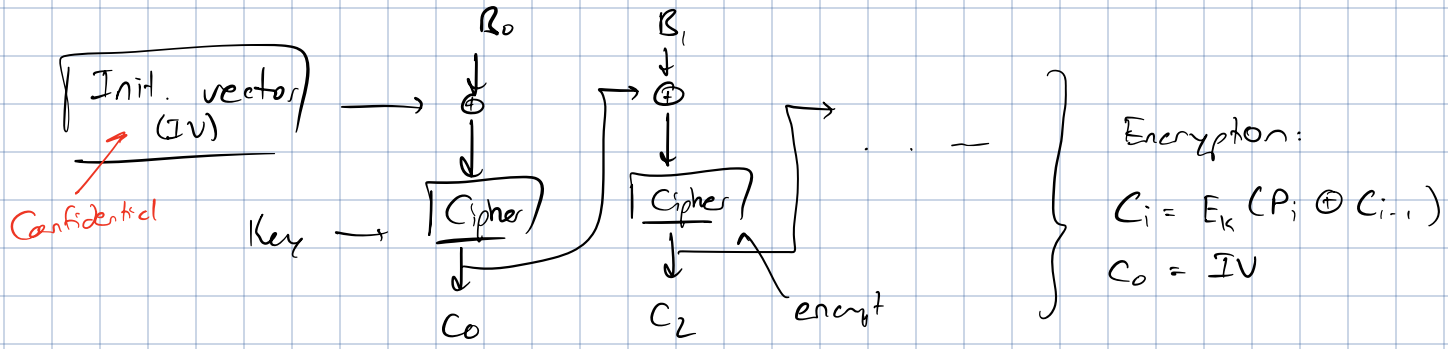
Cons:

① Repeated $B_i \rightarrow$ same C_i

② Can modify block order / eliminate

③ Last block need padding

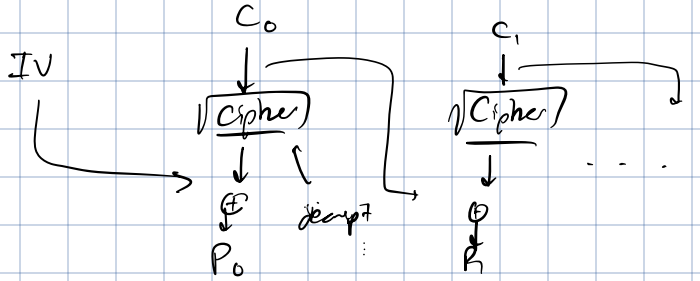
② Cipher Block Chaining (CBC)



Encryption:

$C_i = E_K(P_i \oplus C_{i-1})$

$C_0 = IV$



Decryption

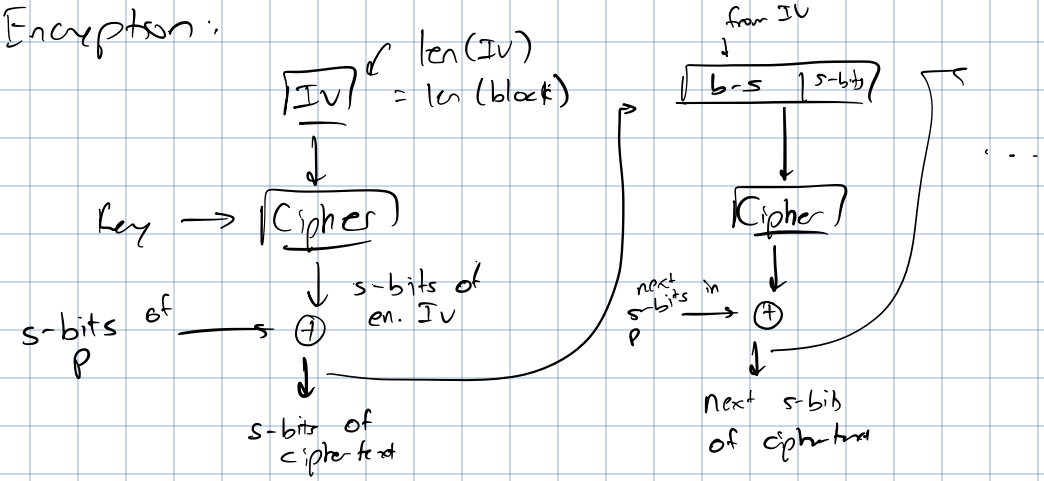
$P_i = D_K(C_i) \oplus C_{i-1}$

$C_0 = IV$

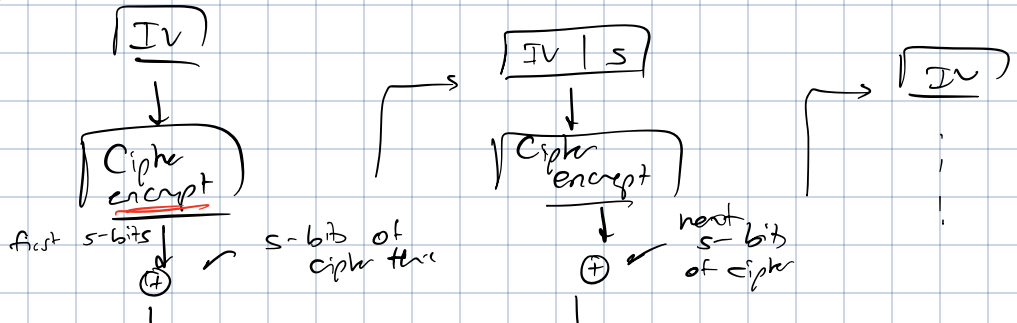
A bit error will affect 2 blocks. Padding is needed

③ Cipher Feedback (CFB)

Encryption:



Decryption:



s bits of plaintext

s bits of plaintext

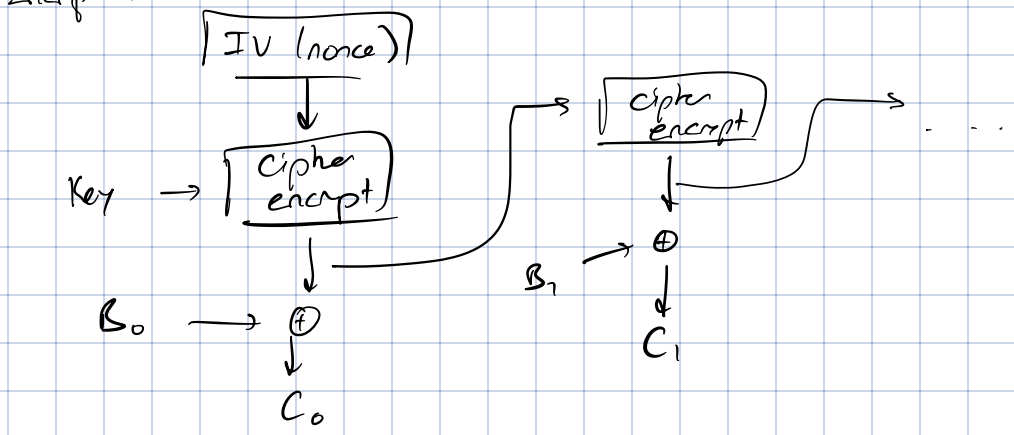
A bit error in transmission affects 2 M_i.

Kinda transforms block cipher into stream cipher since it can encode data < block size

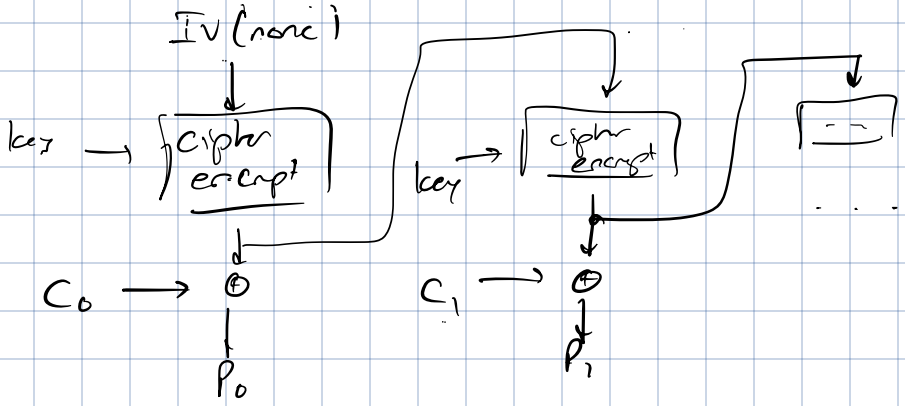
↳ Keystream is influenced by plaintext tho

④ Output Feedback (OFB)

Encryption:



Decryption:



Bit errors only affect single bit of M_i.

Converts block cipher to stream cipher but:

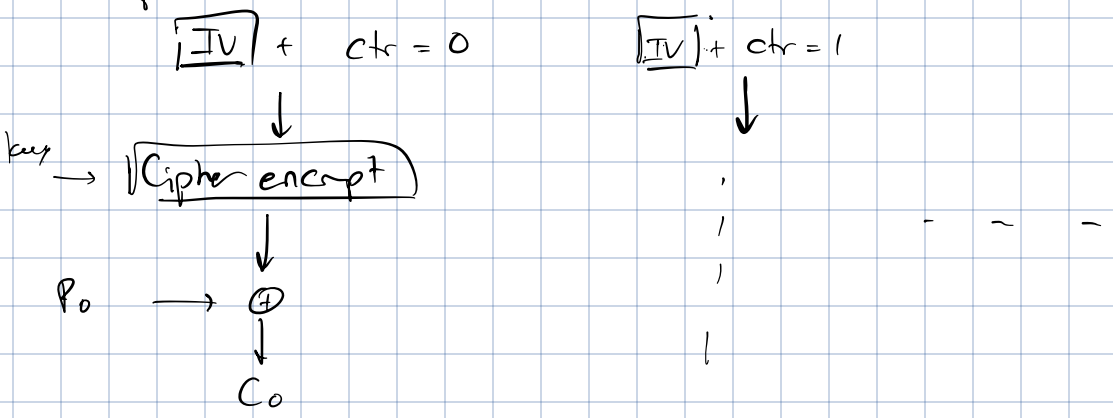
Ⓐ Keystream doesn't depend on plaintext

Ⓑ Works on blocks, not segments

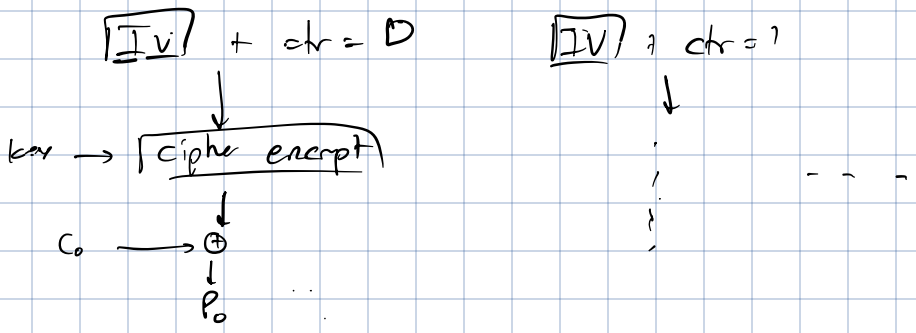
Ⓒ Need to decrypt $C_0 \rightarrow C_{i-1}$ to decrypt C_i

⑤ Counter (CTR)

Encryption:



Decryption:



Bit error only affects 1 bit

Parallelizable

Converts block cipher → stream:

① keystream ≠ plaintext

② Works on blocks, not segments

③ $D(C_i)$ does not need $D(C_0) \rightarrow D(C_{i-1})$

Key Distribution And Asymmetric Encryption

Asymmetric Encryption - Historical Context & Impact

Symmetric requires both party knowing same key, but this is point of failure

Until 1976, no one knew how to pass secret messages w/out same key

British Intelligence found a way in 1976 & proposed Diffie-Hellman Key Exchange also.

Asymmetric Encryption – Model

User key pair composed of:

① Public key

Known by everyone

Sender uses receiver public key to encrypt msgs to receiver

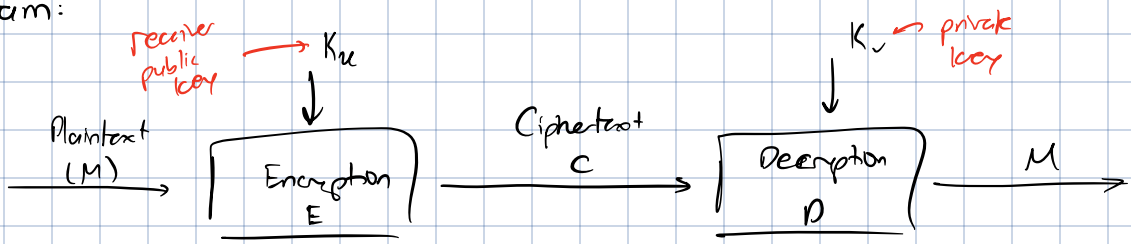
② Private key

Known only by owner

Receiver uses private key to decrypt msgs

Unfeasible to derive private key from public

Diagram:



Pros:

Possible to brute force, but much harder

Cons:

Slower than symmetric algos

Asymmetric Encryption – RSA

Published 1978

Uses difficulty of factoring integer into 2 large primes.

Process:

① Receiver (B) key pair gen.

1. Choose 2 large primes (kept private): p_B, q_B
2. Compute: $n_B = p_B \cdot q_B$

3. Compute: $\Phi(n_B) = \Phi(p_B) \cdot \Phi(q_B)$
$= (p_B - 1)(q_B - 1)$

4. Choose e_B: $1 < e < \Phi(n_B)$ & $\gcd(e, \Phi(n_B)) = 1$
Smaller than $\Phi(n_B)$ — Coprime w/ $\Phi(n_B)$

5. Compute d_B:
$d_B \cdot e_B \bmod \Phi(n_B) = 1$

6. Construct public & private keys:
$K_{u,B} = (e_B, n_B)$ $K_{u,B} = (d_B, n_B)$

② Sender sends message encrypted:

Recall sender (A) know $K_{u,B}$. Thus:

$C = M^{e_B} \bmod n_B$

③ Decryption:

$M = C^{d_B} \bmod n_B$

Hard to break b/c you need d_B, which is $d_B = \text{inv}[e, \Phi(n)]$, which requires knowing p_B & q_B

Asymmetric encryption - RSA attacks

Textbook RSA is deterministic → not IND-CPA secure

↳ Produce m_0 & m_1. Encrypt one: $c^* = m_B^{\,e} \bmod n$. We can then encrypt both m_0 & m_1 and see if c^* matches either

Textbook RSA is malleable

↳ If $C \rightarrow M$, can obtain $C' \rightarrow M'$ where $M' \sim M$.

How? Let α be some Z. We know $C = M^e \bmod n$. Then let $C' = C \cdot \alpha^e \bmod n$. $C' \rightarrow M' = \alpha \cdot M$

Asymmetric encryption - RSA-OAEP

Process:

① Message padding:

$X = M \oplus G(r)$
$Y = r \oplus H(X)$

} r = random string, M = plaintext msg, G, H = crypto. func.

③ Create new message

$M' = X \,||^{\text{concat.}}\, Y$

④ Encrypt:

$C = (M')^e \bmod n$

⑤ Decrypt:

$M' = C^d \bmod n$

↳ $M' = X \,||\, Y$ ✓

$r = Y \oplus H(x)$

$M = X \oplus G(r)$

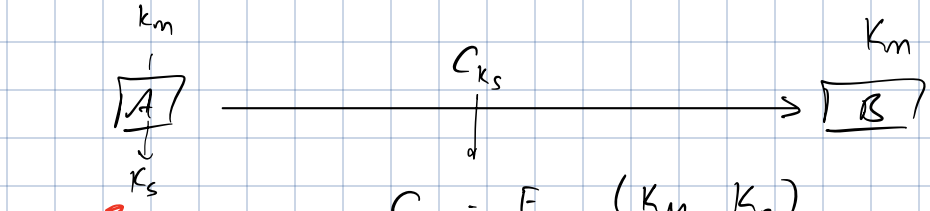
Pretty secure!

Key distribution – symmetric key distr. via sym. crypto.

① Key wrapping

Idea: what if A & B already share a key → encrypt a new key to share w/ other party

Diagram



A chooses K_s

$C_{K_s} = E_{SYM}(K_M, K_s)$

↳ Encrypted K_s

② Key hierarchy via key distr. centre (KDC)

Idea: A & B have secure channel w/ C, who chooses key & shares w/ A + B

Process:

1. Each user shares master key K_x w/ KDC
2. KDC creates 1-time session keys K_s & shares via encrypted channel
3. A & B decrypts message w/ K_s

If n users:

- $\frac{n(n-1)}{2}$ session keys
- n master keys

Key distribution - symmetric key distr. via asym. crypto

Use asymmetric cryptography to distribute keys but use symmetric cryptography to encrypt data

Known as hybrid encryption or KEM (key encap. mechanism, asym.) / DEM (data encaps. mech., symmetric)

Process:

1. $M \rightarrow E_{K_s}(M)$ where K_s randomly gen.
2. K_s asym. encrypted using receiver public key $K_{u,B}$.
3. Receiver $E(K_s) \rightarrow K_s$ via private key
4. $E_{K_s}(M) \rightarrow M$ via decrypted K_s

RSA - KEM process

1. A chooses $x \in \{1, \ldots, n_B - 1\}$
2. Generate symmetric key via $k = H(x)$, where H is some hash function
3. Generate $c = x^{e_B} \bmod n_B$ & send to B
4. B determines x via decryption & can derive $k = H(x)$

Distribution of public keys

1. Public announcement
2. Public directory

   Use trusted 3P to store public keys. Requires registration & secure access.

③ Public key authority

Like directory but w/ more control mechanisms

Participants need public key of public authority & real-time access

④ Public key certificaton authorities

Process:

1. Generation: CA creates public key cert. via requestor's identity & public key

2. When sending public keys, receiver can receive cert. & check via CA public key to see if genuine

Pros:

1. Can exchange keys w/out CA being online since you only need CA public key to verify certs.

2. CA is scalable

3. Certs. are revokable

Diffie-Hellman Protocol for Key Exchange

Problem: share a secret over an insecure channel (like a key)

Idea: use public key cryptography (asym. crypto.)

Process:

① A chooses prime p & base g, a primitive root mod p

② A chooses random private key x_A & B does same (x_B) s.t. $x_A, x_B \in \{1, \ldots, p-1\}$

③ A creates public key $y_a = g^{x_A} \bmod p$

④ A sends p, g & y_a to B

⑤ B chooses b & public key $y_b = g^{x_B} \bmod p$ & sends to A

⑥ Symmetric key created:

A: $s = y_b^{x_A} \bmod p$ B: $s = y_a^{x_B} \bmod p$

Security:

① Computing a/b knowing only y_a / y_b is v. hard (discrete log problem)

② Computing s knowing only y_a & y_b is v. hard (Diffie-Hellman problem)

In practice:

① p & g standardized & everyone knows

② s is not initial key. Other sym. keys generated using s (eg. $s' = H(s)$)

Vulnerabilities:

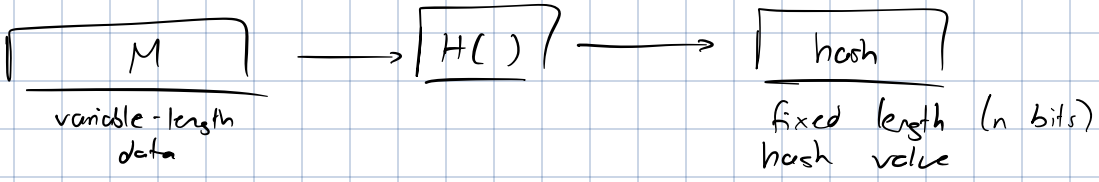
① No auth

② Vulnerable to person in middle attack

} Soln: auth.

Hash Functions, MAC, Authenticated Encryption

Hash Functions: Intro



Note that collisions can happen since hash space is finite: $H(M_1) = H(M_2)$

Cryptographic Hash Functions

Hash function w/ following requirements:

① Must work w/ msg. of any size

② Must compute fixed-size hash values

} Compression

③ Output must satisfy pseudo-randomness req.

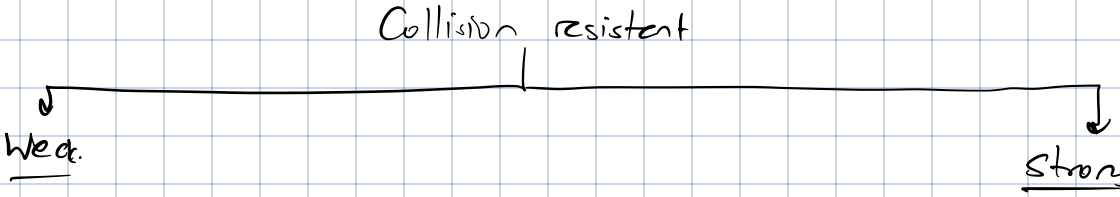
Additional items:

↳ Diffusion: if 1 bit in M changes, then half of bits of $H(M)$ should change

↳ Determinism: multiple runs of $H(M)$ should give same value

↳ Efficiency: hash calc. should be fast

One-way property: for given h, it should be comp. difficult to find M s.t. $H(M) = h$



Weak.

For any msg. M, it is comp. unfeasible to find $M' \neq M$ s.t. $H(M') \neq H(M)$

Strong

It is comp. difficult to find 2 msgs. M & M' s.t. $H(M) = H(M')$

What does it mean to be comp. unfeasible?

↳ No better algo. than brute force to produce collisions

Strength of hash function depends on:

① Design: only brute force attack available

② Hash length: should be long

Collision probabilities:

① One-way collision attack: $1/2^n$

② Weak " : $1/2^n$

③ Strong " : $\frac{1}{2^{n/2}}$ ⇒ birthday paradox

} We need this to be infeasible

A "broken" hash func. is one where an algorithm can produce a collision in less than brute-force time.

Recommended bit length: 256

Hash function examples

Merkle-Damgård construction:

① Message preparation

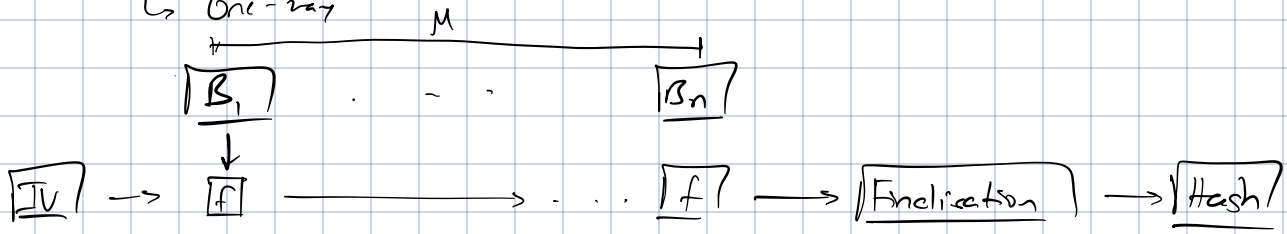
i) Message is divided into L blocks of length B

ii) Message length appended to last block

iii) Padding applied

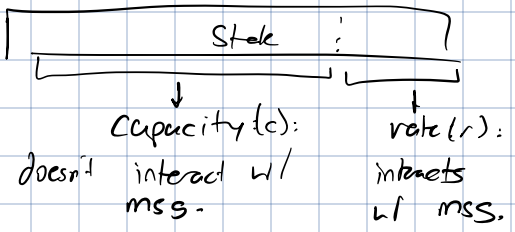
② Select an initial value as the "seed"

③ Compression: take prev. value (IV if never done) & corresponding block & create hash of length n. Repeat for all blocks
↳ Needs to be collision resistant → hash func. will be.
↳ One-way



Sponge construction: basis of SHA-3

① State initialization

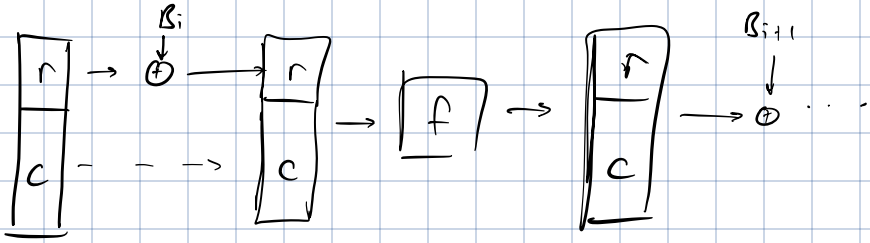


② Absorption

i) Msg is padded & divided into blocks of size r

ii) Each block is XOR'ed w/ rate part

iii) New state constructed using fixed, invertible function (not 1-way)

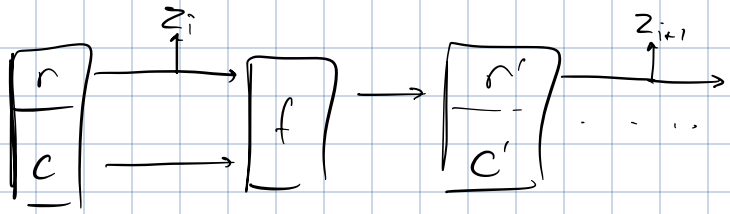


③ Squeezing

i) Output block takes rate part of state

ii) r & c put through f to get new state

iii) concatenate output blocks for final hash



Other hash functions:

① MD5: weaknesses discoed in 1996

② SHA-0: broken in 2005

③ SHA-1: "

④ SHA-224, 256, 384, 512: alright soln.

⑤ SHA-3: new

MAC Overview

Uses secret key to compute fixed-length value (auth code) from var. length mss. ↳ synchronous

If entity has secret key, it can verify message integrity w MAC (authentication)

MAC functions do not need to be reversible, which means that you cannot use MAC(M) to find M

Collisions are possible

MAC security requirements

① Should be infeasible to determine a M w/ same MAC value as given

② Uniform distn: P(find M & M' w same MAC) = $\frac{1}{2^n}$

③ $P(MAC(K, M) = MAC(K, M')) = \frac{1}{2^n}$

same ↓ transform from M

Potential attacks: given M_i & $MAC(K, M_i)$, attacker wants to generate $(M', MAC(K, M'))$ s.t. $M' \neq M$ $\forall i$, $0 \leq i \leq n$

① Brute force attack: $\min(2^k, 2^n)$ — 2^k: key space attack; 2^n: MAC attack

② Cryptanalysis attack: looks at algo vulnerabilities

Hash-based MAC

Formula: $HMAC(K, M) = H[(K \oplus opad) \,||\, (k \oplus ipad) \,||\, M]$

- padded with 0's until $|k| = b$
- hash function
- 0x36 repeated b/8 times
- 0x5C repeated b/8 times
- concatenation op.

Block cipher-based MAC

Process:

① Divide M into blocks

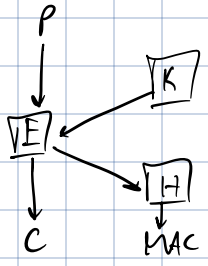
② Encrypt each block via block cipher using K w/ CBC mode of IV=0

③ Last ciphertext block is MAC, but uses diff. K

Authenticated encryption

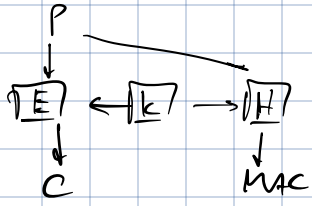
Schemes

Encrypt-then-MAC



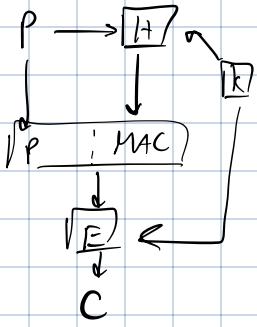
High-security!

Encrypt & MAC



Can lead to case of same text → same MAC (collision)

MAC → Encrypt



Can use diff. keys in MAC vs. encryption to prevent vulnerabilities

Examples:

① Galois Counter Mode: encrypt-then MAC

② AEAD: Auth. Encryption w/ Associated Data

Digital Signatures

Digital signature schemes

Defn: Result of crypto. transform that provides following:

① Source/identity auth.

② Data integrity auth.

③ Support for signed non-repudiation (sender cannot deny that they sent msg.)

Ideally, these signatures are easy + cheap to produce & diff. for each piece of data it is sending

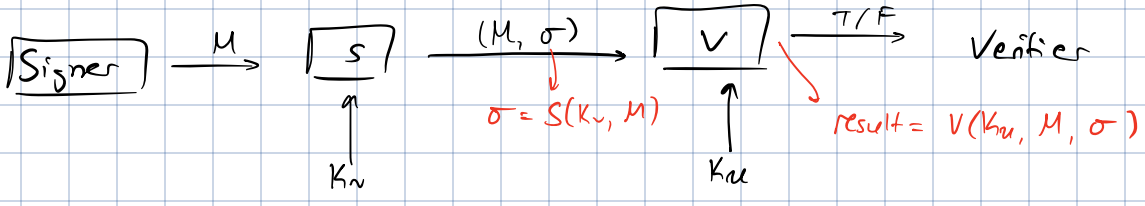
Recognize that this signature doesn't provide confidentiality!

Components:

① Key gen. algo (G) ⇒ creates private key (K_v) (generate sig.) & public key (K_u) (sig. verif.)

② Signature gen. algo (S)

③ " verification algo (V)



Signature schemes

Deterministic

Signature of M_1 & M_2 same if $M_1 = M_2$

Randomized

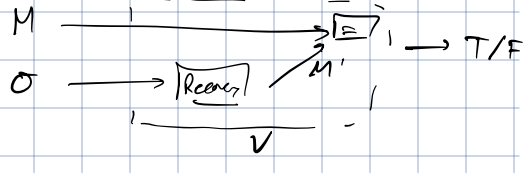
Signature of M_1 & M_2 ≠ same if $M_1 = M_2$ b/c depends on index

Signature verification types

Sep. from msg

Signature is appended to msg.

Messg. Recovery



Signatures usually computed on hash of message, so verification also requires hash of msg.

RSA Digital Signature

Properties:

① Deterministic

② Message recovery

③ Secure: based on problem of integer factorization

Scheme:

A	B
① Generate public & private keys	
② Compute $\sigma = M^{d_A} \bmod n_A$	
M, σ →	
	③ Compute $M' = \sigma^{e_A} \bmod n_A$
	④ Accept authenticity if $M == M'$

ElGamal Digital Signature

Properties:

① Randomized

② W/ appendix

③ Based on discrete log problem

Usually, DSA (digital sig. also.) used, variant of ElGamal

Scheme:

A B

① Compute p_A, g_A, x_a & y_a like D.H.

② Private key:
$k_{v,a} = x_a$

③ Public key:
$k_{M,a} = (p_A, g_A, y_a)$

④ Sign message:

i) Choose k_s, $0 < k_s < p_A$
& $r = g^{k_s} \bmod p_A$

ii) Compute: ($k_s x \equiv 1 \bmod p_A - 1$)
$s = (M - x_A \cdot r) \cdot k_s^{-1} \bmod (p_A - 1)$

$M, (r, s)$
& D.H. public

⑤ Verification

i) $V_1 = y_a^r \cdot r^s \bmod p_A$

ii) $V_2 = g_A^M \bmod p_A$

iii) Accept if $V_1 \equiv V_2$

Attacks

Goal: forge valid signature of fake message

Strength ↓

Total Break
Determine A's privk key

Univocal Forgery
Has signing algo equiv. to A

Selective Forgery
Can forge on message chosen

Existential Forgery
Can forge on at least 1 msg., but no control

RSA Attack 1:

① C knows e_A, n_A

② C chooses $\sigma \in Z_{n_A}$, computes $M = \sigma^{e_A} \bmod n_A$

③ C sends (M, σ). B accepts

RSA Attack 2:

① C gets valid messages & signatures: $(M_1, \sigma_1), (M_2, \sigma_2)$

② C creates: $M' = M_1 \cdot M_2 \bmod n_A$, $\sigma' = \sigma_1 \cdot \sigma_2 \bmod n_A$

③ B verifies σ' on M'

Solution: add padding & randomness (salt) to RSA & sign hash of msg.

For ElGamal, apply signature to hash to avoid similar existential attacks.

Digital Signatures & Encryption

To build a secure channel w/ public key cryptography, you need:

① Public key cryptosystem

② Digital signature scheme

How to combine?

① Sign → encrypt
② Encrypt → sign

} Secure if rcv. & send identities are in sign. & ciphertext.

③ sign & encrypt

④ Signcryption: only B can decrypt message & B can verify A sent it

PUBLIC KEY INFRASTRUCTURES

Problem: public keys not linked w/ identities by default

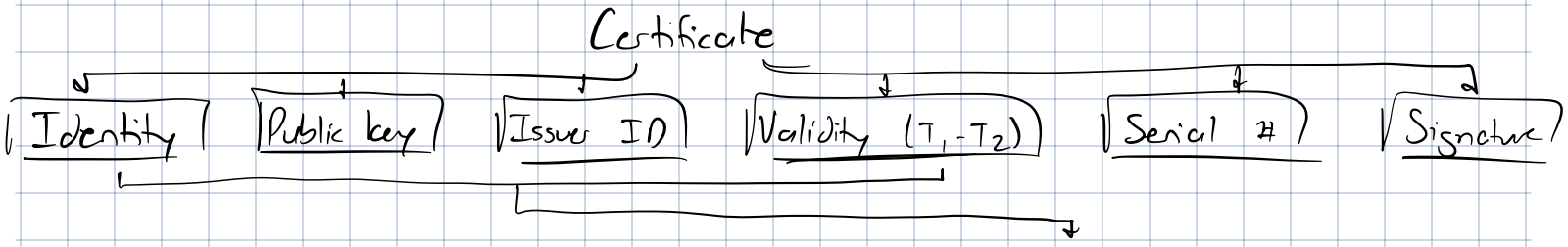
Solutions:

① Public key directories / authorities

Problems: need online access & not scalable

② Digital certificate

Public key certificate



Signature = signing-algo (private key, data to sign)

Process of using public key certificate (A sends M to B)

① B sends certificate (C_B) to A

② A verifies C_B

③ A signs message (for data integrity) & attaches certificate C_A

④ B verifies signature & certificate

How to verify a certificate?

① Get AC (authorization center) public key certificate

② Verify signature of certificate via AC public key

③ Verify time in valid period

④ Verify certificate is still valid

⑤ Verify key supposed to be used for cert. issuing

Public key infrastructure (PKI)

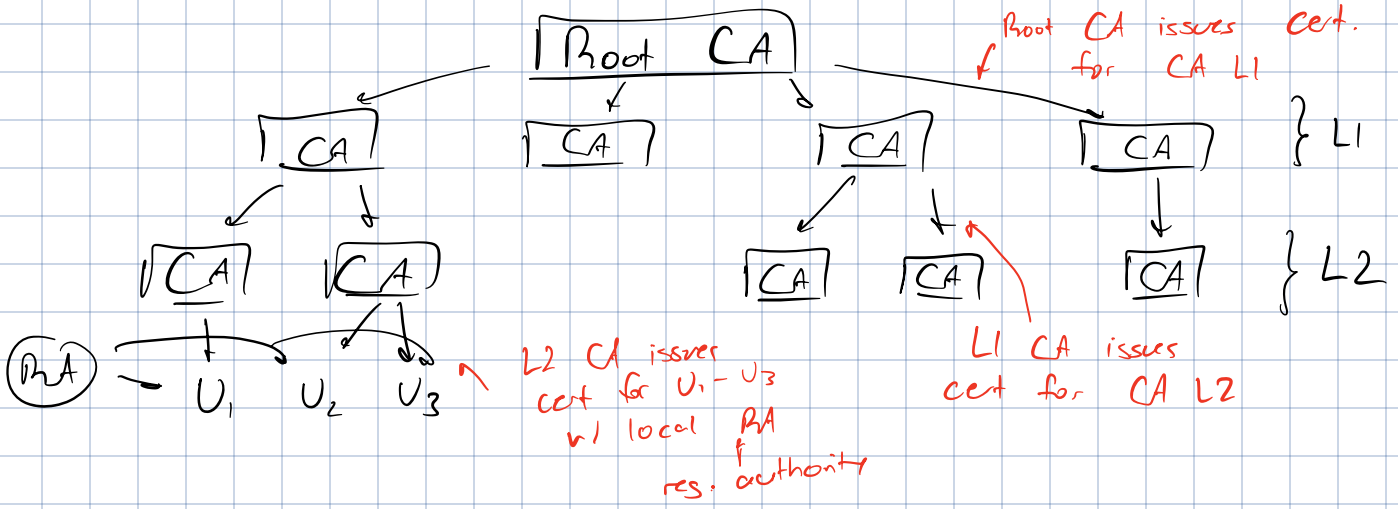
Defn: set of intn. standards that defines

① Structure of X.509 certificates

② Cert. revocation lists

③ Hierarchical model of CAs

Hierarchical model



Certificate chain verification

① Get cert CA

② Verify CA signature by getting next CA's cert, C_{AC2}

③ Verify C_{AC2} via C_{AC1}

④ Continue until hit root authority certificate, where trust decision made

For A to sign message, must concatenate A's certificate & whole certificate chain, which B must verify entirely

X.509 Public Key Certificates

Components

① Version: 3

② Serial #: unique ID of cert in realm of AC

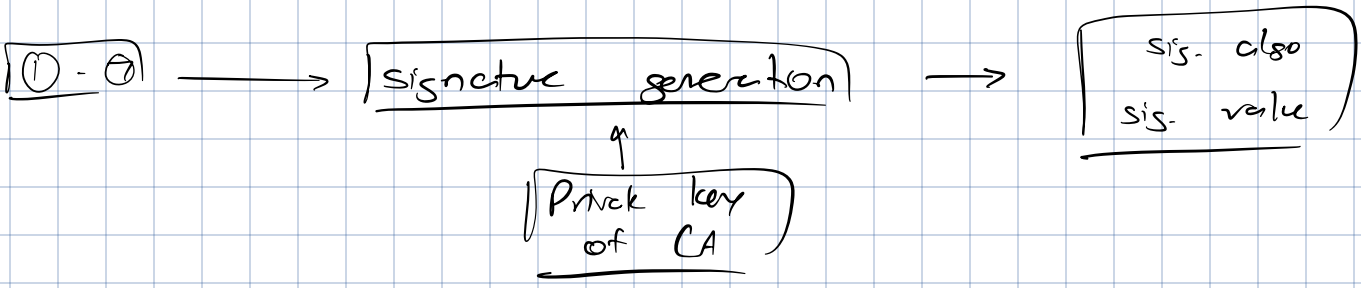
③ Issuer: name of issuing AC

④ Validity period: [not bf., not afte]

⑤ Subject: name of subject owner of cert.

⑥ Public key: info about key, algo

⑦ Extensions: additional info, eg. specify intended key usage

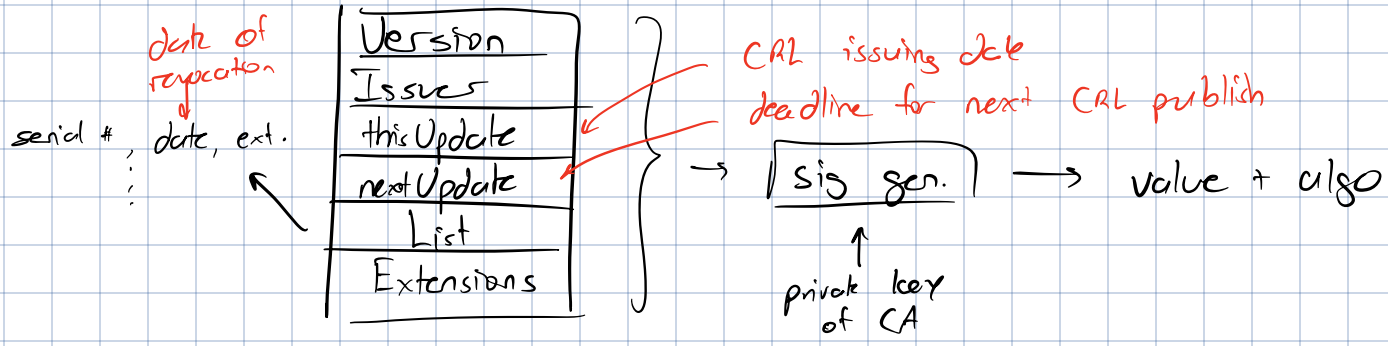


Public Key Cert. State Validation

The state of a cert., if revoked, is published on lists (CRL) that is periodically updated

To query about the state of a cert., use Online Cert. Status Protocol (OCSP)

Structure:



This is housed in AC & does not include expired certs

Other Aspects

① Operational services: cert. request, reg., renewal, revocation, status checks
CRL publication

② Key generation & storage: key authority can generate & keep copy of key pair

③ Certification practice statement: SOP from CA

Decentralized Model

Each user certifies keys of users they trust → trust chains
Quick & cheap to set up, but not scalable & public key must go in secure channel

User Authentication

Defn.: verifying identity of user

Two steps:
1. Identification
2. Verification

Auth factors: secrets, tokens, biometrics or combo

Secret-based authentication

Idea: auth based on only info that user & system knows

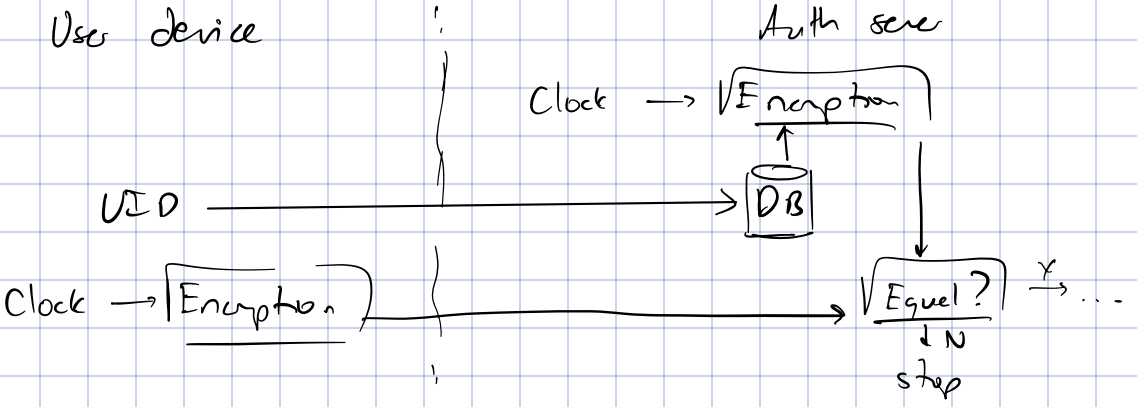
Examples: passwords, PIN, challenge-response

Token-based authentication

Usually crypto. devices (smart cards, USB tokens, digital sig.) or OTP

OTP: single-use password, no password management necessary
- Is Based on randomness, so secure
- Types:

① Synchronous: some synchronization b/w token clock & auth. server → verification



② Chained: gen. of OTP depends on prev. OTP

Initialization:
1. Auth server chooses one-way function f

2. User chooses max # of auth, n

3. Token inits seed & calculates $f^n(s)$ → $(f(f \ldots f(s)\ldots)$ (n times)

4. User sends n & $f^n(s)$ to auth. server

5. Auth server registers $f^n(s)$ w/ user ID

User

1. Token sends ID & $f^{n-1}(s)$ to auth server

$n>0$ {

2. Auth access $f^n(s)$ & calculcls $f(f^{n-1}(s)) == f(s)$

3. Auth stores $f^{n-1}(s)$, $n := n-1$

Attacker must invert f to get next OTP

③ Challenge: gen. of OTP ← auth server challenge + internal ctr.

Biometric-based auth

Enrolment process of reg. user's biometrics & then compare on use