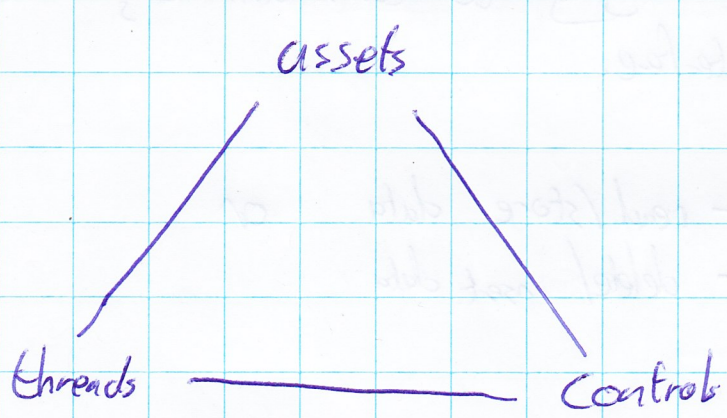
1. Wat is computer security?

Computerbeveiliging gaat over het regelen van toegang tot (digitale) assets

- assets: the valuables that need protection
  → private data or company secrets.
- regulating acces:
  - Identification
  - Authentication
  - Authorisation

⇒ There is an attacker that is trying to get acces.

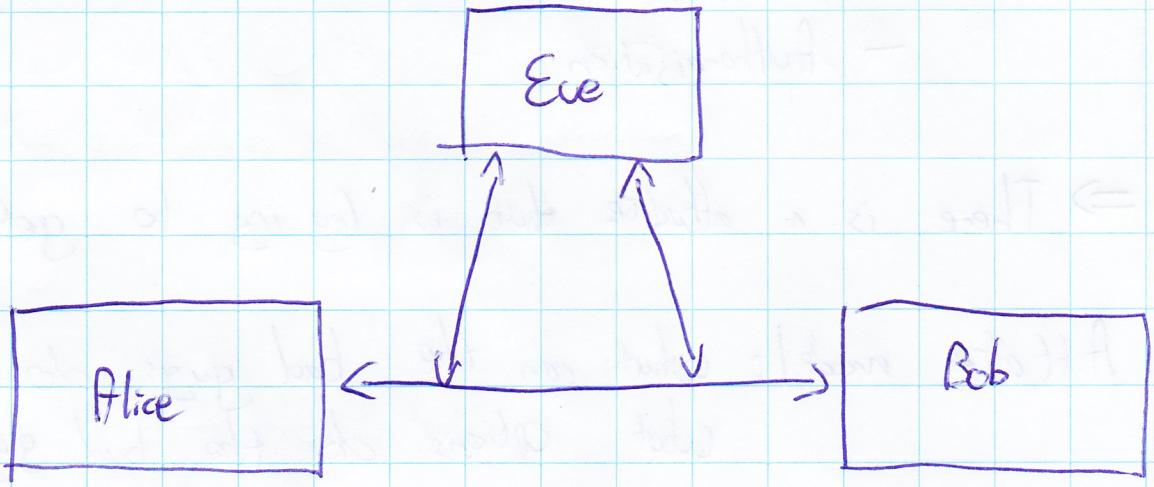
- Attacker model: what can the bad guys do / what options do the bad guys have.



Security requires a mix of:

- Technical measures (e.g. Cryptography)
- Organisational measures (e.g. ID's to regulate acces)
- Legal measures (e.g. Criminal law)

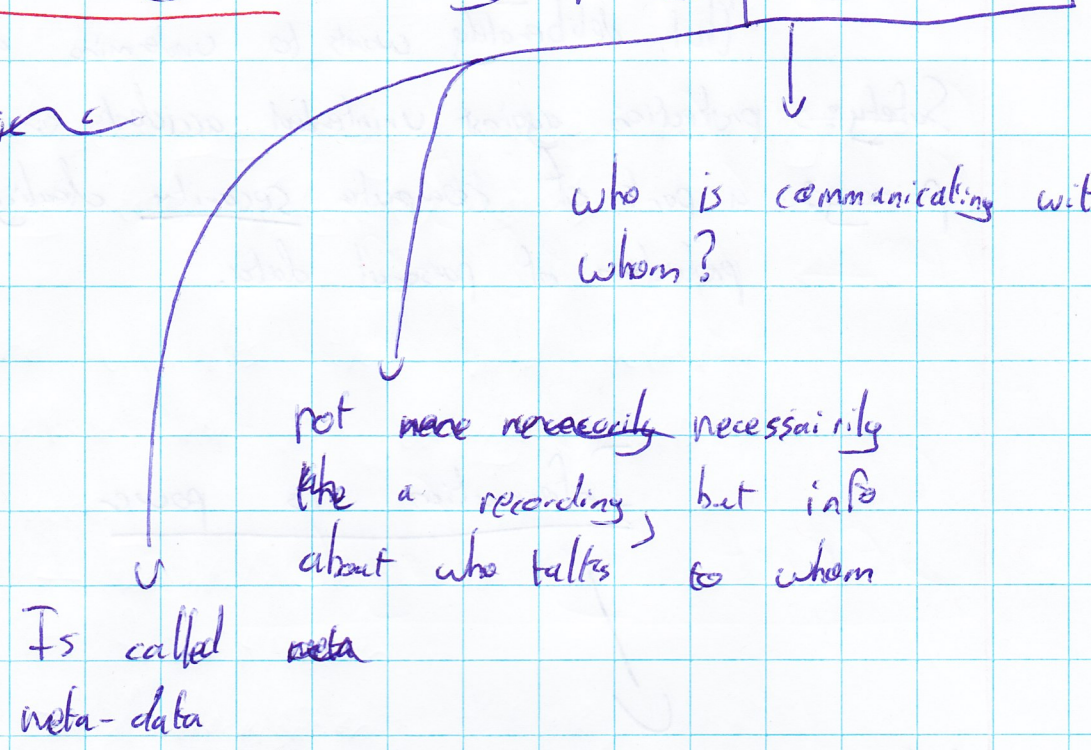
Intrusion scematic:



Alice & Bob are trying to communicate, Eve is trying to interfere.

Eve can: - read/store data or
- delete/insert data

Traffic analysis: recording/exploiting connection info



Main security goals:

- Confidentiality: Eve cannot read the content of what Alice and Bob are communicating.
- Integrity: Eve cannot alter the content of the communication.
- Authenticity: Alice and Bob are certain about each other's identities.
- Availability: Eve cannot prevent the connection between Alice and Bob
- Non-repudiation: Alice and Bob cannot deny what they have communicated at a particular stage
- Accountability: There is a reliable log of the communication history

Security: protection against an active, malicious attacker that deliberately wants to undermine a system.

Safety: protection against unintended accidents or errors.

Privacy: a part of computer security dealing with protection of personal data.

Information is power

Who has acces to which information determines the power relations in the world

Computer security is all about regulating this acces

Security stakeholders:

- Banks: Integrity, non-repudiation
- Health care: confidentiality, integrity and availability

Protocols

C = Car
CK = Car Key
K{m} = m encrypted with key K

1. Identification number:

CK ⟶ C : Idnr

2. Encrypted version of (1):

CK ⟶ C : K{Idnr} (K is a shared crypto key)

3. Sequence number:

CK ⟶ C : K{N+1} (N is last number used)

4.

4. Challenge - response:

$Ck \longrightarrow C$: "open"
$C \longrightarrow Ck$: $k\{N\}$
$Ck \longrightarrow C$: $k\{N+1\}$

Secret Key Crypto

~~plaintext~~

plaintext

Cryptography | encryption

ciphe text

Cryptanalysis | decryption

Original plaintext

cryptology = cryptography + cryptanalysis

crypto system = { algorithm + key (parameter)

Kerckhoffs principle: The strength of a crypto system must rely on the secrecy of the key
↓
the algorithm must be public.

Crypto primitives:

n. o. keys	name	key names	notation
0	hash functions	–	$h(m)$
1	symmetric crypto	shared secret	$K\{m\}$
2	asymmetric crypto	public & private key pair	$\{m\}_K$

Alphabets

An alphabet is an arbitrary set A. $a \in A$ are called letters.

- $A = \{0, 1\}$, the alphabet of bits
- $A = \{a, b, c, \dots, z\}$, the alphabet of lowercase Latin characters
- $A = \{00, 01, \dots, FF\}$, the alphabet ASCII alphabet

Words: a finite sequence $(w = a_1 a_2 \dots a_n)$ of letters over an alphabet A. $(a_i \in A)$

Basic encrypting techniques:

Suppose we have a message m and want to encrypt it to $K\{m\}$ using key K.

1. Substitution: exchange characters from the alphabet.
   K = character exchange function

2. Transposition: exchange positions of characters, block by block
   K = position exchange function

3. One-time pad: take bitwise XOR with keystream, for binary messages only.
   K = the keystream which must have at least the same length as the message

Substitution:

Key: - a function $K: A \longrightarrow A$

(A: alphabet)

- the function is bijective: it has an inverse $K^{-1}: A \longrightarrow A$ such that:

$K^{-1} \circ K = \text{identity}$

(K^{-1}: Is needed for decryption)

$m = a_1 a_2 \dots a_n$ becomes $k\{m\} = k(a_1) k(a_2) \dots$

for example: Caesar's cipher

$$C : \{a, b, \dots, z\} \longrightarrow \{a, b, \dots, z\}$$

$$C(a) = d, \; C(b) = e, \dots, C(z) = c.$$

$C\{\text{ikbengek}\} = C(i)C(k)C(b)\dots$

$= \text{lnohqjhn.}$

$$C^{-1}(d) = a, \; C^{-1}(e) = b, \dots, C^{-1}(c) = z$$

Weakness: frequency analysis

Transposition :

Key: ~~for~~

First, choose a blocksize N like $N=64$

Key: - an exchange of positions within such a block via a bijective function

$$k = \{1, 2, \dots, N\} \longrightarrow \{1, 2, \dots N\}$$

Encryption

Encrypt m = ikbengek with $N=3$ and

$$k: \{1,2,3\} \longrightarrow \{1,2,3\}$$
$$k(1)=3,\ k(2)=1,\ k(3)=2$$

1. Chop the word m into blocks with length N

ikbengek

~~ikb~~ a = ikb

b = eng

c = ek

2. Add padding to the last block

$c = ek(x)$

3. use the keg to transposition the plaintext

a = ikb ⟶ bik
b = eng ⟶ gen
c = ekx ⟶ xek

Columnnar transposition:

plaintext: ikbenknetter gek
keg = bart

b	a	r	t
i	k	b	e
n	k	n	e
t	t	e	r
g	e	k	x

alphabetical order

a b r t

b = int g
a = kk te
r = bnek
t = eeux

⟶ kkkte intg bnek eeux

↳ ciphertext

Weakness: — a transposition does not change letter frequencies
— frequent 2-letter combinations can be exploited

One-time pad

Binary message: $b_1 b_2 \dots b_n$
$b_i \in \{0, 1\}$

Key: — the same length as the plaintext

Encryption:

$$k\{m\} = m \oplus k \quad (\oplus \rightarrow \text{XOR})$$
$$= (b_1 \text{ XOR } k_1)(b_2 \text{ XOR } k_2) \dots (b_n \text{ XOR } k_n)$$

Decryption: the same as encryption

$$(b \text{ XOR } k) \text{ XOR } k = b \text{ XOR } (k \text{ XOR } k)$$
$$= b \text{ XOR } 0$$
$$= b$$

OTP is very secure when the key material is truly random.

⇒ when key material can be guessed, the OTP is broken.

⇒ OTPs require a lot of key material

↓

running out of key material is a problem

↳ keys may never be re-used

↓

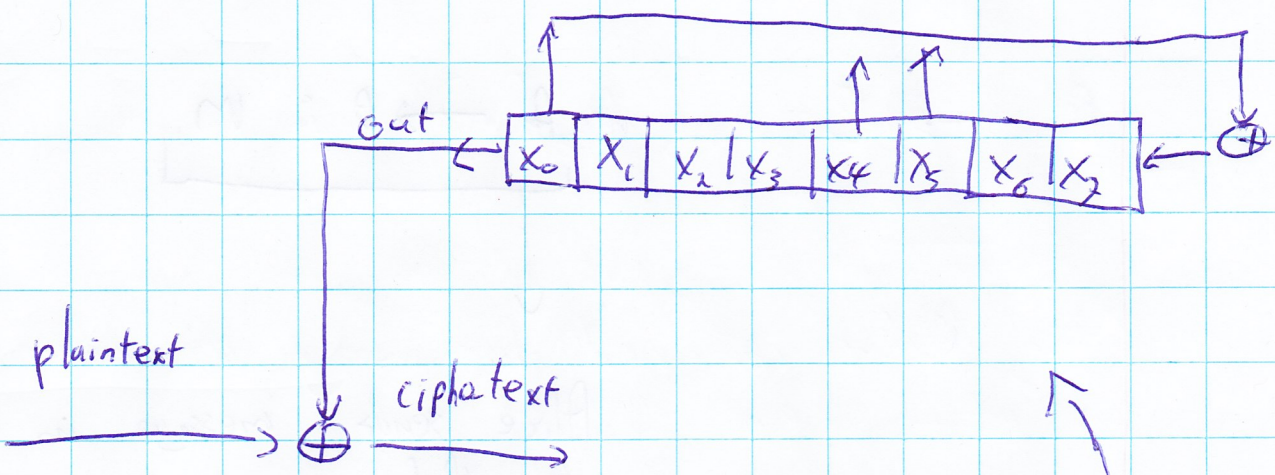
$$(b \text{ XOR } k) \text{ XOR } (c \text{ XOR } k) = b \text{ XOR } c$$

Linear Feedback Shift Register (LFSR)

↓

generates a "random" bitstream

LSFR :



at state : 1 | 00 10 10

the next state is : 100 10 100 → (1 ⊕ 1 ⊕ 0)

and the output is : 1

Good LSFRs contain all 2^n possible words

(how many x's used)

This can be analyzed by basic linear algebra.

Disadvantages of symmetric crypto:

- Large number of keys : N people need $\frac{N(N-1)}{2}$ secret keys
- If Alice and Bob share a key and Bob looses it, Alice is also affected

Security protocols: a security protocol is a list of communications in the form:

$A \rightarrow B : m$

Alice sends message m to Bob.

There is an attacker Eve who tries to undermine the goal of the protocol

→ The attacker can:
- read
- delete
- copy
- rebuild

→ The attacker cannot break encryptions (with unknown keys) or hashes.

Protocol for basic confidentiality

↳ Eve cannot read what Alice and Bob are sending eachother.

$A \longrightarrow B : K_{AB}\{m\}$

K_{AB} ↓ shared key

This protocol does not guarantee integrity

↓

Eve can change a bit in the ciphe text

↳ Bob cannot verify the message of Alice

It is very important that the key length is chosen appropriately

↓

↓ determines the strength of the encryption

If a key ever gets compromised, old messages may become readable.

⟶ Some protocols protect against this (forward secure)

Integrity protocol

$A \longrightarrow B: \; m, K_{AB}\{m\}$

Here, Eve can read the message so confidentiality is not achieved

Confidentiality and Integrity:

$A \longrightarrow B: \; K\{m, K\{m\}\}$

→ Not wise for one-time pads, since the message is revealed by two successive encryptions.

Authenticating via a shared key

↓

Problem: the key is used in every authentication session.

$C \longrightarrow B : C$

$B \longrightarrow C :$ "prove it"

$C \longrightarrow B : K_{BC}$

This is very weak because the key (or PIN) is used every time C authenticates.

Solution: send a riddle that can only be solved by with the secret key

↓

Challenge - response

- It is important that the riddle is fresh

↓

achieved via a nonce (number only used once)

Important:

→ wide range of numbers

→ randomness

Challenge - response example:

$A \longrightarrow B : N_a \quad A$ (N_a is a fresh nonce)

$B \longrightarrow A : K_{AB}\{N_a\}$

Note: the authentication key must be different from the encryption key.

Freshness alternatives:

- Nonces: require a secure random number generator
- Timestamps: require reliable / secure / synchronised clocks
- Sequence numbers: are predictable, need more carefull use.

Reflection attack: attack by mixing two sessions (a and b)

Protocol

$A \longrightarrow B : A, N_A$
$B \longrightarrow A : K_{AB}\{N_A\}, N_B$
$A \longrightarrow B : K_{AB}\{N_B\}$

Attack:

(a) $E \longrightarrow B : A, N_A$
(a) $B \longrightarrow E : K_{AB}\{N_A\}, N_B$
(b) $E \longrightarrow B : A, N_B$
(b) $B \longrightarrow E : K_{AB}\{N_B\}, N$
(a) $E \longrightarrow B : K_{AB}\{N_B\}$

Note that Eve can take the initiative for this attack.

Attack prevention

→ A solution is to use different keys for both directions:

$A \longrightarrow B: A, N_A$
$B \longrightarrow A: K_{BA}\{N_A\}, N_B$
$A \longrightarrow B: K_{BA}\{N_B\}$

A better solution is to use domain seperation:

$A \longrightarrow B: A, N_A$
$B \longrightarrow A: K_{AB}\{1 \| N_A\}, N_B$
$A \longrightarrow B: K_{AB}\{0 \| N_B\}$

OR an increment:

$A \longrightarrow B: A$
$B \longrightarrow A: K_{AB}\{N_B\}$
$A \longrightarrow B: K_{AB}\{N_B + 1\}\ K_{AB}\{N_B\}$
$B \longrightarrow A: K_{AB}\{N_A - 1\}$
$A \longrightarrow B: \text{key}\ K(N_A \oplus N_B)$

→ This also has the benefit that it sets up a session key.

Man-in-the-middle attack

Protocol

$A \longrightarrow B : A, N_A$
$B \longrightarrow A : K_{AB}\{N_A\}, N_B$
$A \longrightarrow B : K_{AB}\{N_B\}$

Attack

$A \longrightarrow E : A, N_A$
$E \longrightarrow B : A, N_A$
$B \longrightarrow E : K_{AB}\{N_A\}, N_B$
$E \longrightarrow A : K_{AB}\{N_A\}, N_B$
$A \longrightarrow E : K_{AB}\{N_B\}$
$E \longrightarrow B : K_{AB}\{N_B\}$

→ A thinks E is B and
B thinks E is A

→ Eve can't take the initiative, but waits until she can intercept an initiative of A.

→ This is used for opening cars.

Ov-chip card protocol:

There is a key management problem of secret key crypto.

→ n users require $\frac{n(n-1)}{2}$ keys.

Solution: Diversified keys, secret key K_c of card C from its identity using a master key K_m

$K_c = K_m\{ID_c\}$

Protocol:

C ⟶ T : Id_c → Terminal computes K_c

T ⟶ C : N

C ⟶ T : $K_c\{N\}$

Also, a session key can be derived from a card transaction counter:

C ⟶ T : Id_c, T_c (Transaction counter)

T ⟶ C : N

C ⟶ T : $K_c\{N\}$

C and T now share session key

$K_s = K_c\{T_c\}$

Overview:

Replay attack:

- some part of the protocol is send again
  ↳ username + password

=> Include nonce checked by verifier

Reflection attack:

- Attack on challenge - response protocol
- Data from one session is used in another session

=> Include ID info, key or domain separation

Man-in-the-middle attack:

- passive: without modification = relay attack
- active: re-encryption

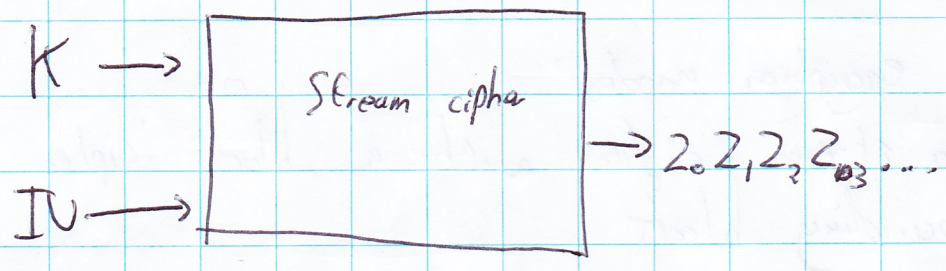
=> protecting keys, out-of-band method

Lunch-break attack:

- physical acces: car key, access badge
- attacker gets responses from prove and uses them later

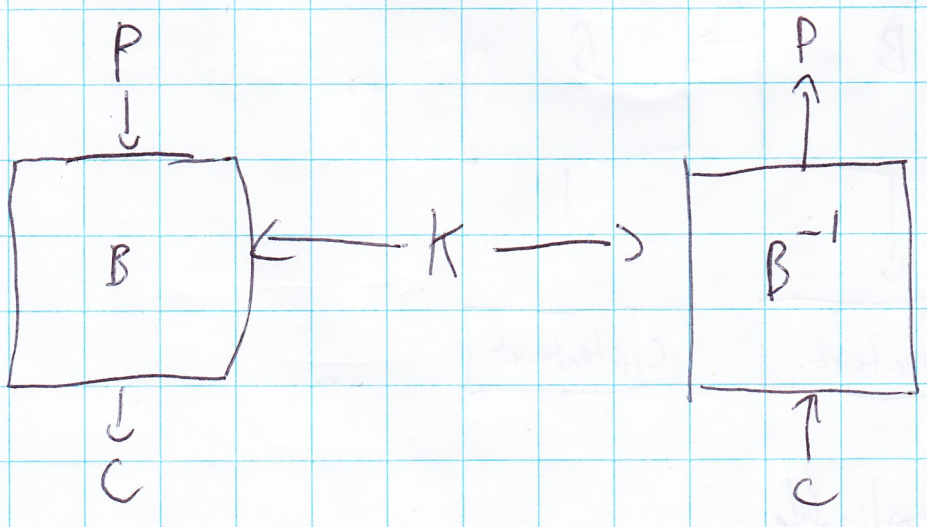
=> unpredictable challenge from verifier

Stream cipher:



A stream cipher generates a bit stream from
- K: secret key
- IV: initial value, for generating multiple keystreams per key

Block cipher:



$C = K\{P\}$

and must be invertible $P = K^{-1}(C)$

Block encryption modes

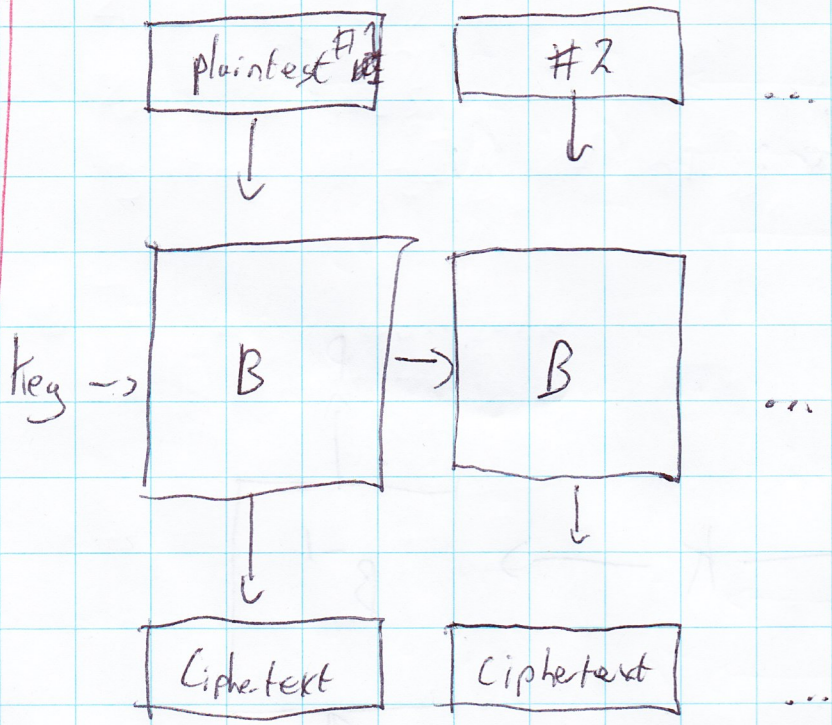
- split the message in blocks
- padding incomplete last block if needed

Stream encryption modes

- build a stream cipher with a block cipher as building block

Electronic codebook mode (ECB):

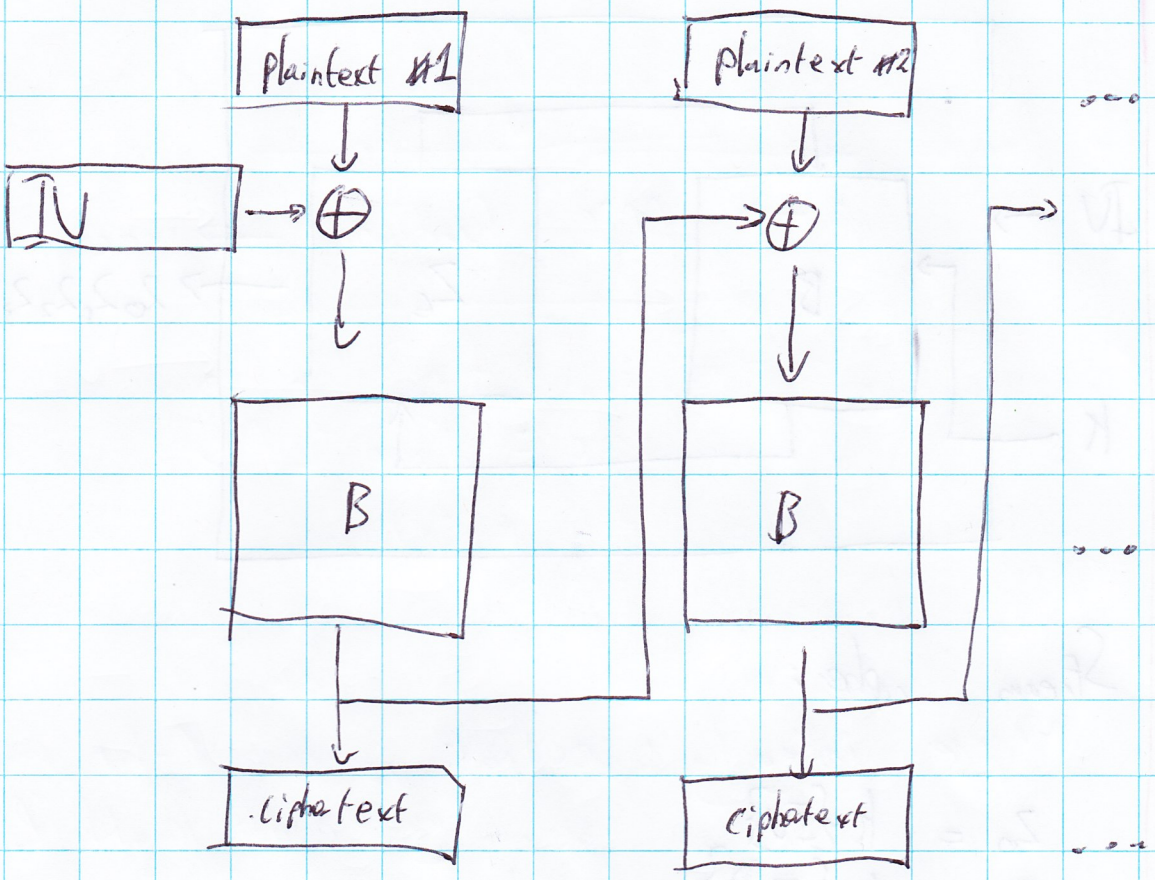
Encrypt pieces of the plaintext with the same block cipher.



→ parallelizable

But equal plaintext = equal ciphertext

Cipher block chaining mode (CBC)



The first plaintext block is randomized with an Initial value (IV).

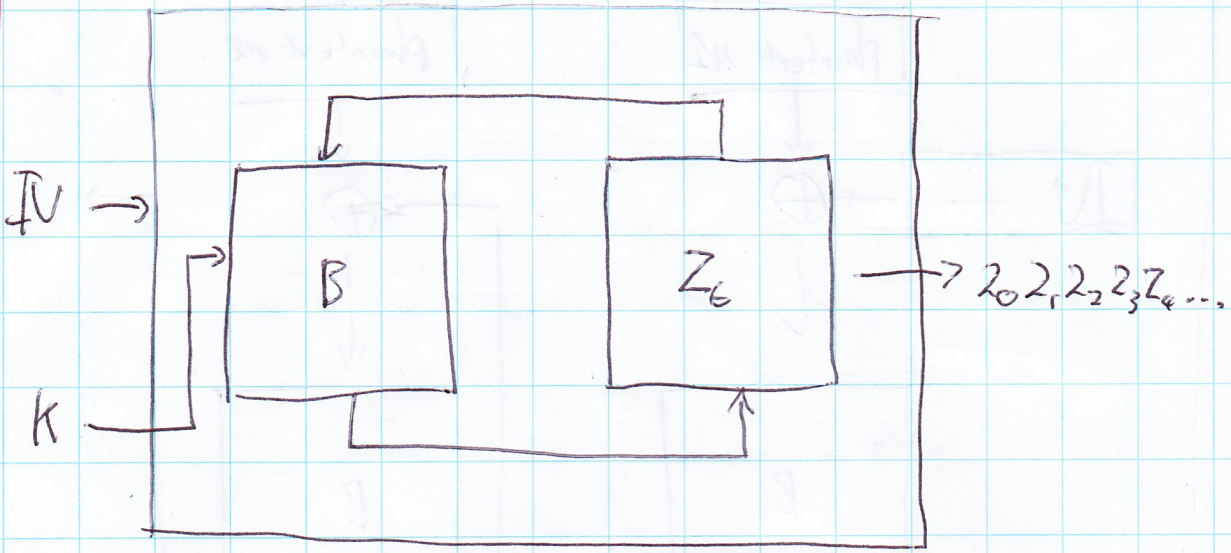
Solves information leakage in ECB:

- Equal plaintext leads to different ciphertext.
- requires randomly generating IV
- can't be done parallized.

IV → most of the time IV = 0

→ decryption can be parallel
→ IV has to be managed and transferred.

Stream encryption: Output feedback mode (OFB)



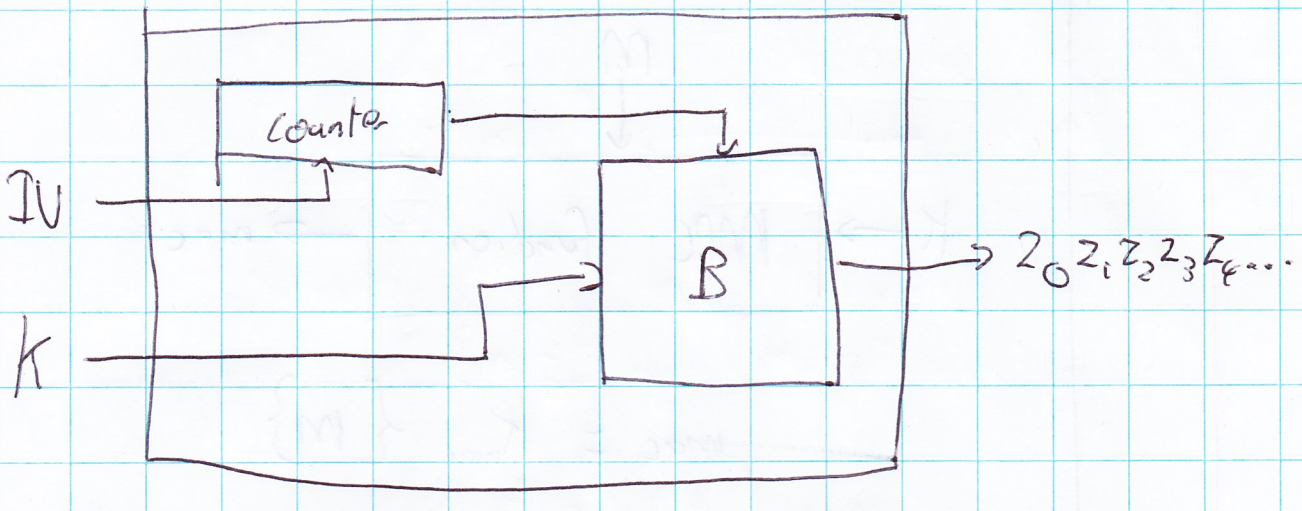
Stream cipher:

$$Z_0 = K\{IV\}$$
$$Z_1 = K\{Z_0\}$$
$$Z_{2i} = \{K_{i-1}\}$$

Key stream: $K\{IV\}, K\{K\{IV\}\}, K\{K\{K\{IV\}\}\}, \ldots$

$\Rightarrow$ Is not parallizable

Stream encryption: Counter mode



Stream cipher:

$z_0 = k\{IV\}$

$z_1 = k\{IV+1\}$

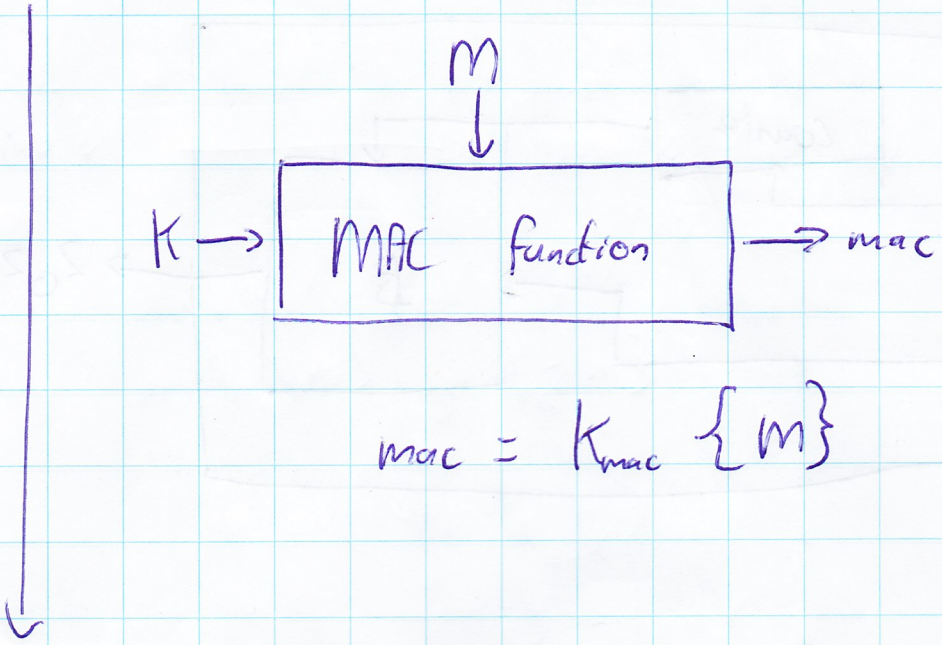
$z_i = k\{IV+i\}$

key stream: $k\{IV\}, k\{IV+1\}, k\{IV+2\}, \ldots$

This method is fully ~~parallizable~~ parallelizable and is the most used block cipher mode

$\Rightarrow$ IV management is critical for security

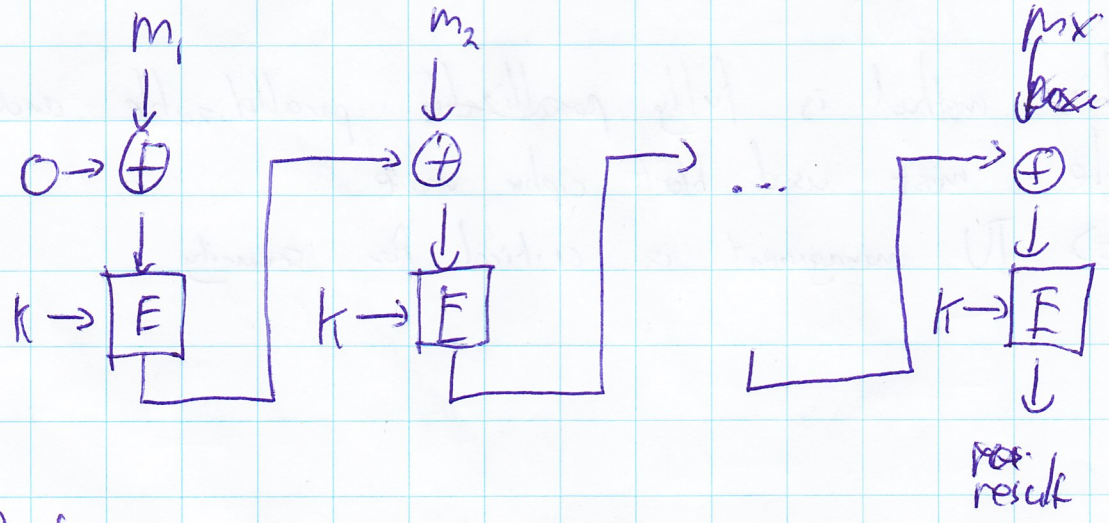
Message authentication code functions (MAC)



$$mac = K_{mac}\{M\}$$

Message integrity: output is a ℓ-bit mac or tag with length ℓ.

Cipher block chaining MAC (CBC-MAC)



→ Apply CBC to padded message
→ Take MAC equal to last ciphertext block
→ Throw away the blocks

Protection of sensitive data on e-passport.

↓

MRZ: machine readable zone that contains keys

↓

K_{enc} : confidentiality
K_{mac} : integrity

BAC - protocol:

⇒ Reader has K_{enc} and K_{mac}

Passport ⟶ Reader : N_P

Reader ⟶ Passport : $K_{enc}\{m\}$ $K_{mac}\{m\}$

$m = N_P, N_R, K_R$

Passport ⟶ Reader : $K_{enc}\{n\}$ $K_{mac}\{n\}$

$n = (N_P, N_R, K_P)$

Session key $K = K_P \oplus K_R$

Vulnerabilities:

- Fingerprinting passports by collection out-of-protocol messages read from different countries and then using this info to for your advantage
  ⇒ to know what country someone's from.

- Timing attacks:

Reader ——————> Passport:

$$K_{enc}\{m\} \quad , \quad K_{mac}\{m\}$$

$$m = (N_P, N_R, K_R)$$

We can guess N_P one by one because it will take a longer time to check N_R when N_P is right.

4e Hashing

Hash function: A hash function h takes a message m of arbitrary length and yields an outcome h(m) of fixed length

$$h: \{0,1\}^* \longrightarrow 2^n \quad (\text{typical: } N = 160,\ 256,\ 512)$$

- Every bit depends in a complicated way of all bits of m

The ideal case is that hash functions behave like Random Oracles.

1. m arrives at RO.
2. If m was received earlier, earlier response z is send back
- 2b. If not in archives, ~~employs~~ RO will randomly generate z
3. ~~Manage copies~~ z RO makes copy of z
4. RO puts one z and m back in the archive
5. RO returns z

Properties of hash functions:

- Collission resistance: It is hard to find two bit strings $x \neq x'$ such that $h(x) = h(x')$

  expected Cost: $2^{n/2}$ computations

- Preimage resistance: given a bit string y, it is infeasible to find a bit string x such that $H(x) = y$

  expected Cost: 2^n computations

- 2nd Preimage resistance: given a bit string x, it is infeasible to find a different bit string x' such that $H(x) = H(x')$

  expected Cost: 2^n computations

- Fixed length: the length of the output is fixed and does not rely on the input size

Collission resistance implies 2nd pre image resistance but not vice versa.

Password protection on servers

→ It is not wise to store user passwords on a server in the clear

Solution: store hashes of passwords

- After entering a password, the server computes the hash and compares it to the data base entry of the user.

⇒ Requirement: pre-image resistance
(given $h(M)$, it is hard to find M)

A password file would look like this:

user	password
bart	h(passwd1)
pete	h(passwd2)
...	...

Issues: - users with the same passwords have the same hash
- users with easy-to-guess passwords can be hacked more easily

What can a hacker do?

- Dictionary attack ⟶ extremely fast
- Build passwords from a dictionary and try these
- Try all combinations of characters

↓

A password guess can hit ANY entry in the hash file

Protection against these attacks:

Diversification: include unique username in hash input (h(username; passwd)).

↓

each attempt is dedicated to a single user's password.

Salting: include a random salt pa user: (h(salt; passwd)

Salted password file:

user	salt	hash
bart	4a	h(bla, passwd)
peter	aap	h(aap, passwd)
...	...	...

Domain seperation:

(Salting is an application of domain seperation)

A hash function h can be used to build two hash functions h_0 and h_1:

$$h_0(m) = h(0 \| m)$$
$$h_1(m) = h(1 \| m)$$

↳ a hash function can be used to build 2^n hash functions.

$$h_a = h(a \| m)$$
$$a = n\text{-bit string}$$

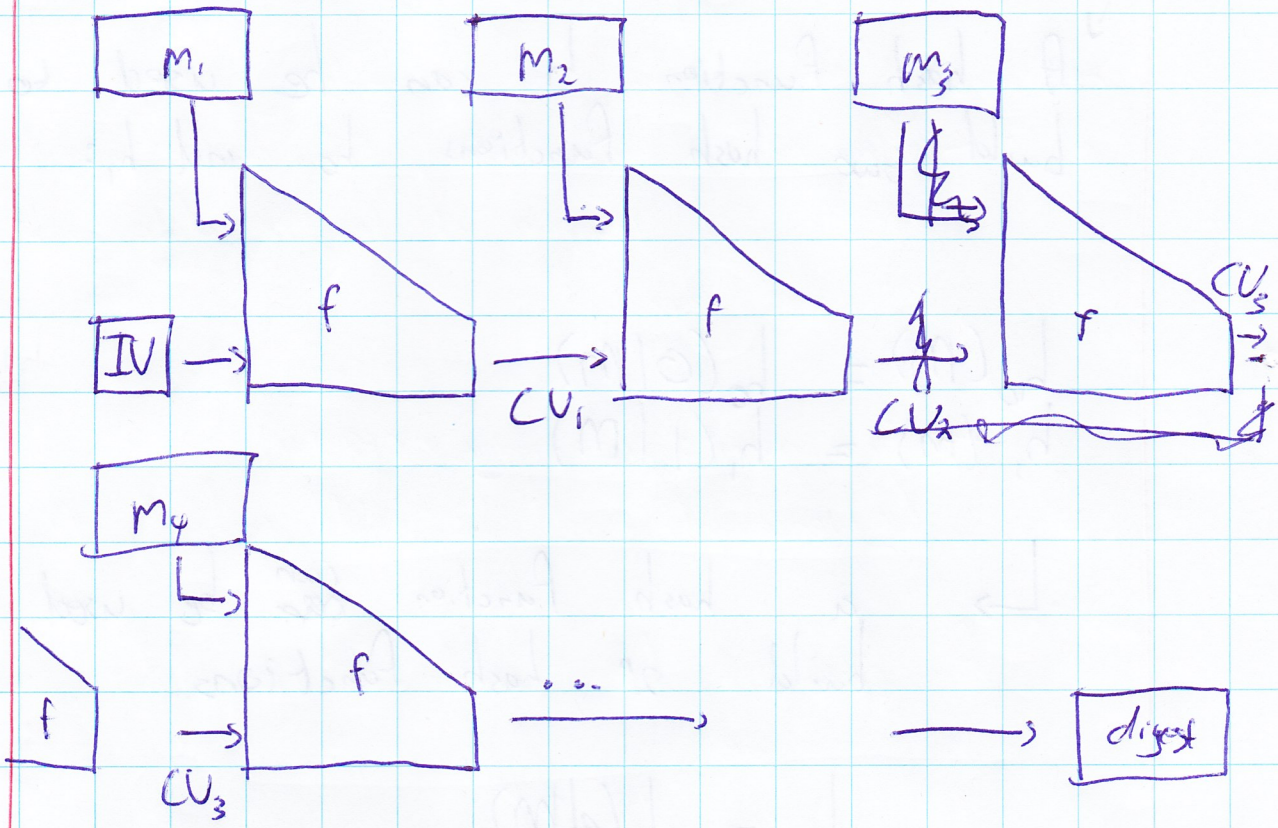
key stretching: slowing down a hash function to make it harder to crack

↓

balance between convenience and security

Merkle – Damgård design:

Problem: how to build a function accepting input with any length?



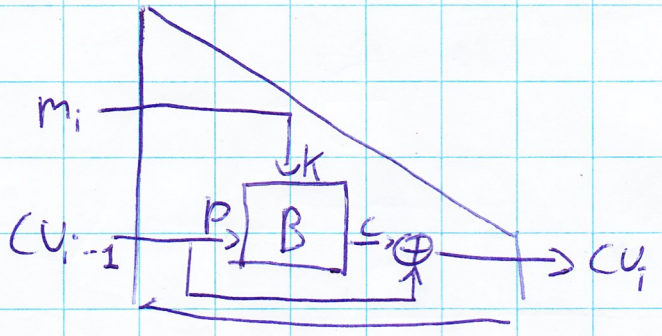
f: - executed untill M is hashed
- Requires fixed input length
- Requires IV

↓

Problem is reduced to f

It was a dominating hash idea function but now it's not the way to do it anymore.

Davies meyer's "f" function:



Problem: generating a collission only takes ~~2^64~~.
$2^{n/4}$ = 4 billion ⊕ computations

Overview standard hash protocols:

MD5:
- based on addition, rotation and XOR: ARX
- 128-bit digest

SHA-1:
- designed at NSA, mostly rip-off of MD5
- SHA = Security hash algorithm
- 160-bit digest

SHA-2:
- better version of SHA-1 (also designed at NSA)
- 6 functions with 224-, 256-, 384- and 512-bit digest

⇒ No motivation or rationale was ever given for any of them.

HMac authentication:

Mask generation function 1 (MGF 1)

↓

Comp for long hash output

Compute: $h1 = h(m|1)$

$h2 = h(m|2)$

$h3 = h(m|3)$

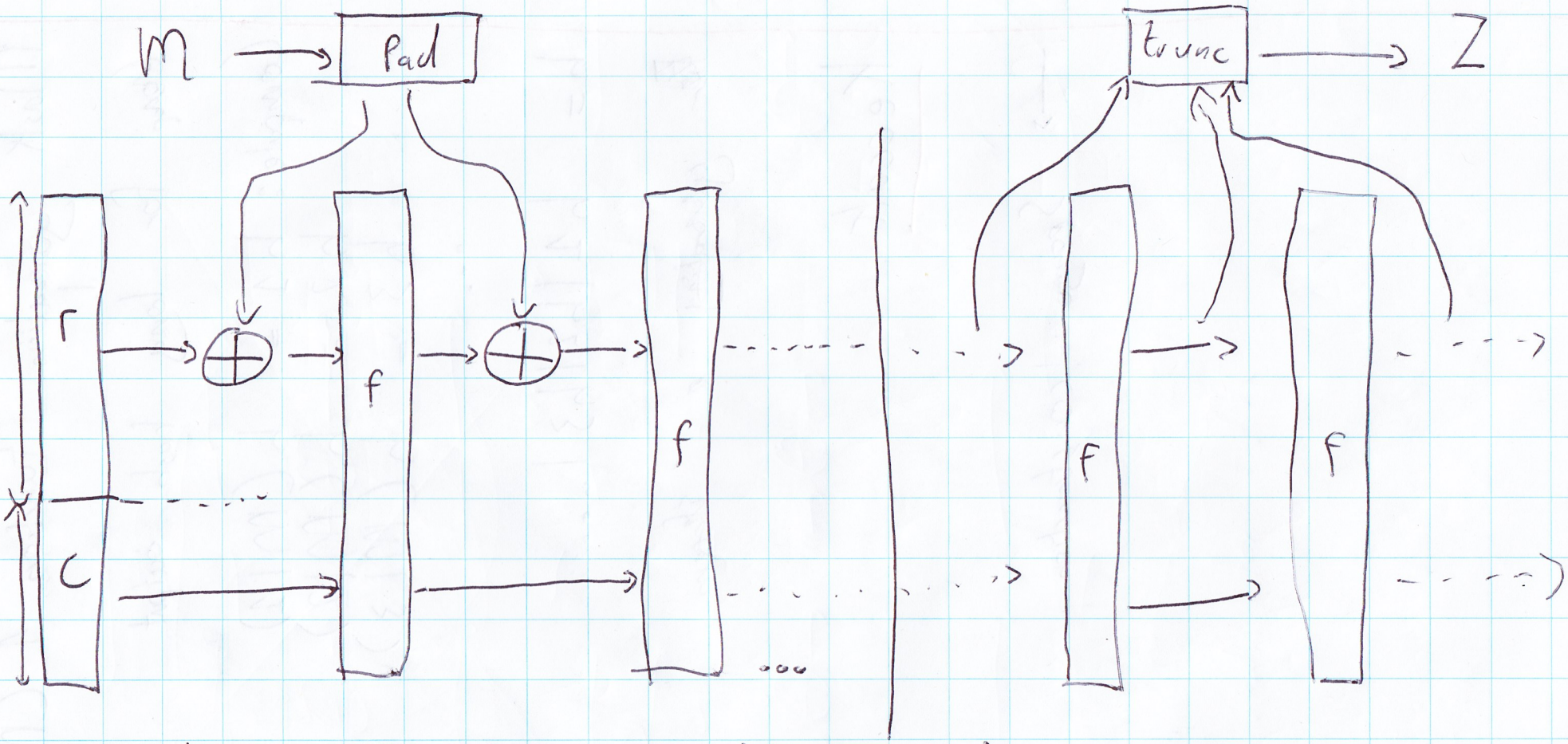
...

$h = h1|h2|h3|...$

~~For~~ generating a stream

Keccak

—> Sponge construction



f = b-bit permutation (b = r + c)

r = bits of rate

c = bits of capacity (secure up to $2^{c/2}$)

function f: — block cipher with fixed key
— every output bit depends on all input bits in a complicated way.

↓

Q7 7 different permutations with different widths:
- 25
- 50 ↓ x2
- 100 ↓ x2
- 200 ↓ x2
- 400 ↓ x2
- 800 ↓ x3
- 1600 ↓ x2

$b \in \{25, 50, 100, 200, 400, 800, 1600\}$
$b = r + c$

Security strength $= c/2$

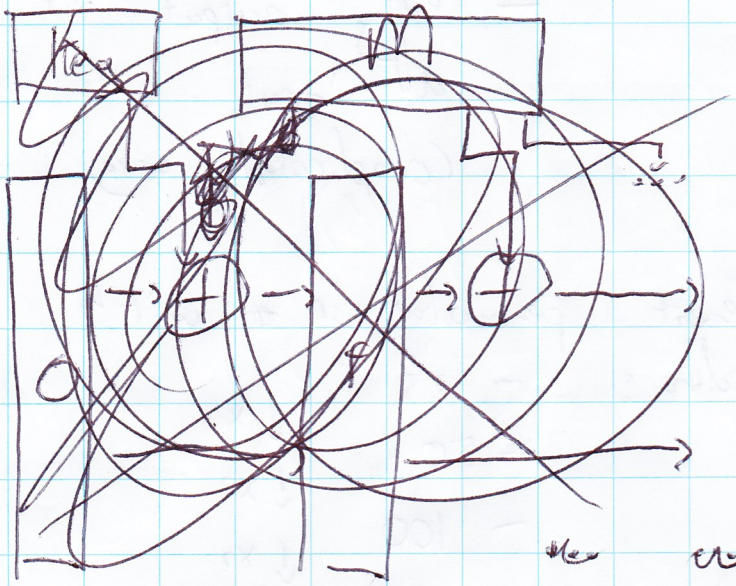
Q6. SHAKE 128: $r = 1344$ and $c = 256$
→ $b = 1600$

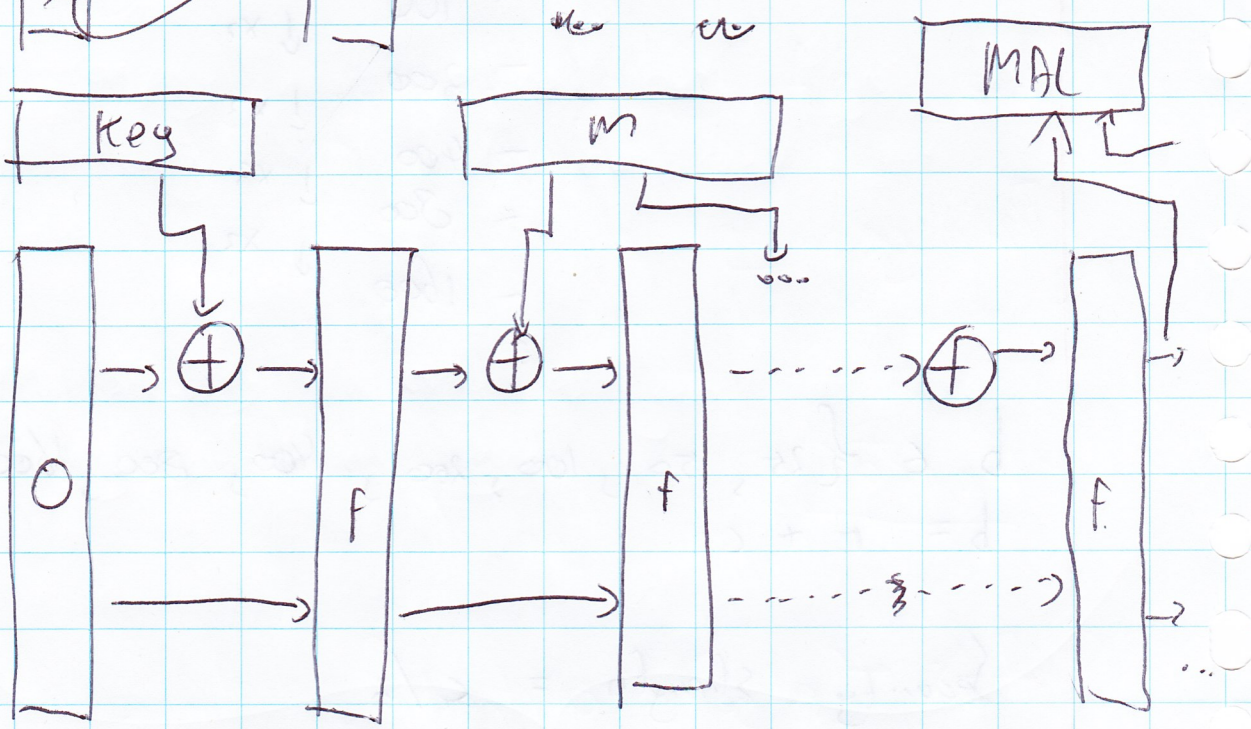
strength $= 128$

lightweight one: $r = 40$ and $c = 160$
→ $b = 200$

strength $= 80$ (SHA-1's strength)

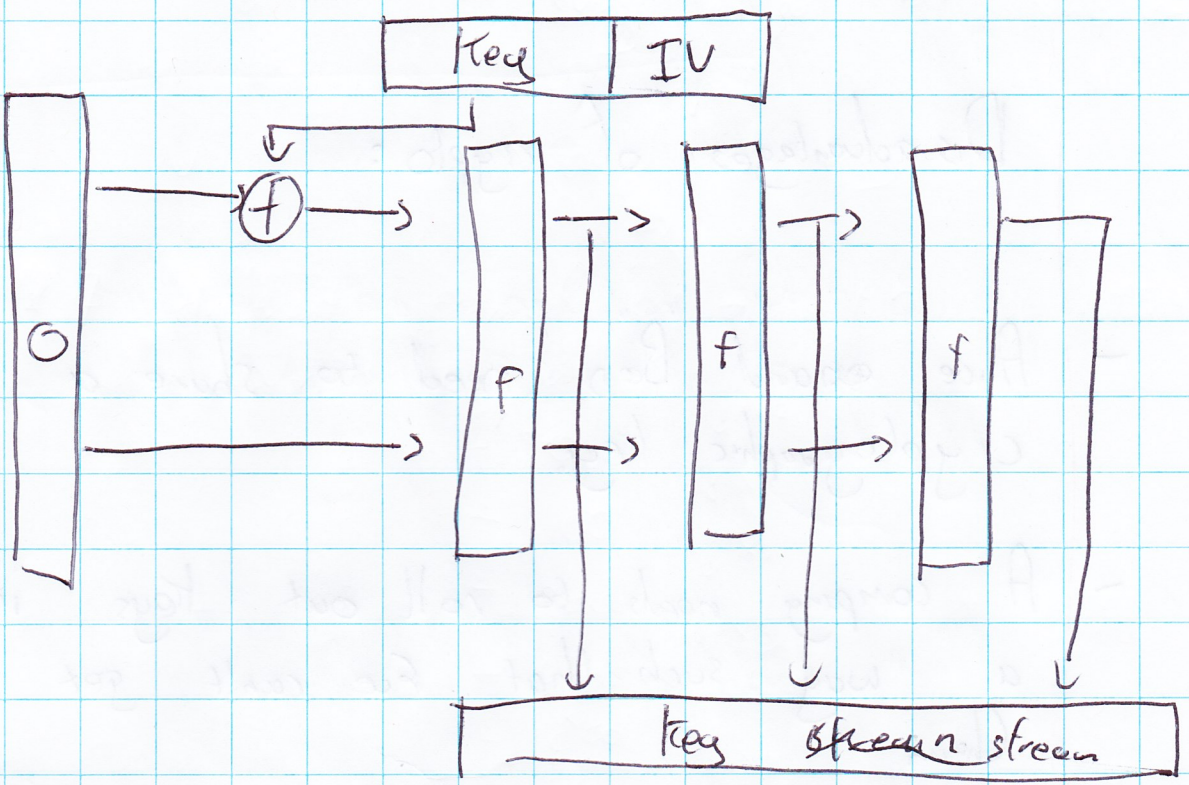
MAC - modes





→ No need for HMAC

Stream cipher :



Public Key crypto

Disadvantages of crypto:

- Alice and Bob need to share a cryptographic key.
- A company needs to roll out keys in a way such that Eve can't get them
- The security is only as good as the secrecy of these keys

Cryptography only reduces problems to:

- securely generating keys
- securely establishing keys
- keeping the keys out of Eve's hands

Establishment:

How do Alice and Bob establish a shared key?

- Physically meet
- A trusted common friend

For a company, it is much harder:

- Keys must be protected in the field

Trusted third party: Wally

Alice's key: A_K
Bob's key: B_K

1. $A \longrightarrow W : K\{K_A\}$
2. $W \longrightarrow B : K_W\{K_A\}$

1. Alice sends an encrypted key A to Wally
   Wally decrypts key A and encrypts it with his personal key
2. Wally sends it to Bob
   Bob decrypts key A

Problem: Alice and Bob both rely on Wally

We need to get Wally out of the equation.

More trusted common Friends.

1. Alice generates a random key K_i for each Friend.
2. Alice sends the keys via the common Friends to Bob.
3. Alice and Bob take the xor of all K_i's as shared key K

Risk:--Conspiracy

- Denial of service: A Friend can send a wrong key to Bob

Identifying this Friend is not easy.

Public key crypto:

- Authenticate an entity without sharing a key with that entity
- Electronic signatures
- Set up a key remotely without the need for a secure channel

Public key crypto works with two keys per user:

- Private key P_rK: never to be revealed to the outside world
- Public key PK: to be published freely

Signatures:
- Alice uses P_rK_A for signing messages
- Anyone with PK_A can verify Alice's signature

Encryption:
- Anyone can use PK_A to encipher a message meant for Alice
- Alice can decipher with P_rK_A

Key establishment:
- Bob uses P_rK_B and PK_A to compute K_{AB}
- Alice uses P_rK_A and PK_B to compute K_{AB}

Private and Public Keys are like Dictionaries:

P_{rk} : Ourgeze → Putch
P_k : Dutch → Ourgeze

No-one has P_k, but the P_k can be computed with P_{rk} but it's very difficult.

Prime Factorization:

Any non-prime number can be written as a product of the primes.

$30 = 2 \cdot 3 \cdot 5$
$100 = 2^2 \cdot 5^2$

↓

computationally a hard problem

↓

finding two very large primes p and q, you can publish $n = p \cdot q$ and no-one will easily find out what p and q are.

Z_n : the set of numbers modulo N.

$Z_5 = \{0, 1, 2, 3, 4\}$

$Z_n = \{0, 1, 2, 3, \dots, n-1\}$

$10 + 6 \equiv 1 \mod 15$

$3 + 2 \equiv 5 \mod 15$

If a product is 1, one can say:

$4 \cdot 4 \equiv 1 \mod 15$ $\qquad$ $2 \cdot 13 \equiv 1 \mod 17$

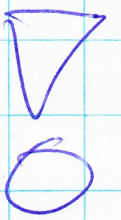
$\frac{1}{4} \equiv 4 \mod 15$ $\qquad$ $\frac{1}{2} \equiv 13 \mod 15$

Multiplication table:

Z_5:

Z_5	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

every prime p has a table Z_p for which every $n \in Z_p$ has an inverse $\frac{1}{n} \in Z_p$



A Group:

definition:

- closed: $\forall a, b \in A \longrightarrow a \circ b \in A$
- associative: $\forall a, b, c \in A \longrightarrow (a \circ b) \circ c = a \circ (b \circ c)$
- neutral element: $\exists e \in A, \forall a \in A \longrightarrow a \circ e = e \circ a = a$
- inverse element: $\forall a \in A, \exists a' \in A \longrightarrow a \circ a' = a' \circ a = e$
- abelian (optional): $\forall a, b \in A \longrightarrow a \circ b = b \circ a$

$\circ$ = a binary operation
(it can also be $+, -$, etc.)

So:

$(A, +)$: $e = 0$ $\quad a' = -a$
$(A, \times)$: $e = 1$ $\quad a' = a^{-1}$ $\quad (1/a)$

$\#A$: the number of elements in A

Group:
Vb $(\mathbb{Z}, +)$
$(\mathbb{Q}, +)$

Non Groups:
$(\mathbb{N}, +)$: no neutral element, no inverses.

$\langle g \rangle$: cyclic group

$\{ [0]_g, [1]_g, [2]_g, \ldots$

Neutral element: $[0]_g$
Inverse of $[i]_g$: $[\#g - i]_g$

g = generator

Sequence:

$i = 1$: $a \circ a$
$i = 2$: $a * a$
$i = 3$: $a * a * a$
$i = n$: na (+) of a^n (·)

$\#a$ = smallest n such that $a^n = 1$ (·) or $na = 0$ (+)

Vb Z_{21} : 21

Order of 0 : 1
Order of 1 : 21
Order of 2 : 21
Order of 3 : 7

if $\gcd(a,b) = 1$ then a and b are called coprime

Euclidean algorithm:

$\gcd(n,m) = \gcd(m, n \bmod m)$

↓ can be applied till one of the elements is 0

$\gcd(171, 111) = \gcd(111, 171 \bmod 111)$
$= \gcd(111, 60)$
$= \gcd(60, 111 \bmod 60)$
$= \gcd(60, 51)$
$= \gcd(51, 60 \bmod 51)$
$= \gcd(51, 9)$
$= \gcd(9, 51 \bmod 9)$
$= \gcd(9, 6)$
$= \gcd(6, 9 \bmod 6)$
$= \gcd(6, 3)$
$= \gcd(3, 6 \bmod 3)$
$= \gcd(3, 0)$
$= 3$

Subgroups

$(B, *)$ is a subgroup of $(A, *)$ if:
- $B \neq \emptyset \subseteq A$
- $e \in B$
- $\forall a, b \in B: a * b \in B$
- $\forall a \in B$: the inverse of a is in B

If $(B, *)$ is a subgroup of $(A, *)$: $\#B$ divides $\#A$

$Z_p^*, \times$ with prime p

(Z_p^* : Z_p with 0 removed
→ Order of the group is $p-1$
↳ Group turns out to be cyclic

Inverse of $x \in Z_p^*$

$$x^{-1} = x^{(p-1)-1} = x^{p-2}$$

$$Z_p^*, \times \cong Z_{p-1}, +$$

$$Z_{23}^*, \times = Z_{22}, +$$

Z_n^* redefined:

Z_n^* := all integers smalle then n and coprime to n ($\gcd(x,n) = 1$).

↓

If: $\gcd(a,n) = 1$
$\gcd(b,n) = 1$

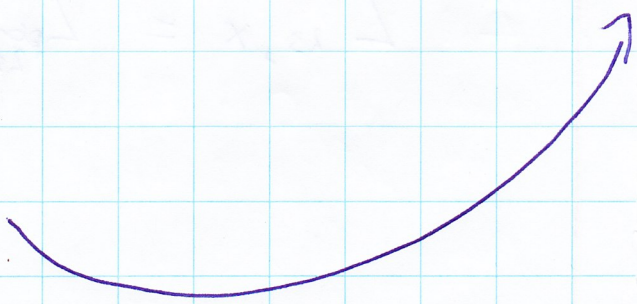
Then: $\gcd(ab,n) = 1$

Extended Euclidean Algorithm:

$n \cdot x + m \cdot y = \gcd(n,m)$

Vb $\gcd(171, 111)$ → we need to find $xn + ym = \gcd(n,m)$

$(171 - 1 \cdot 111) = 60$
$(111 - 1 \cdot 60) = 51$
$(60 - 1 \cdot 51) = 9$
$(51 - 5 \cdot 9) = 6$
$(9 - 1 \cdot 6) = 3$
$(6 - 2 \cdot 3) = 0$



This was the Euclidian algorithm, now backwards substitution:

$$3 = 9 - 1 \cdot 6$$
$$3 = 9 - 1 \cdot (51 - 5 \cdot 9)$$
$$= -51 + 6 \cdot 9$$
$$3 = -51 + 6 \cdot (60 - 1 \cdot 51)$$
$$= 6 \cdot 60 - 7 \cdot 51$$
$$3 = 6 \cdot 60 - 7 \cdot (111 - 1 \cdot 60)$$
$$= 13 \cdot 60 - 7 \cdot 111$$
$$3 = -7 \cdot 111 + 13 \cdot (171 - 1 \cdot 111)$$
$$= 13 \cdot 171 - 20 \cdot 111$$

$x = 13$ $\quad$ $y = -20$

Relative primes Lemma

m has a multiplicative inverse modulo N if $\gcd(m, N) = 1$

Proof ——————————→

Extended gcd yields:

x & y with

$$m \cdot x + N \cdot y = \gcd(m, N) = 1$$

$$(m \cdot x) \bmod N + \underbrace{(N \cdot y) \bmod N}_{0} = 1$$

$$m \cdot x \bmod N = 1$$

$$m \cdot x \equiv 1 \bmod N$$

so there is an integer y such that

$$m \cdot x = 1 + N \cdot y$$

$$m \cdot x - N \cdot y = 1$$

$\gcd(m, N)$ divides m and N, so it divides $m \cdot x - N \cdot y = 1$. But if $\gcd(m, N)$ divides 1 it must be 1 itself.

Order of (Z_n^*, x)

$\#(Z_n^*, x) = \phi(n)$ ($\phi(n)$ = all integers smaller than and coprime to n)

- for a prime p, all integers from 1 to $p-1$ are coprime to p: $\phi(p) = p-1$
- if $n = a \cdot b$ with a and b coprime $\phi(a \cdot b) = \phi(a) \cdot \phi(b)$
- for the power of a prime p^n:

$$\phi(p^n) = (p-1)p^{n-1}$$

Com Computing $\phi(n)$:

- factor n into primes
- apply $\phi(p^n) = (p-1)p^{n-1}$

↓

computing this is as hard as factorizing n.

$$x^{\phi(n)} \bmod n = 1$$

$$\Rightarrow x^{-1} = x^{\phi(n)-1} \bmod n$$

RSA

↓

trapdoor one-way function $y = f(x)$

→ given x, computing $y = f(x)$ is easy
→ given y, finding x is difficult
→ given y and trapdoor info: computing $x = f^{-1}(y)$ is easy

RSA public key function:

$$y = x^e \bmod n$$

with $x, y \in Z_n$

(n, e) = public key
$n = p \cdot q$ (large primes)
e = small prime
Trapdoor: knowledge of p and q

RSA private key:

$(Z_n^*, x) \longrightarrow$ order $= \phi(n)$

$\forall x \in Z_n^*$:

$$x^{\phi(n)} \bmod n = x^{\underbrace{(p-1)(q-1)}} = 1$$

primes so $\phi(n) = (p-1)(q-1)$

d:

$$e \cdot d + k \cdot (p-1)(q-1) = 1$$

$$d = \frac{1 - k \cdot (p-1)(q-1)}{e}$$

$$(x^e)^d = x^{e \cdot d} = x^{1 - k \cdot \phi(n)} = x \cdot x^{-k \cdot \phi(n)}$$

$$= x \cdot (x^{\phi(n)})^{-k}$$

$$= x$$

Private key operation:

$$x = y^d \bmod n \quad \text{with} \quad d = e^{-1} \bmod (\phi(n))$$

d is an inverse of e modulo $\phi(n)$

Public key = (e, n)
Private key = (d, n)

Security of RSA relies on difficulty of factoring n

Using RSA

Bob computes: $C = m^e \bmod n$
(e, n) = public key

Alice deciphers: $m = C^d \bmod n$
(d, n) = private key

Attention points:

- m shall have enough entropy (Eve can guess m)
- Algebraic properties
  $C_1 = m_1^e$
  $C_2 = m_2^e$
  Eve can construct $C_3 = C_1 \cdot C_2 \bmod n$
- RSA decryption is relatively slow

Solution: use RSA for establishing a symmetric key.

↓

1. Bob randomly generates $r \in Z_n$
2. Bob sends $c = r^e \bmod n$ to Alice
3. Alice deciphers c back to r
4. both compute shared secret as hash(r)

RSA for signatures:

1. Alice signs message with Private key
   $s = m^d \bmod n$ (d, n private key)
2. Bob verifies the signed message
   $m' = s^e \bmod n$ (e, n public key)
   $m' = m$?

Attention points:

- $s_1 = m_1^d$ & $s_2 = m_2^d$

  ↓

  $s_3 = s_1 \cdot s_2 \bmod n$

  ↳ Solution: random bits or hashing

RSA Key pair generation

1. Choose public exponent e (small prime)
2. Chooses a ~~mod~~ length for n ($|n|$)
3. generates prime p with length $\ell = |n|/2$
4. generates prime q such that $p \cdot q$ has length $|n|$
5. computes $n = p \cdot q$
6. Computes $d = e^{-1} \bmod (\phi(n))$ ~~w~~

6. RSA Example

1. $e = 3$
2. $p = 5$, $q = 11$

$$n = p \cdot q = 5 \cdot 11 = 55$$

$$\phi(n) = \underbrace{p-1 \cdot q-1}_{\downarrow} = 4 \cdot 10 = 40$$

∇ test $p-1$ & $q-1$ co-prime to e.

3. Compute d

$$3x = 1 \mod 40$$

$$40 = 13 \cdot 3 + 1$$
$$1 = 40 - 13 \cdot 3$$

$$d = -13 \mod 40 = 27$$

4. $m = 19 \in \mathbb{Z}_n$

a) $c = m^e \mod n = 19^3 \mod 55 = 39$ (encrypt)

b) $m' = c^d \mod n = 39^{27} \mod 55 = 19$ (decrypt)

Public key validation

How does Bob know the public key of Alice is valid?

- Bob relies on Alice alone
- Bob relies on their mutual friends
- Bob relies on Central Authority (CA)

Trust on first use (TOFU)

→ Bob trusts that received public key is Alice's without validation.

↓

Man-in-the-middle: Eve can substitude public key by hers

System supports manual key validation

ID chip card

Two key pairs:
- authentication
- ~~non-repudiation~~ non-repudiation (can't deny signing)

each key has its own PIN

↓

if sign PIN, chip returns [x10] Prk
if auth PIN, chip returns [x11] Prk

Discrete logarithm

Let g be a generator of $(Z_p^*, \cdot)$

$\downarrow$

$(Z_{p-1}, +)$

Let $A = g^a \bmod p$ & $B = g^b \bmod p$

then $A \cdot B = g^a \cdot g^b = g^{a+b \bmod p-1}$

→ requires knowledge of exponent a and b, given A and B

$\downarrow$

finding this exponent is called discrete log (very hard with large p)

Key pairs based on discrete log:

private key: $a \in Z_{\#g}$

public key: $A = g^a \in \langle g \rangle$

for $(Z_p^*, \cdot)$:

private key: $a \in Z_{p-1}$

public key: $A = g^a \in Z_p^*$

Merkle-Diffie-Hellman key exchange

Alice has $PrK_{Alice} = a$ and $PK_{Alice} = g^a = A$
Bob has $PrK_{Bob} = b$ and $PrK_{Bob} = g^b = B$

↓

Alice → Bob : A
Bob → Alice : B

Bob: A^b
Alice: B^a

K_{AB}: $A^b = g^{ab} = g^{ab} = g^{ba} = B^a$

↳ Both need to work in the same g

El Gamal

Encryption with $Pk = A$

→ convert clear-text M to element $m \in \langle g \rangle$

→ randomly generate r, $R = g^r$

→ define cryptogram as $\{m\}_A = (R, m \cdot A^r)$

De cryption:

→ $C = (c_1, c_2)$ with $c_i \in \langle g \rangle$

→ plaintext

$$[(c_1, c_2)]_a = \frac{c_2}{(c_1)^a}$$

$$\downarrow$$

$$[\{m\}_A]_a = [R, m \cdot (g^a)^r]_a$$

$$= \frac{m \cdot g^{a \cdot r}}{(g^r)^a} \rightarrow c_1^{-x} \rightarrow r = -x = p-1-x$$

$$= m$$

El Gamal signatures

Signing with $Pk = a$ and message m

→ generate $(r, R = g^r)$ randomly with $gcd(r, \#g) = 1$

$$sign_a(m) = \left(\frac{h(m) - a \cdot R}{r} \bmod \#g \right)$$

Verification: $(m, (s_1, s_2))$

$$g^{h(m)} \stackrel{?}{=} (s_1)^{s_2} \cdot A^{s_1}$$