Perfect Cypher

The plaintext space should be $\in \{0,1\}^n$ to D
Given a cyphertext C the probability that exists k_2 such that $D_{k_2}(c) = P$ for any plaintext P is equal to the apriori probability that P is the plaintext

↳ Cyphertext does not reveal ANY INFORMATION on the plaintext

$$Pr[P \mid C] = Pr[P]$$

An example: One Time Pad

The key is symmetric, and chosen at random

$$E_k(P) = C = P \oplus K$$

↳ xor-ring plaintext and key

this kind of encryption is strong, as long as the key is NOT REUSED

↳ The randomness of the cyphertext is IMPORTANT, and its directly correlated to the key.

the key is usually called keystream (the seed)
(can get generated by another key for RC4)

XOR properties

$$\begin{bmatrix} 0 \oplus 0 = 0 & 1 \oplus 0 = 1 \\ 0 \oplus 1 = 1 & 1 \oplus 1 = 0 \end{bmatrix} \quad \begin{matrix} X \oplus 0 = X \\ Y \oplus 1 = \bar{Y} \end{matrix}$$

↳ Knowing the cyphertext and the plaintext is enough to retrieve the key

$$P \oplus K = C$$

$$\underbrace{P \oplus P} \oplus K = P \oplus C$$

$$0 \oplus K = P \oplus C \Rightarrow K = P \oplus C$$

↳ We need a constant stream of keys!

(sono stupido e ho fatto questi appunti in ritardo, aiuto)

CAMPI FINITI (o CAMPI DI GALOIS)

I campi finiti sono campi (che supportano quindi addizione, sottrazione, moltiplicazione, divisione e inverso) che hanno un numero FINITO di elementi. Tutte le operazioni sono CHIUSE, nel senso che qualsiasi op. tra due numeri di campo produce un risultato interno al campo stesso

un gruppo moltiplicativo è un sottogruppo rispetto alla moltiplicazione degli elementi invertibili di un campo. Non è incluso l'elemento 0 del campo.

Son classificati

- ogni campo finito ha p^m elementi, con p numero primo e m un intero positivo
- per ogni primo ed intero esisterà un solo CAMPO $GF(p^m)$

per $m = 1$

Si parlerà di campi primi, e le sue operazioni seguiranno le regole dell'ARITMETICA MODULARE modulo p.

L'aritmetica modulare, o anche "dell'orologio" a causa delle sue proprietà, parlerà di CONGRUENZA.

due interi a, b sono CONGRUENTI (modulo n) se $|a-b|$ è multiplo di n Si scrive

$$a \equiv b \pmod{n} \quad \text{s.se} \quad \exists k \in \mathbb{Z} \mid a-b = k \cdot n$$

ad esempio, nel caso dell'orologio ciclico, si dice che le 15 sono anche le 3 del pomeriggio. $15 \equiv 3 \pmod{12}$.

inversioni

dato un elemento $a \in GF(p)$, allora a^{-1} deve soddisfare il fatto che $a \cdot a^{-1} = a^{-1} \cdot a \equiv 1 \pmod{p}$.

L'inverso può essere calcolato difficoltosamente con l'algoritmo Euclideo esteso.

per $m > 1$

Si parla di Extended Fields.

We'll be talking specifically about $GF(2^n)$, since we'll talk about bits.

Will have element representation; the element of $GF(2^m)$ are polynomials: $a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \dots + a_1 x + a_0 = A(x)$

we can say that $A(x) \in GF(p^m)$, $a_i \in GF(p)$ $i = 0, \dots m-2$

$GF(2^3) = a_2x^2 + a_1x + a_0$ · $\{0, 1, x, x+1, \dots x^2+x+1\}$

How to compute with these elements?

(+) $C(x) = A(x) + B(x) = \sum_{i=0}^{m-1} c_i x_i$ $\Rightarrow$ $c_i \equiv a_i + b_i \pmod 2$

the coefficient are computed in $GF(p)$ using congruence rules p.

In binary, the operation is the bitwise XOR, no carry.

(·) Multiplication works differently, because we need to stay in field! We need to use the congruence, to reduce the polynomial. it's irreducible if it doesn't factor over $GF(p)$

ex. irreducible polynomial for $GF(2^3)$: $P(x) = x^3 + x + 1$

DEF: Let $A(x), B(x) \in GF(2^n)$ and let $P(x) = \sum_{i=0}^{n} p_i x^i$ $p_i \in GF(2)$ be an irreducible polynomial. Then:

$$C(x) = A(x) \cdot B(x) \pmod{P(x)}$$

Ex $\underbrace{(x^4+x^3+x+1)}_{A \cdot B} : \underbrace{(x^3+x+1)}_{P(x)} = x+1$

$\hookrightarrow x^2 + x$ remainder $= A \cdot B \bmod P(x)$

For every field $GF(2^n)$ there are more than one irreducible polynomial.

To multiply two polynomials we NEED $P(x)$, or it would have different results.

For AES encryption, which uses $GF(2^8)$

$P(x) = x^8 + x^4 + x^3 + x + 1$

(a^{-1}) to work with inverses we need to find $A^{-1}(x)$ such that

$$A(x) \cdot A^{-1}(x) \equiv 1 \pmod p$$

Expansion of Euler's Theorem for RSA scheme

if p and q are primes $\varphi(pq) = (p-1)(q-1)$

pf: Since in $[1, pq-1]$ there are $p-1$ multiples of q and $q-1$ multiples of p we have that

$$\varphi(pq) = \underbrace{pq-1}_{\text{all integers}} - (p-1) - (q-1)$$

$$= (p-1)(q-1)$$

So, by Euler's theorem:

$$\forall x \in \mathbb{Z}^*_{pq} \quad x^{\varphi(pq)} \bmod pq = x^{(p-1)(q-1)} \bmod pq = 1$$

$$\Rightarrow \boxed{x^{\varphi(pq)} \equiv 1 \pmod{pq}}$$

- we want to choose e s.t. $\gcd(e, \varphi(pq)) = 1$ (e is relatively prime to $\varphi(pq)$) and $1 < e < \varphi(pq)$
- we choose d s.t. $de \equiv 1 \pmod{\varphi(pq)}$

d is multiplicative inverse of e

$$\Rightarrow ed \bmod \varphi(pq) = 1$$

$y = x^e$ then $y^d \cdot x^{ed} = x^{1+c(p-1)(q-1)} = x \cdot x^{[(p-1)(q-1)]^c} = x$

Bezout's Identity
given $x, y \in \mathbb{Z}$ $\exists a, b \in \mathbb{Z}$ s.t.
$ax + by = \gcd(x, y)$

Cl. $m^{ed} \equiv m \pmod{pq}$

pf: By Fermat Little th. $a^{p-1} \equiv 1 \pmod{p}$ $\forall a \in \mathbb{Z}$ p prime

for Bezout $ed - 1 = h(p-1) = k(q-1)$, we need to see if they are congruent mod p and mod q separately (chinese remainder theorem)

if $m \equiv 0 \pmod{p}$

m multiple of p $\Rightarrow$ m^α also multiple of p

if $m \not\equiv 0 \pmod{p}$

$m^{ed} = m^{ed-1} m = m^{h(p-1)} m = (m^{p-1})^h m = 1^h m \equiv m \pmod{p}$

Shannons Theorem

the keystream is as long at least as plaintext to get perfect cypher.

proof is by contraddiction

Assume keys are smaller than the messages.
We have C_0 so that $Pr[C_0] > 0$.

For some key K consider $P = D_K(C_0)$. There exists at most l keys of such messages.
We choose P_0 such that its not of the form $D_K(C_0)$ - n-l messages exist like that ! -

$$\hookrightarrow Pr(C_0 | P_0) = 0$$ ↳

OTP

In OTP we have a stream cypher, in which the key MUST NOT be reused. If the key is truly random, then it's perfect.

Stream cypher of nowadays take inspiration from OTP, although being different.

Attack Models

- Eavesdropping
- Known plaintext
- Choosen plaintext ↳ there is the adaptive attack in which the next plaintext will be chosen based on the cyphertext

- physical access attack
- "physical" modification of messages.

Network security: private communication in a public world

↳ Textbook

Wikipedia goes in detail.

STREAM VS. BLOCK CYPHERS

Stream cyphers are inspired to OTP, mostly relying on a way to generate a key in a secure way, using a SEED (and optionally using the information on the previous bytes to increase entropy):

- only the seed - synchronous.
- both key and first i-1 bytes of cyphertext - asynchronous.

Some examples:

- enigma (WWII - Nazis used it)
- A5 - GSM (lots of version existed, A5/2 weakened)
- WEP (Wireless Encryption Protocol)
- RC-4 (Ron's Code)

[SECURITY BY OBSCURITY DOES NOT WORK !!!]

RONALD RIVEST IS A VERY GOOD MATEMATICIAN. RC-4 was kept a secret. The US decided that exporting cryptography (considered weapons) was illegal. Now it's illegal against "bad" countries.

RC-4

A very fast key generator (for 1 byte of output it requires 15-16 bytes). It's synchronous. The sequence will repeat, but after a very long time.

- 1°: INITIALIZATION (generate permutations of first 255 numbers, randomly)
- 2°: KEY-STREAM GENERATION: (at every iteration it generates the byte to be XORed)

BLOCK CYPHER

differently from a stream cypher, a block cypher uses a block P of a fixed size (& fixed). The key should be of a fixed number of bytes.

the block has usually a small size (the most used algorithm uses 128 bits)

How should we encrypt every block of a file.

SOME EXAMPLES

- DES, 3-DES: very well known, uses 56-bit key. Too weak! 3-DES is 3 permutations of DES (3 different keys) E : E(D(E(P)))
- RC-2 : used in IBM computers. Vulnerable to 2^{34} chosen plaint. its broken
- IDEA uses 128-bit key. Pretty strong, weaker var. broken
- blowfish uses a variable-size key (from 32 to 448 bits)
- RC-5 uses variable block size and variable key size, with numbers of permutation rounds.
- AES (Advanced Encryption Standard) uses 128-bit block and 128/192/256-bit key size. Golden standard AES has a round kind of life

AES - Advanced Encryption Standard

- Symmetric block cypher
- key lenghts: 128, 192 or 256 bits

- block lenght 128 bit
  arranged in 4x4 matrix of bytes

$$\begin{matrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{10} & A_{11} & A_{12} & A_{13} \\ A_{20} & A_{21} & A_{22} & A_{23} \\ A_{30} & A_{31} & A_{32} & A_{33} \end{matrix}$$

bytes viewed in $GF(2^8)$

the matrix is called STATE, at first its the plaintext and in the end the state is the output. It's IN-PLACE.

=

key is 128, 192 or 256 bits
the cypher key layout is in matrix 4 by $n/32$ matrix of keys. The remainder of bytes are generated by the key itself

the high-level code can be represented as

- AES (state, key)
  - keyExpansion (key, expandedKey)
  - AddRoundKey (state, expandedKey[0])
  - for (i=0, i<R, i++) do
    Round (state, expandedKey[i])
  - Final Round (state, expandKey[R])

AES uses 10 rounds, can be simplified and still make it resistant, with 6 rounds:

- the secret key is expanded from 128 bits to 10 ROUND keys, 128 bit each
- each round changes state, then XORs the round key

The inverse operation is infeasible without the secret key

OPERATIONS IN 1 ROUND

One round does 4 operations

1: substitution using a lookup table S-box
$A_{ij} \leftarrow A_{ij}^{-1}$
NON LINEAR in $GF(2^8)$
if $A_{ij} = 0$ don't change

2: Cyclic shift of rows (Rotation)
1st isn't rotated, 2nd is rotated once, twice, 3 times

3: Mix colums
Every state column is considered as a polinomial over $GF(2^8)$
multiply with invertible polynomial

4: AddRoundKey bits xORED from round key

KEY EXPANSION

Generate a different key, per round, expanding from its original bit lenght

USE GALOIS FIELD !!! What it's that, really, needs to be seen on internet (haven't capito one cazz)

preRound with key xoring

9 SAME ROUND

↳ 10th round doesn't have mixColums

1 SubBytes
we do the inverse bytes, using a substitution bytes

2 ShiftRows
we shift the rows n times as is their row number

3 MixColumns
Vertical columns are polinomials of Galouis field, and we multiply it with a fixed polinomial.
We substitute the column with the result

4 AddRoundKey
we xor the round key

Rijndael is the one who made the encryption protocol

AES is the standard !

Key Schedule

Construct entropy to create more keys from the input key

We expand the key matrix, completing it with Rot of the last column then doing subBytes and then XORRING with Rcon. This is for the first column of the new ROUND key.

The other 3 columns are just XOR of the last analyzed column with the previous round

Modes of operation of Block Cyphers

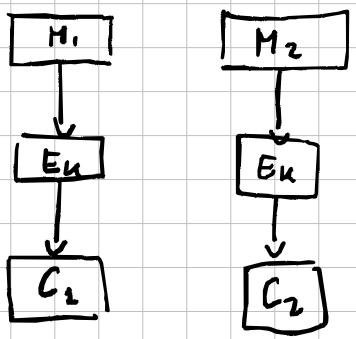
One block is small, if we want to encrypt a file that has multiple blocks it can be an issue

Other modes

ECB (Electronic Code Book)

Every block is encrypted with the SAME key!
Natural, but dangerous: if we have two blocks of equal content, it will be visible in the cyphertext. Useful for information gathering.

It should be used for experimenting only!
Useful for parallel encryption
We can see REPETITIONS

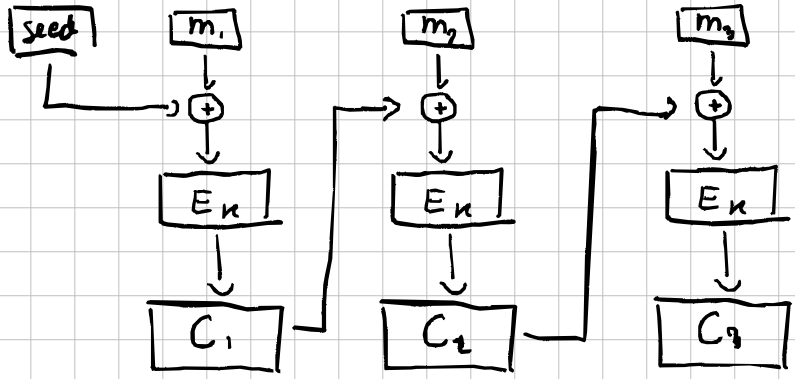


if $M_1 = M_2$
$\Rightarrow$ $C_1 = C_2$

INFORMATION

CBC (Cipher Block Chaining)

Message is subdivided in blocks. The cyphertext of the previous block is XORED with the plaintext block, and then encrypted with same key



the seed is used as starting point, and should be random but isn't secret. If using the same seed and we send file with same key, then cyphertex will be same.

cannot be parallelized in encryption or decryption

decryption

it's basically the same, but the block ciphered will be xored with the plaintext of the next blocke
Decryption is faster than encryption, can be parallelized.

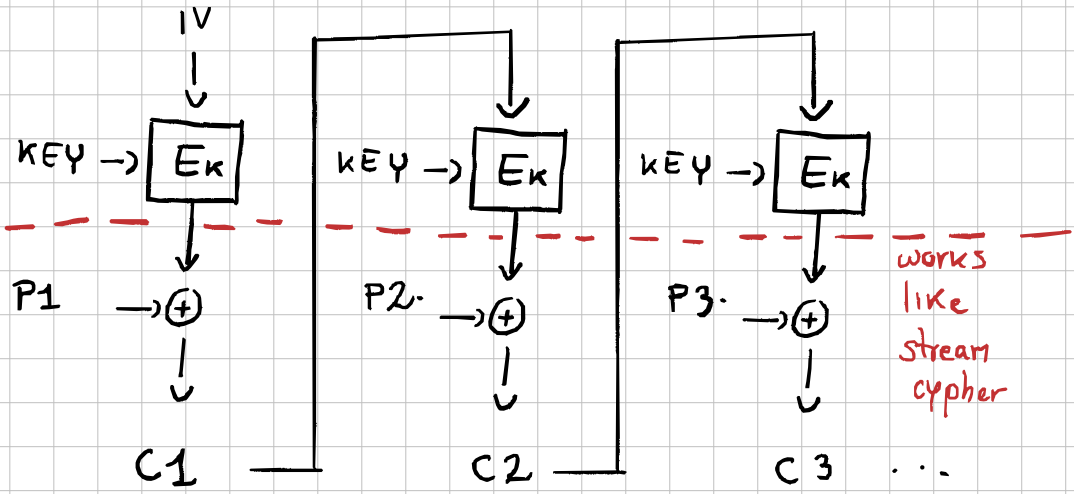
Changing a bit of the cyphertext can give you garbled block i, but the next one will get changed in a predictable way, and all of the others are untouched!

HIDES PATTERNS!

CFB Cypher FeedBack

Like CBC, makes block cypher into an asynchronous stream cipher.

It's used to resyncronize after error.



Still asynchronous! Plaintext is XORRED to stream of bytes (coming from previous cypher encryptions

Can't be encrypted in parallel, but can be decrypted parallel by precomputing decryption stream

to decrypt we use the same principle, but while STILL USING ENCRYPTION at decryption

PERFECT CYPHER

Given a plaintext space $\{0,1\}^n$, given a ciphertext C the probability of a k existing such that $D_k(C) = P$ for any plaintext P is EQUAL to the probability that P is a plaintext

$$Pr\left(D_k(C) = P\right) = Pr(P) = Pr(P \mid C)$$

INTEGRITY

We're now interested in ensuring integrity of a message. we don't need confidentiality, so we'll deal with plaintexts.

Specifically we want to be sure if a message has been modified.
We'll send then, a tuple $(m, A_K(m))$, with A_K being the authentication tag of the message, and K the authentication key (different from the eventual encryption key !!!)

We'll run V_K as our verification algorithm

$$V_K(m, A_K(m)) = \text{"accept"} / \text{"reject"}$$

$A_K(m)$ is usually denoted as the MESSAGE AUTHENTICATION CODE, authentication tag

Properties

- We don't want another legal pair $(m, A_K(m))$ to be constructed, even after seeing lots of pairs.
- Output should be as short as possible, to be more efficient, and to have a standard length.
- The MAC function shouldn't be 1 to 1 (otherwise it should've had same cardinality) so it shouldn't be possible to get m from $MAC_K(m)$

Adversary data:

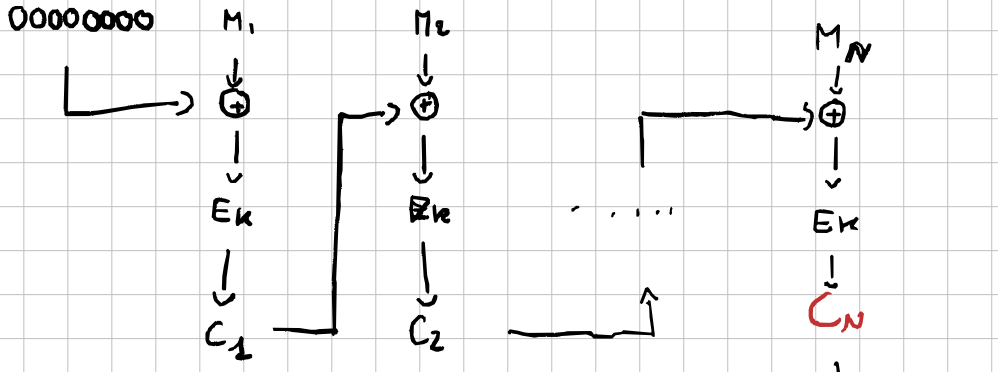
- MAC algorithm
- known plaintext
- chosen plaintext

$\Rightarrow$ given n legal pairs $(m_i, A_K(m_i))$ create new legal pair $(m, A_K(m))$, even if "meaningless"

MAC used:

- Based on CBC block encryption
  - slow → MAC-CBC-ELB
  - uses block cypher
- Based on cryptographic hash functions
  - fast
  - export-safe

CBC block mac → we encrypt ⊕ XOR message bytes



Serialized encryption that needs to be done every time we check

↳ SLOW!

This is the Authentication Tag

$MAC_K(M) = C_N$

Fixed CBC-MAC is more secure than variable CBC-MAC!

SAFE! E_K should transform input to pseudo-random like output $[Pr(P) = Pr(P|C)]$

Variable-lenght messages are INSECURE! if attacker has (m,t) and (m',t') it can create VALID m''

$m'' = \underbrace{m \,||\, (m_1' \oplus t)}_{\text{this creates } m_1'} \,||\, m_2' \,||\, \dots \,||\, m_v'$

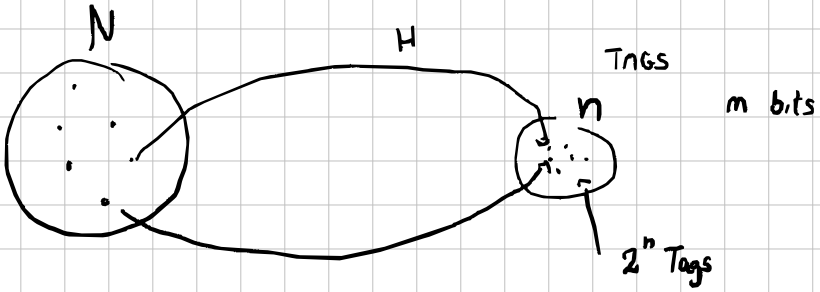
this is because we would just be "continuing to encrypt" and $t'' = t'$

$m_n \rightarrow t$ so t being "the iv of m_1'" $(m' \oplus t)$ [constr] ☺

we here

t

HASH FUNCTION



The function is not 1 to 1, so it's necessary that some messages point to the same tags.
Collisions are bad!!!

↳ average numbers of collisions are N/n

2 type of hashing function
- KEYED
- UNKEYED → these are the hashing functions

Collision resistancy

a hash function $h: D \to R$ is weakly collision resistant for $x \in D$ if it is hard to find x' such that $x \neq x'$ and $h(x) = h(x')$

a hash function $h: D \to R$ is strongly collision resistant if it is hard to find x, x' such that $x \neq x'$ and $h(x') = h(x)$

Strong implies weak. Proof: ! weak → ! strong
- Given h, suppose we find alg A_h s.t. $A_h(x) = x'$ and $h(x) = h(x')$
- we construct poly alg. B_h s.t. $B_h() = (x, x')$ s.t. $h(x) = h(x')$
  - randomly choose x
  - return $(x, A_h(x))$

We want to find a strong collision resistant hash function.

Birthday Attack

given function f find 2 different inputs x_1, x_2 such that $f(x_1) = f(x_2)$

e.g.

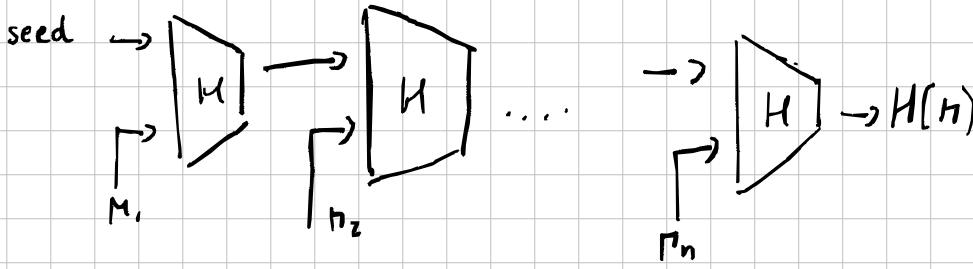
if 64-bit function, there are 1.8×10^{19} different outputs. if the are all equally possible, then it would take 5.98×10^9 tries to find collision. LITTLE TRIES ON PCs.

A criptographic hash function should be strongly collision-resistant, and hard to invert.

Should be hard to invert, since otherwise a threat could try a dictionary attack

Merkle-Damgaard

Approach to hash variable-lenght blocks

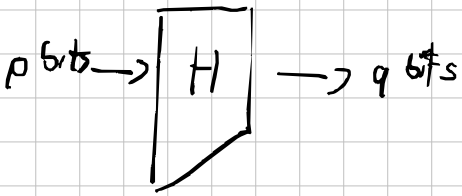


$MAC_K(M) = h(K \| m)$ NO! Broken, adversary can add blocks without problems and still have valid messages.

$MAC_K(M) = h(m \| K)$ birthday paradox exploitable: if adversary finds collision messages will collide for every key

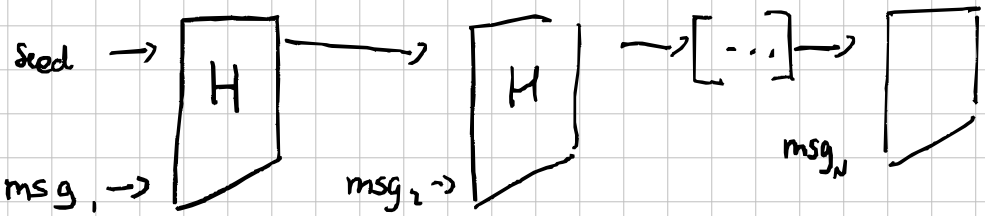
$MAC_K(M) = h(K \| m \| K)$ ok!

HASHING FUNCTION



Extending to longer strings

→ Were using block hashing function, since we have a block as an input.



Merkle - Damgård construction, input is subdivided in 2 parts (q size seed, p-q messages)
An adversary looking for the transmission, and having some pair can do message concatenation attack

We can use Sponge construction with SHA-3 but we won't talk about its workings.
1) Absorbing step
2) Releasing step

A popular approach is based on HMAC (kind of a "meta" hashing function, with input key, hashing function, and message, in which we have the function itself as input.

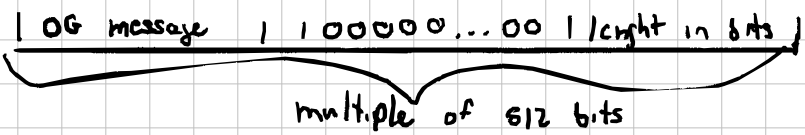
SHA-1 approach is to make lots of rounds (80) to single block, making permutation in itself

SHA-1 basic

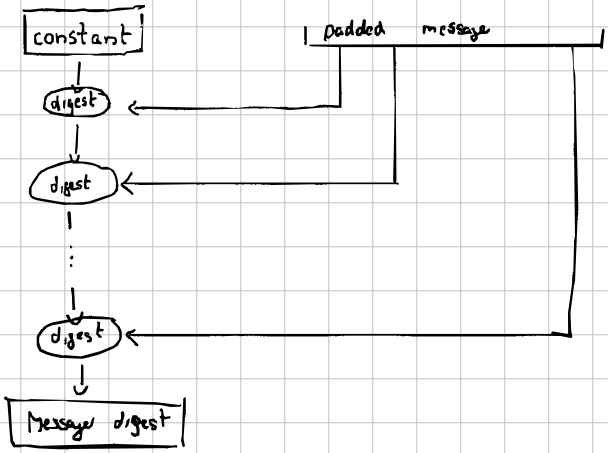
Similar to MD4 & MD5

- $|message| \leq 2^{64}$ → original message padded
- $|digest| \leq 2^{160}$
- $|block| = 512$ bits

SHA-1 is deprecated !
↳ Google stopped using it long ago



160-bit message digest consists of 5 32-bit words.
A | B | C | D | E



we have 80 rounds, each one modifies buffer (A,B,C,D,E)

$$(A,B,C,D,E) \leftarrow (E + f(t,B,C,D) + (A <<< 5) + W_t + K_t, A, B <<< 30, C, D)$$

- t round number
- f non-linear function, round dependant
- W is 32-bit word, expands 16 words into 80 words its 512-bit long
- K_t constant

SHA-1 Dismission

Collision attacks became too computationally simple, so Google Chrome dismissed SHA-1 certificates on 2017.

↳ MD5 completely reversed
↳ SHA-1 broken by birthday attacks.

HMAC

Outputs MAC by

$$HMAC_K(m,h) = h(K \oplus opad \,||\, h(K \oplus ipad \,||\, m))$$

- opad : outer padding
- ipad : inner padding

message hashed with ipad-ded key.

RFC 2104, it's a way to authenticate hash. HMAC can be forged if and only if H fuction is flawed.

Still flawed by birthday paradox, but even with a valid block collision. The key won't still be valid, with VERY HIGH probability.

this is part of
- FIPS standard
- SSL/TLS
- WAP → WTLS
- IPSec

$$\begin{bmatrix} opad = 0x5c5c\ldots5c5c \\ ipad = 0x3636\ldots3636 \end{bmatrix}$$

Birthday paradox still holds true, adv. needs $2^{n/2} + 1$ messages with same key to have probability > 0.5.

IMPORTANCE OF KEY ROTATION

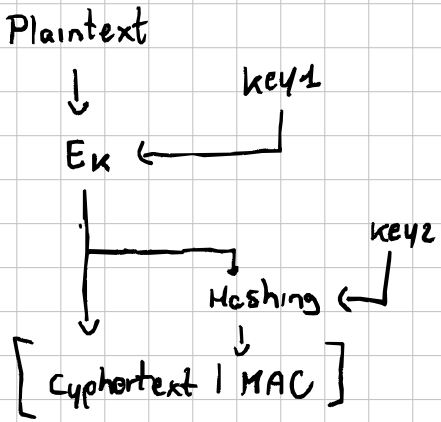
AUTHENTICATED ENCRYPTION

Way to incorporate CONFIDENTIALITY, INTEGRITY & AUTHENTICITY with small header for integrity verification

Different ways:

- Encrypt-then-MAC — EtM [see below]
- Encrypt-and-MAC — E&M [encrypts and hashes with same key]
- MAC-then-Encrypt — MtE [encrypts (Plaintext || MAC)]

EtM



[we may want to do AE with associated data that we don't want to encrypt]

GALOIS COUNTER MODE

Way to · using GALOIS FIELD GF(2^{128})

we'll use hash key

$$H = AES(key, 0^{128})$$

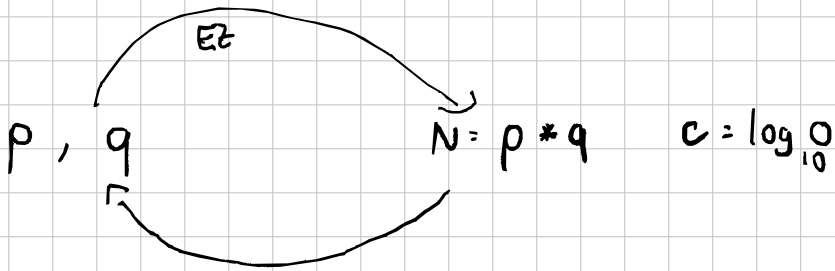
multiplication of H with associated data blocks, reduced modulo

$$x^{128} + x^7 + x^2 + x + 1$$

Creates Authentication tag

ONE-WAY FUNCTIONS

E.g. Hashing function
Usable in public key cryptography.



Discrete Log

- Let G be a finite cyclic group with n elements and g a generator of G

generator = $g \in G \mid g^x \in G \quad \forall x \in G$

let y be any element of G

$y = g^x$, and x the minimal nonnegative integer satisfying the equation

if we have y and g, finding x is non trivial $\Rightarrow$ $x = \log_g \bmod n$

NP problem !

we can use the Euler's theorem

Modular Exponentiation

$$c \bmod m = (a \cdot b) \bmod m$$
$$= ((a \bmod m) \cdot (b \bmod m)) \bmod m$$

Euler's Theorem

$$b^e \bmod p = (b \bmod p)^{e \bmod \varphi(p)} \bmod p$$

(b and p coprime)

DEF: One way functions

A function $f: \{0,1\}^* \to \{0,1\}^*$ is one way if f can be computed by a polinomial time algorithm, but for every polinomial time algorithm A, for every polinomial $p(n)$ and all sufficiently large n

$$Pr\left[\left(f(A(f(x)))\right) = f(x)\right] < \frac{1}{p(n)}$$

it should be "hard to invert" in the average case

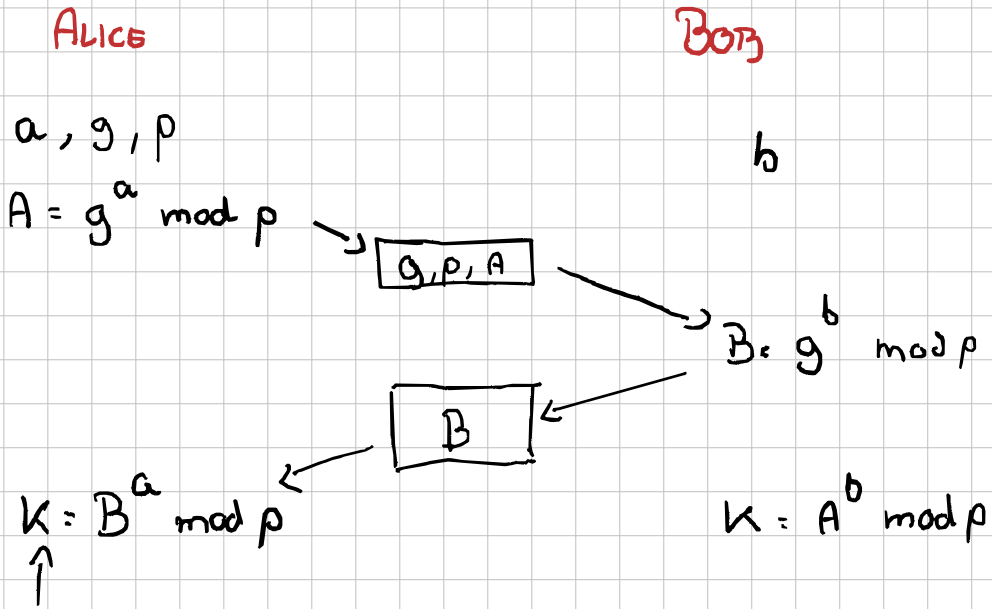
(assuming x is choosen from the uniform distribution on $\{0,1\}^n$, and the randomness of A

Do they truly exist?

If yes, $P \neq NP$

Public Key Exchange

can be exchanged securely via public key encryption



K will be symmetric encryption key!

$$A^b \bmod p = (g^a \bmod p)^b \bmod p$$

$$= g^{ab} \bmod p =$$

$$= (g^b \bmod p)^a \bmod p$$

$$= B^a \bmod p$$

$$g^{abc} \bmod p$$

- a prime p and element g (possible generator)
  → use $p = 2q + 1$ with q also prime

- Alice choose a in range $a \in [0, p-1]$
- Bob chooses b random $b \in [0, p-1]$

they send each other

$g^a \bmod p$ $\qquad$ $g^b \bmod p$

$g^{ab} \bmod p$

we have SHARED KEY!

vulnerable to RAINBOW TABLES!!!

Perfect forward secrecy! Compromission of shared key doesn't compromise former, or subsequent, shared keys!

Diffie Helmann does NOT provide authentication, vulnerable to MITM attacks!

We can extend the Diffie Helmann to three-way easily

$g^{abc} \bmod p$

Multiplicative Group Z_{pq}^*

Let p and q be two large primes. We'll denote $N = pq$

$Z_N^* = Z_{pq}^*$ contains all integers in range $[1, pq-1]$ relatively prime to both p and q.

In $[1, pq-1]$ we have $(q-1)$ multiples of p and viceversa we have

$$|Z_{pq}^*| = \varphi(pq) = pq - 1 - (p-1) - (q-1) = (p-1)(q-1)$$

↳ by eulers theorem $\forall x \in Z_{pq}^*$

$$x^{\varphi(pq)} \bmod pq = 1$$

Let e be integer $1 < e < (p-1)(q-1)$. If e is relatively prime to $(p-1)(q-1)$ then exponentiation is ONE to ONE

proof by Bezout's Identity

$$xe - (p-1)(q-1) = 1 \Rightarrow$$

$$\Rightarrow ed = 1 + C(p-1)(q-1)$$

$$x^e = y \Rightarrow y^d = x^{ed} = x^{1 + c(p-1)(q-1)}$$

$$= x^1 \left(x^{(p-1)(q-1)}\right)^c = x \cdot 1^c = x$$

x NEEDS to BE $\in Z_{pq}^*$!!!

RSA

Rivest
Shamir
Adelman

- Let $N = pq$ (with p, q being two large primes)
- Choose e s.t. $\gcd(e, \varphi(pq)) = \gcd(e, (p-1)(q-1)) = 1$
- find d s.t. $ed \equiv 1 \mod \varphi(n)$

→ Public key (e, N)
→ Private key (d, N)

Encryption $M \in \mathbb{Z}_{pq}^*$: $M^e \mod N = C$

Decryption $C \in \mathbb{Z}_{pq}^*$: $C^d \mod N = M$

RSA is slow

Usually used to encrypt a symmetric key (or also called session key), that then its used to encrypt a file.

Symmetric key is long 128 bits, so it's okay for

Computing multiplicative inverse

$p = 47$ $N = 2773$ choose $d = 157$

$q = 59$ $\phi(N) = 2668$

Bezout Identity $au + bv = d$ $d = \gcd(a, b)$

we need to find e s.t.

$-1 \times 2773 + e \cdot 157 = 1$

given integer x, y s.t. $\gcd(x, y) = 1$ find multiplicative inverse of $x \pmod y$

assume w log $x \geq y$

traditional Euclid's algorithm for computing $\gcd(x, y)$

$r_n = r_{n-2} \% r_{n-1}$ $r_{n-2} = x$, $r_{n-1} = y$

and Bezout's identity $u_n x + v_n y = 1$

we will need to reconstruct the algorithm DO IT

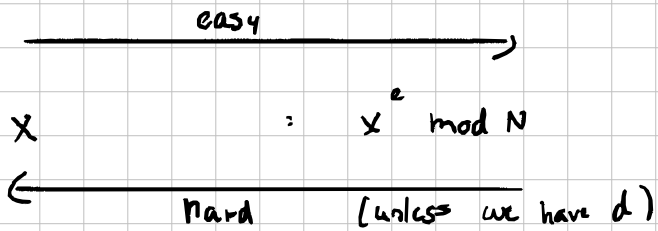
(what the fuck does it mean ???)

TRAPDOOR ONE WAY FUNCTION

Def: $f: D \to R$ is a trapdoor OWP if there is a trapdoor s such that:

- → without knowledge of s, f is a one-way function
- → given s, inverting is easy

EXAMPLE: RSA is trapdoor OWF, given s the private key.



RSA is a "collection" of trapdoor OWF, since every set of natural numbers could make a function in itself.

Let I a set of indices, a collection of one way trapdoor functions Φ is a set of functions f parameterized by I

if:

$$\forall i \in I \mid \exists f_i \mid f_i \in F \wedge f_i : D \to R_i \text{ is trapdoor OWF}$$

Attacks on RSA

It's not always robust:

→ factorization of $N = pq$, maybe using rainbow tables. Hard, with precautions:
- take p, q very large
- take them enough far apart
- chose them s.t. p, q have large prime factors.

→ may need to preprocess informations.

FACTORING

given n, e p and q easy to compute d
given n, e:
→ if we factor n then it can compute $\phi(n)$ and d

if you factor n then we can break RSA

EASY TO DECRYPT MESSAGE

Ex $M^e \bmod n$ with $M=0 \to C = M$
with $M=1 \to C = M$

- if $M = 0, 1, \dots, n-1$ then $RSA(M) = 1$

(e is odd ≥ 3, hence $(n-1)^e \bmod n = n-1$
because $(n-1)^2 \bmod n = 1$ $\quad n^2 - 2n + 1 \bmod n = 1$ (0 0)
$e = 2j-1 \Rightarrow \left(N-1^{2j}\right)^k \cdot (N-1) \bmod N$

} may use salt

- if both M and e are small enough
$M^e < N$
$M^e \bmod N = M^e$

– we may add non-zero bytes to avoid small messages

} add non zero bytes to avoid small messages.

we should add invertible pre-processings!

- Small e

if we have 2 messages such that
$c_1 = m^3 \bmod n$ and $c_2 = (m+1)^3 \bmod n$
$m = \frac{c_2 + 2c_1 - 1}{c_2 - c_1 + 2}$ (given $c_1, c_2, n, e = 3$)

choose BIG exponent e

CHINESE REMAINDER THEOREM

suppose $n_1, n_2, \ldots, n_k$ are positive integers such that they are pairwise coprime.
Then for any given $a_1, a_2, \ldots, a_k$ there exist an integer x solving system of simultaneous congruences

$$X \equiv a_i \pmod{n_i} \quad i = 1, 2, \ldots, k$$

Furthermore, all solutions x are congruent modulo multiplication $N = n_1 n_2 \ldots n_k$

es.

$e=3$, we send same message 3 times with different public keys. $(3, n_1)(3, n_2)(3, n_3)$

Attacker knows ciphers and public keys

Attacker can compute (with chinese remainder theorem)

$$m^3 \bmod (n_1 n_2 n_3) = m^3$$

given that

$m < n_i \quad i = 1,2,3$. then $m^3 < n_1 n_2 n_3$

SMALL MESSAGE SPACE

if message space is small, attacker can do a precomputation of all possible messages in precomputation.

sol: add random strings.

MULTIPLICATIVE PROPERTY

IF $M = M_1 * M_2$ then $M \bmod N =$
$= (M_1 * M_2) \bmod N = ((M_1 \bmod N) \cdot (M_2 \bmod N)) \bmod N$

can be generalized if $M = M_1 * M_2 * \dots * M_k$

solution: padding on short messages!

CHOSEN CYPHERTEXT ATTACK

we want to decrypt $C = M^e \bmod n$
→ adv computes $X = (C \cdot 2^e) \bmod n$

adv uses X as chosen cyphertext and asks the oracle for $Y = X^d \bmod n$

$$X = (C \bmod n)(2^e \bmod n) = M^e \bmod n$$

=

we have $C = M^e \bmod n$
we choose X and compute $C' = CX^e \bmod n$ and ASK FOR DECRYPTION

$$C'^d = \underbrace{C^d (X^e)^d}_{X} = MX \bmod n$$

TIMING ATTACK!

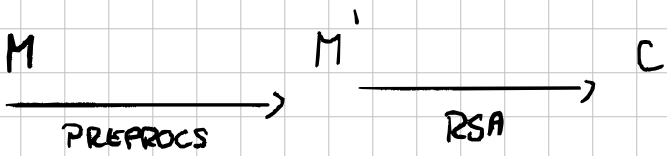
↳ SOL: add random steps!

IMPLEMENTATION of STANDARD RSA!

Textbook implementation of RSA is NOT SAFE:

- many vulnerabilities
- doesn't satisfy criteria

We NEED TO PREPROCESS m so that we have m' with the exact content information!



PUBLIC KEY CRYPTOGRAPHY STANDARDS (PKCS)

We need a PKI, so we had creation of standards by RSA Security - a company -

PKCS (1-15):

#1 : standards to send messages through RSA

m = 0 || 2 || at least 8 non-zero bytes || 0 || M

- 0 → m < N
- 2 → 1 - digital key, 2 - message
- at least 8 non-zero bytes → SALT.

PKI is set of Public Key Cryptography Standards

#2 describes NON repudiation (obsolete)

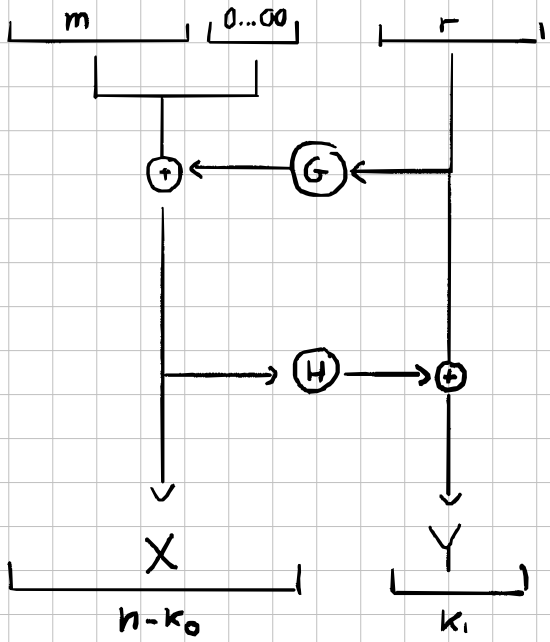
Optimal Asymmetric Encryption Padding

Padding scheme often used for RSA encryption, based on pair of random ORACLES, G, H to do plaintext processing prior to encryption.

- → makes secure for chosen plaintext / ciphertext attacks.
- → Prevents partial decryption of ciphertexts, needs everything to recreate message

All-or-nothing approach

n = number of bits in RSA modulus
k_0, k_1 : fixed lenghts chosen by protocol
m : plaintext of size $n-k_0-k_1$
G, H cryptographic hash functions
r : random string.



TO RECOVER msg

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

$$m \,||\, 000 = X \oplus G(r)$$

Current version of PKCS-1 (v.2.2) uses OAEP, but supports early versions for retrocompatibility

- → RSA-OAEP
- → RSA-PKCS1-V1_5

MAC vs Signatures

Mac works with a key k used to authenticate message of one another.
Misses non-repudiation, since both parties could authenticate both messages

Even default diffie helman is insecure;
Bob may find random message R, encrypts $S = E_A(R)$ and then sends (R, S) and it would be a valid signature. WE CANNOT use public key for signature creation!

FORGERY

It's the ability to create pair consisting of a message m and a signature σ that is valid for m, where m has never been signed in the past by the legitimate signer

- existential forgery

  it's when the adversary is able to create a pair (m, σ) which is valid. m doesn't need to have any meanings.
  It's the WEAKEST form of forgery, but we'll try to create an algorithm which is "existentially unforgeable"

  implies

- selective forgery

  adversary creates a message/signature pair (m, σ) which m is chosen by the attacker prior to attack, maybe for interesting mathematical property.
  m must be fixed prior to the attack

  implies

- universal forgery

  an adversary can arbitrarily generate a message/signature pair (m, σ) for every m -chosen or random-

Some problems with pure DH algorithm

RSA is multiplicative, so this means that

$$D_A(M_1, M_2) = D_A(M_1)\, D_A(M_2)$$

we may create forged messages by having signed messages before it

SOLUTION: Hash First! [and then we RSA_D the hash]

to sign message M, Alice first computes $y = H(m)$ and $z = D_A(y)$ and sends (z, M) to Bob

↳ To verify Alice Signature bob decrypts $y = E_A(z)$ and then compares $H(m) = y$

E_A is public, so everyone can verify authenticity!

Used schemes in practice

- RSA ~ Everybody can verify hash - DKCS-5 specifies signature! → can be signed with openssl
- El Gamal Signature Scheme
- The DSS - digital signature standard, by NIST in 1994

El-Gamal Signature Scheme

Todo Tomorrow

Generation:

- pick a prime p of lenght 1024 bits such that DL in $\mathbb{Z}_p^*$ is hard
- let g be a generator of $\mathbb{Z}_p^*$
- Pick $x \in [2, p-2]$ rando
- Compute $y = g^x \mod p$
- public key (p, g, y)
- private key x

SIGNING

Let m be hash of M $\quad m = H(M)$

→ pick k in $[1, p-2]$ s.t. $\gcd(k, p-1) = 1$

→ compute $r = g^k \mod p$

→ compute $s = (m - rx)k^{-1} \mod p-1 \quad (*)$

(if $s=0$ restart)

→ output signature (r, s)

VERIFYING

compute $m = H(M)$

accept if

$$(0 < r < p) \wedge (0 < s < p-1) \wedge (y^r r^s \equiv g^m \mod p)$$

this is because, by $(*)$ $s = (m - rx)k^{-1} \mod p-1$

$sk + rx = m$. $\quad r = g^k$ so $r^s = g^{ks}$.

$y = g^x \Rightarrow \quad y^r = g^{xr} \Rightarrow$

$g^{xr} g^{ks} = g^{xr+ks} = g^m$

Digital Signature Standard

NIST, FIPS standard

→ DSS uses SHA as hashing functions and DSA as signature
(inspired by ElGamal)

Let p be an L-bit prime s.t. DL is hard
Let q be 160-bit prime that divides $p-1$

$$p = jq + 1$$

Let α be the q-th root of $1 \bmod p$

$$\alpha = 1^{1/q} \bmod p \iff \alpha^q = 1 \bmod p$$

we'll calculate random s $\quad 1 \leq s \leq q-1$

public key $(p, q, \alpha, y = \alpha^s \bmod p)$

Signing

choose random k s.t. $1 \leq k \leq q-1$ (secret)
2 PARTS:

$$\text{PART I} : (\alpha^k \bmod p) \bmod q$$
$$\text{PART II} : (SHA(M) + s(\text{PART I}))\, k^{-1} \bmod q$$

signature $\langle$ PART 1 , PART II $\rangle$

Random Numbers

mod function is not a good random number generator!
key exchange information.

We have two principal methods of generator

Random Number Generator (RNG) uses physical phenomenon so we can get randomness.
(e.g. temperature, radioactive decadence, noisy image...)

Pseudo-Random Number Generator (PRNG) deterministic algorithm that has extra data (e.g. time) that creates long sequences of pseudo-random data, starting from a SEED!

Very difficult to distinguish RNG & PRNG.

The output of a pseudo-RNG should be long and random enough to be infeasible to guess next number (even probabilistically)

We can take random numbers from user input (mouse movements, keyboard), the time or the positioning of windows.

↳ How many bits we have of randomness?

Common mistakes

→ Using a too-small seed

a 16-bit seed may have 65535 unique numbers, easy to try them all

→ using the current time

with a 10 ms clock granularity, even knowing the time by the hour we'll have:

60 (minutes) x 60 (seconds) x 100 (tenth of seconds)
= 360000

useful to mix multiple inputs to increase possible values

Netscape 1.1 used MD5, it's NOW BROKEN!
It also uses the pid and ppid to generate randomness, which could be easily retrieved.

Another bug in openssl packages happened because of some function removing. Deletes randomness and it made it very secured. (2008)

↳ Valgrind complaining about uninitialized values.
NSA knew about this and actively exploited.

A real bit-generating random number would be better.

BSI evaluation criterias

BSI defines 4 criterias for evaluating a random number generator.

- K1: A sequence of random number with a low probability of containing identical consecutive elements (run)
- K2: Complicated (talked later)
- K3: It should be impossible for an attacker to calculate/guess, for any given subsequence, any previous or future values in the sequence, nor any inner state of generator.
- K4. It should be impossible for an attacker to calculate/guess, for any given inner state, any previous or future values in the sequence, nor any other inner state of generator.

should be one-way.

K2 requirements

monobit test

$|1| \approx |0|$

poker test

special case of X^2 (chi-squared test)

runs test

check frequency of short runs

longruns test

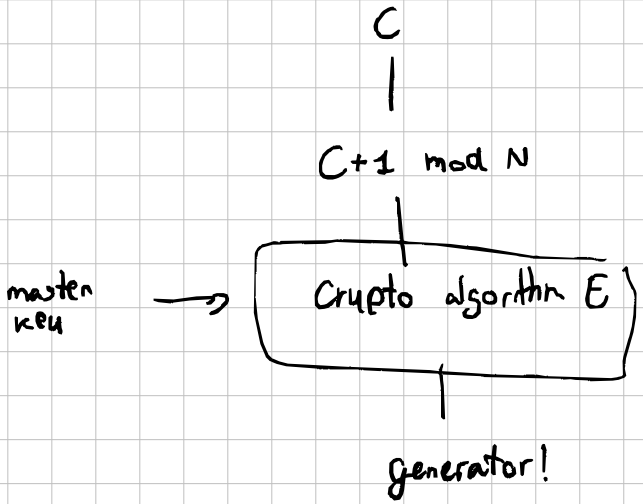
check freavency of 20000 bits. if 34 $\geq$ same bit failing.

autocorrelation test

ones and zeros shouldn't have correlation between each others.

Criptographycally Secure PRNG

Meets requirements of PRNG, and adds a next-bit test.
→ given first k bits of random sequence, there is no polinomial-time algorithm that can predict the (k+1) th bit with success probability different than 50%.



Even if the internal state has been compromised there is no way to reconstruct stream prior to compromission

PUBLIC KEY AUTHENTICATION

Based on digital signature, commercially.
We'll move to other concepts [X509]

Authentication:

based on person A wanting to prove its identity to person B, to share or access information...

Authentication ≠ Identification

↳ Providing identity

↳ Proving to be who someone claims to be

We may want to have 2-way authentication from 1-way

- we may want to use 2 times 1-way authentications. Issues! Different timings means that it can be attacked
- we need a way to do it at same time.

Authentication of People

we need to use:

- What we know (password)
- What we have (smart card)
- Who we are (biometrics)
- Where you are (address, IP, MAC)

Public key authentication

↳ Nonce

- based on the idea of using challenges (better) or timestamps digitally hashed, verifiable by public key.
  - ↳ we need a Public Key Infrastructure authority so that we have a centralized trusted server that guarantees legitimacy of the public key requested. (Impersonation attacks possible otherwise)

Just encrypting a password to authenticate is NOT ENOUGH!

↳ Replay attacks possible without the need of password knowledge

Response is sending hashed nonce with public key.

provide response to nonce, discard if already acquired or if Δ time passes

NEEDHAM - SCHROEDER

- K_{PX} public key of X
- Sig_C digital signature of C (trusted authority guarantees public key owning)

Mutual Authentication:

1: A to X (initialize connection AB)
2: X to A $\langle B, K_{PB}, Sig(K_{PB}, B) \rangle$
3: A checks signature of X, generates NONCE N and sends to B: $K_{PB}(N, A)$
4: B decrypts and asks for A's identity to C
5: C answers $\langle A, K_{PA}, Sig(K_{PA}, A) \rangle$
6: B checks signature, retreves K_{PA}, generates nonce N' and sends to a $K_{PA}(N, N')$
7: A decrypts, checks N, and sends to B: $K_{PB}(N')$

Passible MITM attack.
Based on the fact that authentication does NOT have DESTINATION.
trudy tricks A into starting conversation with T

Alice sends Trudy	
T (as A) -> B	$K_{PT}(N, A)$
B -> T (like A)	$K_{PB}(N, A)$
T -> A	$K_{PA}(N', N)$
A -> T	$K_{PA}(N', N)$
T (As A) -> B	$K_{PT}(N')$
	$K_{PB}(N')$

Reflection attack

Bob thinks it's talking to Alice !!!
FIX.
Add destination at step 7 $K_{PA}(B, N, N')$

Needham-Schroeder

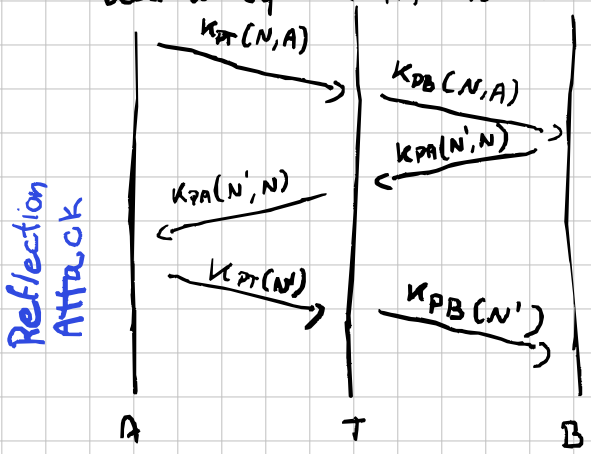
Assuming that the certificate authority is trusted, it's still attackable!

Attack to N.-S. Public Key using MITM

idea of two sessions of the protocol, in which T wants to get in the middle between A and B.

R1: authentication between A and T
R2: authentications between T (like A) and B

A starts legitimate authentication session with T, T uses this to start attack on B, faking it's identity to be A by relaying challenges received by B to A, and then reflecting response to B



trudy knows N, N' and both A and B think they are mutually authenticated

A authenticated T (correctly)
B authenticated A (but really to T)

Alice is Insider!

we can fix this by incorporating the sender in

$K_{PA}(B, N', N)$

X.509 Authentication Standard

Standard for binding public keys to objects (person, domains, ...)
- now in version 3, although fully retrocompatible.
- ↳ SSL/TLS certificates to certify identity of a server, used for TLS.

We want to have authentications in:
- one-way
- two-ways
- three-ways

NOT talking about mutual or single authentication, but of NUMBER OF MESSAGES SENT

Technical details, such as algorithms, are not part of the standard.
otherwise it should be updated every time (to avoid legacy algorithms)

One-way authentication

timestamp t_A
Session key K_{AB}
B's public key P_B
cert A is certificate of public key of A, signed by CA

integrity & authentication

$$A \rightarrow B \quad \langle cert\_A, D_A, Sig_a(D_A) \rangle$$

where D_A : $\langle t_a, B, P_B(K_{AB}) \rangle$

↳ session key encrypted with destination's pubkey

2-WAY (MUTUAL) Authentication

$A \rightarrow B$: $\langle certA, D_A, sig\text{-}A(D_A) \rangle$

$D_A = [t_A, N, B, P_B\langle K \rangle]$

$B \rightarrow A$: $\langle certB, D_B, sig\text{-}B(D_B) \rangle$

$D_B = [t_B, N', A, N, P_A(K')]$

t_A, t_B to prevent delayed messages

K, K' session keys proposed by A and B, to do Authenticated Encryption from one to other.

N, N' nonces to avoid replay attacks

criticisms:

- In the message there is no identity of sender, MITM reflection attack
- Use of P_B BEFORE seeing certificate of public key!

3-WAY (MUTUAL) authentications

Mutual authentication based on nonces, used for unsynchronized clocks (we don't use timestamp)

1 $A \rightarrow B$ $\langle certA, D_A, sig_A(DA) \rangle$ $D_A = \langle 0, N, B, P_B(K) \rangle$

2 $B \rightarrow A$ $\langle certB, D_B, sig_B(D_B) \rangle$ $D_B = \langle 0, N, A, N', P_A(K) \rangle$

3 $A \rightarrow B$ $\langle B, sig_A(N, N', B) \rangle$

Like Needham-Shroeder

Strong against both REPLAY and REFLECTION attacks

=

ISO9798-3 mutual authentication flawed,

↳ "canadian" attack

Public Key Infrastructure

- Certificates are issued by trusted Certification Authority (CA) to all users in the domain

If someone queries for a Pubkey of a user, then the CA provides it, signing it with its private key

we imply that CA is trusted. If it's not, its certificates are useless.
Certificates have an inherent expiration date.

X.509 CERTIFICATES

Kind of text key, containing a tuple <tipe.value>,
Tipes are

- VERSION : version of x509
- SERIAL NUMBER : number that uniquely identifies certificate
- SIGNATURE : specifies the algorithm, with optional parameters
- ISSUER : X500 name of the issuing CA
- VALIDITY : start - end date
- SUBJECT : entity whos key is being certified
- SUBJECT PUBLICKEYINFO : (alg, public key)
- ALGORITHM IDENTIFIER : repeats SIGNATURE
- EXTENSION
- ENCRYPTED : contains for the signature of last, but two fields

X500 names
C : country
O : <company name>
OU : <organizational unit>
CN : <common name>

CRL and OCSP

Certificate Revocation List (UIDs of revocated certificates)
Online Certificate Status Protocol obtains latest certificate status

ISO 9798-3 Mutual Authentication

Proposed protocol, revealed to be flawed

1. $B \rightarrow A$ $\quad N_B$
2. $A \rightarrow B$ $\quad Cert\,A, N_A, N_B, B, sig\text{-}A(N_A, N_B, B)$
3. $B \rightarrow A$ $\quad Cert\,B, N_B, N_A, A, sig\text{-}B(N_B, N_A, A)$

flawed, "Canadian Attack"

1. $T(as\ B) \rightarrow A$ $\quad N_T$
2. $A \longrightarrow T(as\ B)$ $\quad cert\,A, N_A, N_T, B, sig\text{-}A(N_A, N_T, B)$

   a $T(as\ A) \rightarrow B$ $\quad N_A$

   b $B \longrightarrow T(as\ A)$ $\quad Cert\,B, N_B, N_A, A, sig\,B(N_B, N_A, A)$

3 $T(as\ B) \rightarrow A$ $\quad Cert\,B, N_B, N_A, A, sig\,B(N_B, N_A, A)$

Something doesn't add up: on the first communication T (as B) sends N_T, while at the last step T (as B) sends N_B back

$\rightarrow N_T \neq N_B \leftarrow$

PASSKEYS

Potential challenges

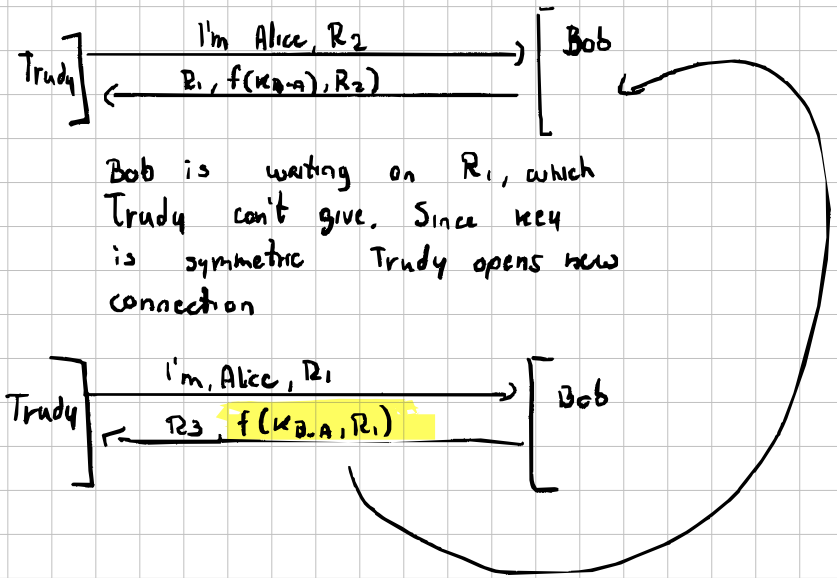
- compatibility with existing system
- user adoption
- device dependent → loss/theft handling.

AUTHENTICATION

Can be mutual if both parties send challenges to one another.

A	starts auth
B	sends nonce R_1
A	answers with encrypted nonce, sends R_2
B	answers with encrypted nonce

Prone to reflection attack!



How to prevent ?

- Use different key
  → maybe COUNTER up
  K_{BA} ; $K_{BA} + 1$

- Use a different challenge from initiator

- Let the responder start the challenge!
  [not perfect, still prone to offline attack by having R_1 and $f(K_{BA}, R_1)$, and tricking connection to malevolous user]

=

Having a shared key is imperfect, since parties need to maintain QUADRATIC set of key - one for each pair)

TRUSTED PARTY

Scenario: Users share key with a trusted authority C (key distribution Center)

A wants to connect to B, they ask C for a (Temporary!!!) Session key.

An attacker may have complete control over the connection flow (ie. can instantiate connection with everyone, also see past session keys, and can also trick everyone to start conversation with it.)

But cannot guess numbers chosen by others in system, and doesn't know the keys used between X and C

TRUSTED SERVER AUTHENTICATION

A and B already exchanged key with X

→ A sends to X A, B
→ X chooses k - session key - and sends back to A

$$K_{AX}(k) \qquad K_{BC}(k)$$

→ A decrypts k and computes k and sends to B

$$\left[X, A, K_{BX}(k)\right]$$

→ B decrypts k and sends back a message of hello

$$k(\text{Hello A, this is B})$$

- POSSIBLE MITM between A and C

→ A wants to connect to B, T catches connection request and changes it to (A,T)
Once C calculates session keys, trudy will catch every request and answer as B

Encrypt the session request!

A sends to C $\langle A, K_{AX}(B) \rangle$

Now attacker doesn't know receiver, but can use $K_{AC}(B)$ to REPLAY the encrypted string taken session before A, $K_{AC}(T)$
Once a sends the message he can now know the destination

USE ONE-TIME nonce

ENHANCING STRENGHT of TRUSTED SERVER AUTHENTICATION

NEEDHAM - SCHROEDER

1) A chooses N and sends X, A, B, N
2) X chooses K and sends back $K_{AX}(N, K, B, K_{BX}(K, A))$
3) A decrypts, checks N and B, sends to B: $K_{BX}(K, A)$
4) B decrypts, sends back to A $K(\text{HELLO-B}, N')$
5) A sends to B $K(\text{HELLO-A}, N'-1)$

Possible attacks based on old compromised session key!

Once A has requested to talk to B, T hashes A and talks in place of A

3) T (as A) replays to B $K_{BC}(K', A)$, where K' is an OLD compromised session key

4) B decrypts K' and sends back to T (thinking is A)

$K'(\text{Hello\_B}, N')$

5) T answers: $K'(\text{HELLO-A}, N'-1)$

Also replay attack! - if K is compromised on-line -

NEEDHAM-SCHROEDER PROTOCOL VARIANT

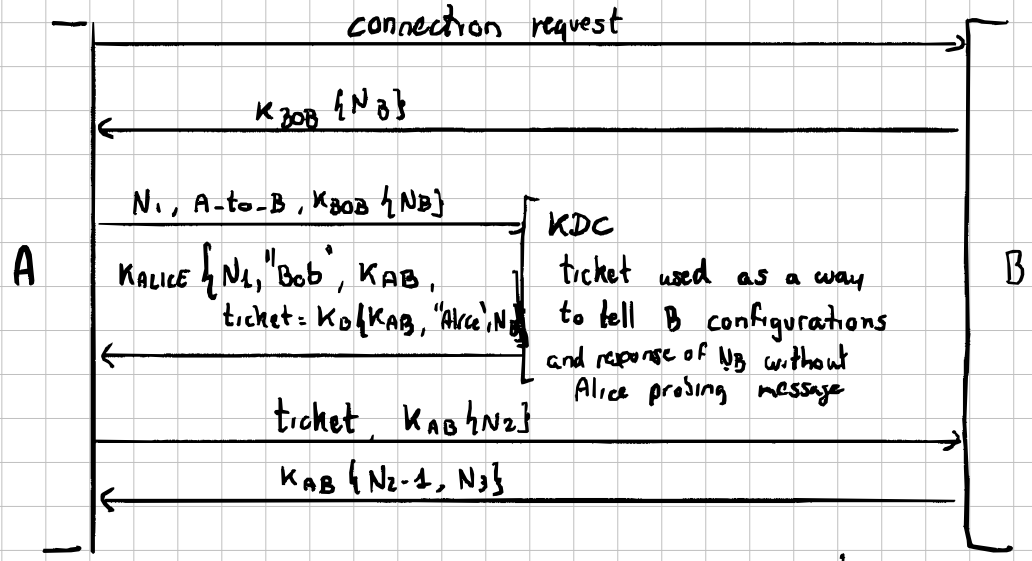
To prevent reuse of possible old keys / nonce we append timestamps to message (to invalidate old packets)

1) A sends connection request to B $\quad \langle A, N \rangle$
2) B chooses N' and sends to X $\langle B, N', K_{BX}(N, A, t) \rangle$
3) X sends to A $\langle K_{AX}(B, N, k, t), K_{BX}(A, k, t), N' \rangle$
4) A sends to B $\langle K_{BX}(A, k, t), K(N') \rangle$

It's the basis for kerberos protocol!

→ No use of old k' (since it's invalidated)
→ No replay of $K_{BC}(N, A, t)$ or $K_{BC}(A, k', t)$

We can lessen the use of KDC by connecting A, B and using the KDC to create an "authentication" ticket



Even in compromise of K_{AB} auth is not enough, since nonce is different!

→ Ticket contains information on sender!

Different from before:

before: $\{K_{AB}, \text{"Alice"}\}_{K_B}$ → (Replay)

after: $\{K_{AB}, \text{"Alice"}, N_B\}_{K_B}$ ((The KDC decrypted N_B and incorporated it)

KERBEROS

Authentication mean developed by MIT.
The scenario is using a ticket system

A needs to access services provided by B
- authentication of A
- optional: auth. of B

C is a trusted server (KDC)

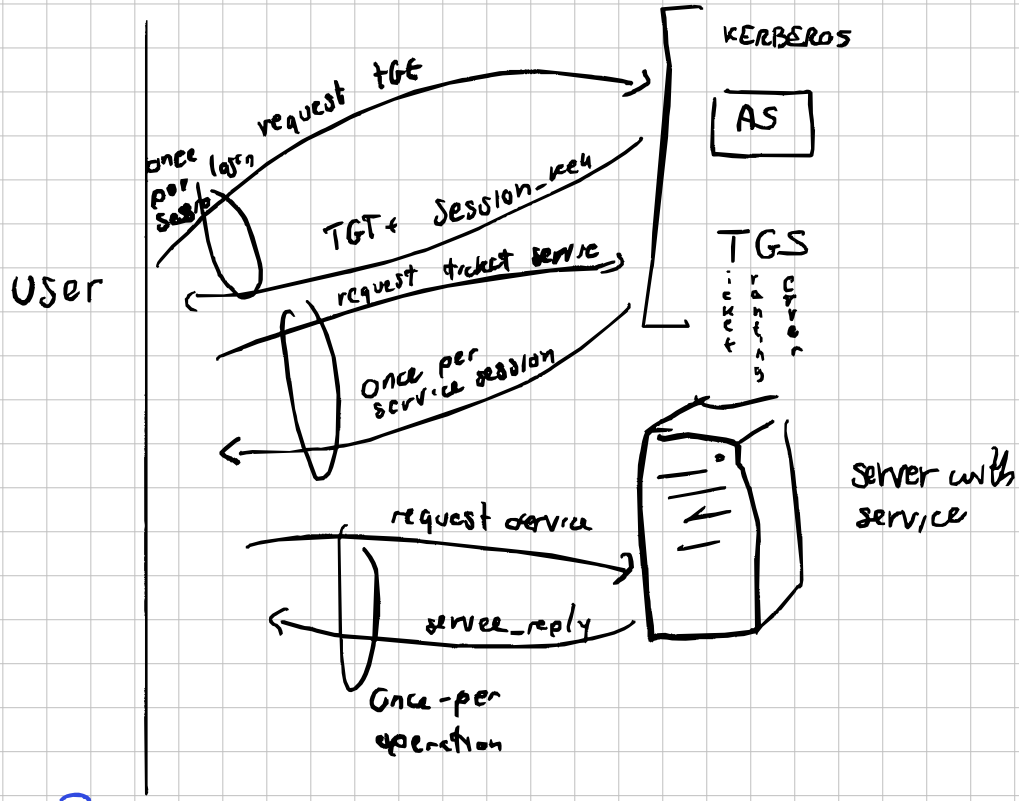
The idea is to use TICKETS to access services, valid in specific time windows

We use a KDC to provide tickets! A ticket is encrypted with secret key associated with the service. It contains information such as
- session id
- username
- service name
- lifetime
- timestamp

Messages between host and KDC should ALWAYS be protected by the master key, but it's inconvenient.

We need a TICKET-GRANTING TICKET, so that we can use this TGT to request access to other services and NOT use the password (sort of like login cookie)

2 KINDS of ticket
- session ticket
- service ticket



REALMS

- In very large systems security and performance issues suggests us to use not just a domain but more

REALMS: the answer. Each realm has a different KDC, that is in contact with other KDC realms using common key

TICKET GRANTING TICKET
TICKET GRANTING TICKET
TICKET GRANTING TICK

Password Authentication

Most popular, not great. Humans are not good with them (password short, repeatable, predictable), computers are (generates long pseudo-random KEYS, and better hidden with OTP).

UNIX Password Hash

password are not directly stored, but they are saved as HASHES (+ salt).
The user password is converted to a 56-bit secret key, and then using DES to encrypt 25 times a string of ZEROES.

-how we know that encrypting many times is basically useless-

A dictionary attack is feasable! We need to add more variables (random)

↳ To increase security we use SALTS of 12-bit random numbers.

$$DES_K(00\ldots0\langle SALT\rangle)$$

The salt is per-user generated and it can be stored in clear.

ALICE AUTHENTICATION

If Alice wants to authenticate herself she can do few things

- Send passwords in clear → Eavesdropper
- DIFFIE-HELLMAN key exchange → Impersonation
- Challenge-response handshake → Dictionary Attack
- USE STRONG PASSWORD protocols

Broad use of session key, for increased safety and MAINTAINING authentication

STRONG PASSWORD PROTOCOL

- LAMPORT'S HASH (1981)

- Bob stores $\langle username, n, h^n(password) \rangle$. where n is a LARGE number.
- Alice workstation sends $x = h^{n-1}(password)$
- Bob computes $h(x)$; if successful, n is decremented, x replaces h^n in Bob's database.

This is safe against eavesdropping! It's important to DECREMENT, otherwise it won't work since an eavesdropper could listen to one hash.

(there is no authentication of Bob, impersonating issues.)

The implementation of salt is crucial for better security, and makes sure that password round is different each time.

Attacks on Lamport's Hash

- $h^{(n-1)}(pwd|salt)$ is used for authentication
- salt is stored at bob's setup, it sends salt along n each time

Using the power n for the hash may lead to some attacks.

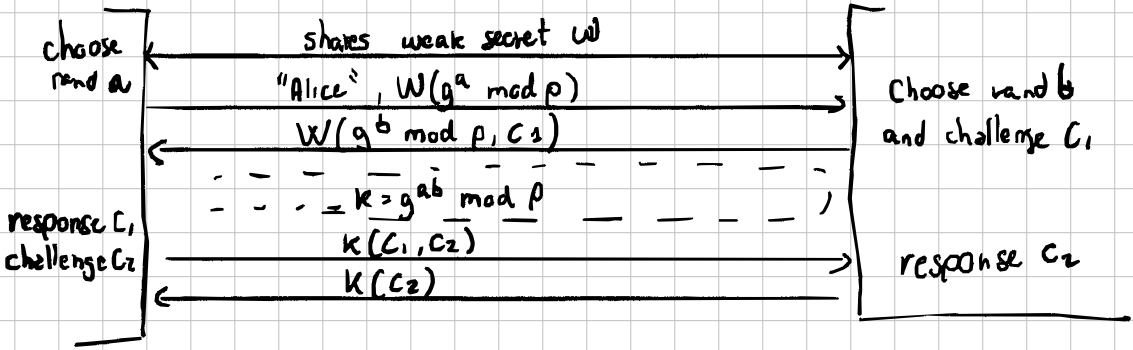
- Small n attack
  - Trudy impersonates Bob and sends $n' < n$ and salt. Once trudy gets the reply she can impersonate Alice for up to $n - n'$ times!
    We need sort of authentication of server, or Alice needs to remember approximate range of n
- "human and paper" enviroment
  - In case Alice's workstation is not trusted, she could be given a list of all hashes starting from 1000, using each hash one (in a sort of one-time password)

To mitigate attacks based on impersonation we need to have CLIENT/SERVER Authentication!

ENCRYPTED KEY EXCHANGE

EKE is diffie-helman with authentication
It's strong against password attacks, ensures mutual authentication and defines a session key through KEY EXCHANGE (of DIFFIE-HELMAN)
-based on shared secret-

pwd = Alice password ; W secret derived from pwd



EKE is strong:
- replay attack
- dictionary attack

Authentication is strong because it uses strong session key K

If the attacker knows the password then vulnerable to same MITM attack.

VARIATION OF EKE

there may be advantages on regards to modifying the EKE algorithm to make authentication easier

SPEKE

uses W in place of g in D-H exchange.
Simplifies the authentication process

PDM

password derived module; p is chosen depending on the password, and uses g = 2

Used for a long time, not anymore because of stronger protocols.

Basic EKE schemes can have augmented property: the idea is to store a quantity from the password, usable to verify the password.

USE OF AUGMENTATION

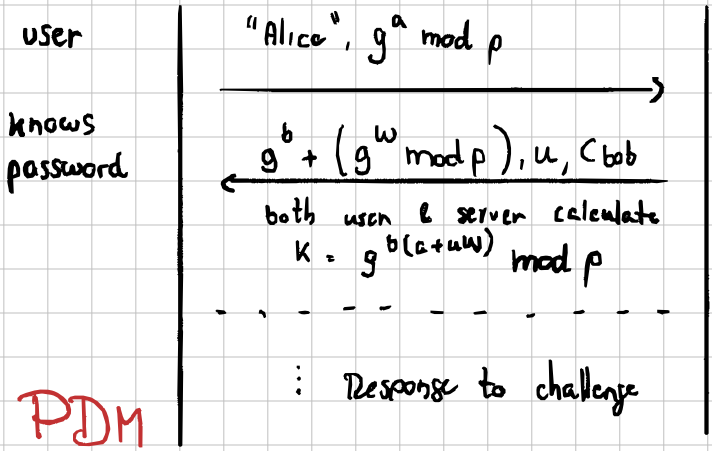
why augmentation?

- Server stores a derived value (not psw itself), even if database compromised attackers cannot directly impersonate users.
- The idea is for the server to STORE QUANTITY derived from the password, that can be used to verify the client. The quantity alone IS NOT sufficient!

The difference between, for example, EKE, is that server stores directly a password derived data. In case of servers compromise, the data BY ITSELF can be used in an impersonation attack (like normal Diffie-Hellman)

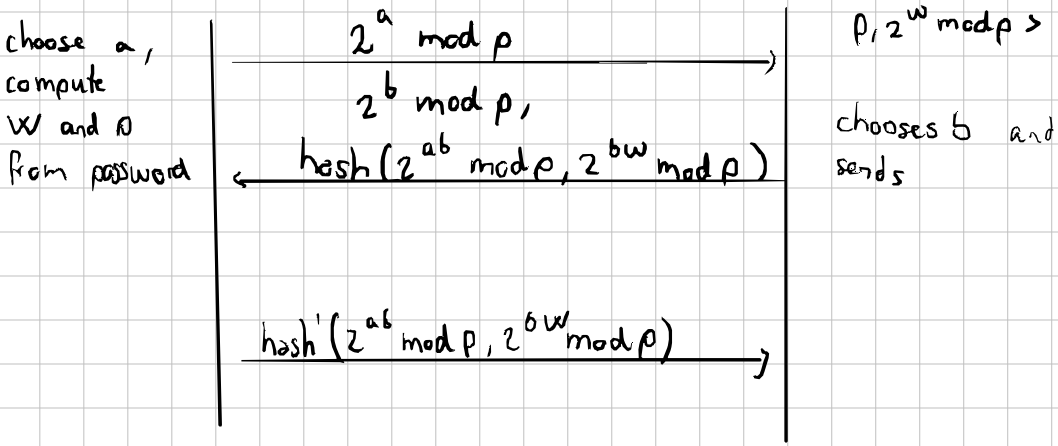
AUGMENTED EKE

server doesn't store w, which if stolen can be used to impersonate user.



PDM

Server stores p and $2^w \bmod p$, where W is a hash of password! - in case of server breach hacker can't use W, since it's "protected" by integer modulo.



Everything is based on the secrecy of the passwords! Whatever EKE and PDM are useless if passwords are compromise

IPSec

- It's a way to handle confidentiality of the 3rd stack.

  It's very complicated, was once mandatory but now it's not.

  ↳ Working IPsecV6, waiting for IPsecV6.

the idea is to encapsulate a real encrypted packet with an extra header.

Routers involved in routing will read extra header to route the packet to a IPsec enabled router, that will unencrypt and

Security features

Includes two protocol:

- **Authentication Header (AH):** for authentication of packets and data integrity
- **Encapsulated Security Payload (ESP):** for encryption of payload, confidentiality, integrity and authentication.

uses:

- Access Control
- Connectionless Integrity
- Data origin Authentication [HMAC]
- Rejection of replayed packets
- Confidentiality [with encryption]
- Limited traffic flow confidentiality.

SECURITY ASSOCIATION

A SA is a one-way relationship between sender and receiver that affords security for traffic flow. For this there will be list of security params that eases sharing.

Database of security association (SADB) will hold parameters. SA are defined by 3 main parameters:

- Security Parameters Index (SPI)
- IP Destination Address
- Security Protocol Identifier (AH, ESP)

SECURITY POLICY DATABASE (SPDB)

other DB that grants access control, different from SADB. Entries on the SPDB discriminate traffic between IPSec protected or not.

Selectors for traffic flow to one or more SA's (in the SADB) to check for packet preprocessing

SA parameters

- We have sequence number counter to avoid replay attack. Overflow should be decided beforehand, with setting flag whenever to reset counter or start new SA
- AH is a better choice for data integrity, since it's fuller and covers also headers.
- ESP needs parameters to
- Lifetime of SECURITY ASSOCIATION
- Tunnel or Transport mode
- MTU (Maximum Transmission Unit) max size of packet before needing fragmentation. This leads to fragmentation attacks.

Security Policy DataBase contains rules on how to process a packet: either IPsec protection or bypass it. Minimum entries H's different than SADBs, since latter contains SA parameters

There are many kinds of IPsec databases, but these two are the only two mandatory

Transport mode summary

Original IP header is not touched, but IPsec information is added between IP Header and packet body that is protected.

↳ No header integrity protection
it's used when between end-to-end

IP header | IPsec | payload
(payload: encrypted & authenticated)

Tunnel mode summary

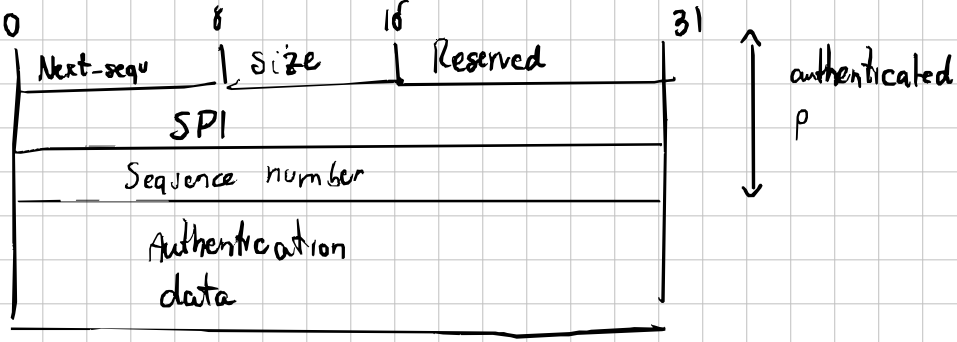
Keeps IP packet intact, but we want to protect it; add new header information outside.

New IP header | IPSec | [old IP header | packet]
([old IP header | packet]: encrypted)
(New IP header | IPSec | [old IP header | packet]: authenticated)

New header contains IPSec aware router, which will unpack envelope and send original packet.
Packet is "enveloped" in a new IP packet. Original is protected (encrypted and authenticated.)

Useful when creating a network of hosts.

AH authentication header

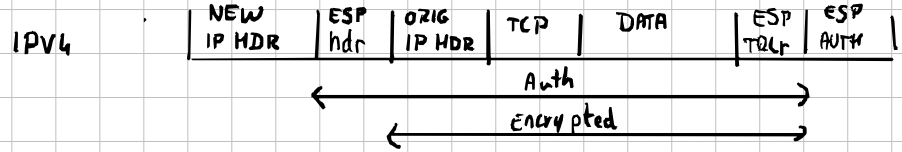


Provides support for data integrity & authentication.
Doesn't encrypt it. Uses MAC (Message Authentication Code) with shared secret key.
AH authenticates the original IP packet entirely, in tunnel mode too. Except MUTABLE fields! Like the TTL.

ESP Encapsulated Security payload

provides message content confidentiality, optionally provides same authentication as AH
Supports range of cyphers, modes, padding

- DES, AES
- SHA
- HMAC (like AH)



KEY MANAGEMENT

IPSec handles key generation and distribution.
It typically needs to pair of keys, one for AH and ESP

It uses Internet Key Exchange (IKE) which is a mix of ISAKMP + OAKLEY

ISAKMP

Internet Security Association & Key Management Protocol.
Defines procedures and packet formats to establish, modify delete SAs.
It's independent of key protocol, encrypt. alg. or authentication method.

OAKLEY

A key exchange protocol based on Diffie-Hellman key exchange, with additions.
Can use arithmetics in prime fields or elliptic curve fields

TLS

focus on TLS 1.3

We have four different TLS version

- TLS 1.0 } Broken
- TLS 1.1 }
- TLS 1.2 } Partially secure
- TLS 1.3 } secure

TLS stands for Transport Layer Security, it's a cryptographic protocol that provides security over transport layer, end-to-end. TLS is an IETF standard, with last revision being RFC 8446.

Main threats addressed

- Eavesdropping,
- Tampering,
- Authentication / Message forgery.

TLS remains connected as long as there is a connection between two parties!

TLS authentication is UNILATERAL, but can be mutual if implemented client-side authentication.

3 phases: (one TCP connection is established)

1) Peer negotiation for algorithm support } HANDSHAKE
2) Authentication and key exchange }
3) Symmetric cypher encryption and message authentication.

Check SSL-certificate : Lower strenght of certification, ↑ No owner!
↳ Let'sencrypt is one!

links domain:public key.

Can send certificate + signature request to CA, that will give back SSL cert. signed by CA.

EV certificate are SSL certificates, with information about domain owner. It's better than SSL.

END-to-END encryption

Important for confidentiality. Whatsapp offers e2e encryption. According to Internet model, we'll have 2 points exchanging a message. Data should be protected while in transit, but when messages pass through delivery servers gets decrypted and re-encrypted at every hop. e2e encryption aims to keep messages protected during the whole chain, not making even the message provider able to read. TLS is E2E encryption.

There will be a mutual key exchange between the only two peers.

It's a way for companies to not be able to send data to feds, they don't have anything.

TLS + IPSec: could be useful to have TLS and IPSEC in a secure site-to-site communications between to local nets, while using TLS to encrypt web traffic within each office.

S/MIME provides end-to-end encryption with the use of a certificate.

OpenPGP is another way. We can use tools as GPG.

TLS 1.3 is noticeably faster than counterpart 1.2, less handshake steps while keeping security.

ClientHello:
- Protocol Version:
- Random:
- Cipher suites:
- Extension:
  - HKDF
  - Supported versions.
  - SNI.
  - ALPN (App-layer protocol Number)
  - Signature algorithms.

An example ClientHello message is provided in Slide 11

ServerHello
- Protocol Version
- Random
- Cipher suite
- Extensions
  - similar to exts. in ClientHello.
  - supported version
  - key sharing (Ephemeral Diffie Hellman)

Modern use of TLS:
fucking everywhere!

SSL/TLS Heartbleed ! Vulnerability

openssl-s_client tools for checking SSL/TLS

SSH Protocol

the Secure SHell network protocol allows data to be shared through a secure channel. It allows for

- one (or two) way authentication
- data confidentiality
- data integrity

It allows to:

- spawn a shell
- use it as a port forwarder (-L)
- copy files (scp)
- forward X from remote host
- securely mount a directory on remote server to local filesystem, thanks to VFS (trough SSHFS)

SSH Communication Security (SCS)

SSH is a trademark of SCS,
founded by Finnish researcher.
We have OpenSSH which is FOSS

versions:

- SSH 1 : Original version, was freware then became paid, full of vulnerabilities (1995)
- SSH 2 : Revised version from (2006)
- OPENSSH : open source project from free fork of last free SSH retrocompatible with SSH1 & SSH2

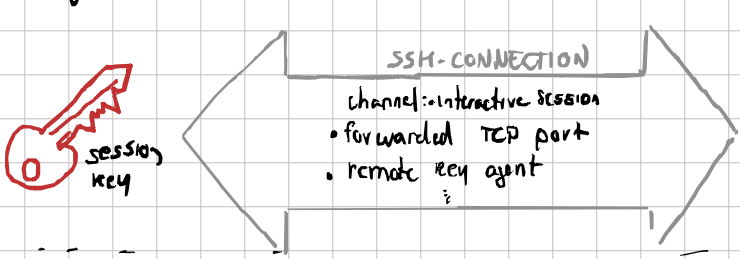
Basic SSH Architecture

CLIENT

list of known_hosts:
{public pair of host: pubkeys
connected before servers}

SERVER

host_key
<PUB : PRIV>



user account

User key
<PUB : PRIV>

Target account

user PUB key

The SSH architecture is mainly divided into 4 pieces, each one formally defined in different IETF standards:

- **SSH-TRANS**: provides integrity, encryption and server authentication protocols. Manages key re-exchange after 1GB or
- **SSH-AUTH**: can use both public-key or DSS to auth client, but can be augmented to other standards. After auth clients invoke SSH-CONN
- **SSH-CONN**: manages multiple logical channels over single SSH connection
- **SSH-SFTP**: provides file transfer capabilities,

SSH protocol workflow

1. **Initialization**: client and server negotiate protocol version and agree on algorithms. SSH-trans deals with this part.
2. **Key exchange**: secure (and authenticated) key generation using Diffie-Hellman or other key exchange mechanisms.
   SSH trans deals with this too, and exposes an interface to upper layer for sending/receiving plaintext data.
3. **Server Authentication**: Server sends its public key for the client to verify along a list of known_hosts.
   SSH-TRANS deals with this. (known_hosts hostname: pubkey)
4. **Client Authentication**. Client authenticates itself using supported methods provided by the server. It's CLIENT-DRIVEN and supported by SSH-AUTH protocol:
   - password-based: password auth, not everyone uses it
   - publickey: supports at least DSA or RSA, also x509. Client sends its public pair and server checks it
   - keyboard-interactive: server sends one or more prompts to enter informations, clients sends back user provided info (password + otp)
   - GSSAPI: extensible scheme to perform SSH auth over external platforms (kerberos/NTLM), providing SSO capabilities to SSH versions.

SSH2 protocol algorithms

- key exchange established through diffie-hellman
- Server authentication via DSS or RSA signatures
- HMAC or MAC algorithms
- 3DES, RC4, or AES
- Pseudo-random functions for key-derivation

SSH Security - UPDATE

WikiLeaks confirms that CIA had software specifically created for SSH interception of connection started from compromised clients

BothamSpy and Grifalcon

usage

ssh [options] <userid>@<hostname> [command]

some of the options include

- -v for different verbose options
- -C enables GZIP compression
- -N doesn't start interactive shell
- -L <outport>:<address>:<inport> forwards the port <inport> of host <address> to localhost at port <outport>
- -1, 2 forces version
- -X forwards X11 session to client

with no option an interactive shell session will start.

IPTABLES

Somewhat old firewall, but D'Amore's wants it.
we define our action as TARGET, in which we can have DROP or PERMIT

We have two attributes for INCOMING & OUTGOING traffic with HOST & PORT.
We may have comment for every entry.

we may have overlapping, but it's not a problem since the pattern matching is sequential. This is easier, as creating partitioned rules that can be difficult to implement.
The general rule should be the last rule!

As an example, we may want to block all access except for web server

we want to allow all incoming traffic to 80 port.
and we want allow outgoing ports >1024

(stateless) PACKET FILTER - DISADVANTAGES

- Cannot prevent attacks based on application levels.
- There is no user authentication, so packets can be spoofed
- There may be security breaches due to firewall misconfiguration.

FRAGMENTATION ATTACK

Fragmented datagrams can be problematic for packet filtering. When fragmenting, it may be not easy to reconstruct correctly the whole packets.

> two packets that pass firewall may be reassembled to packet that wouldn't have passed otherwise.

Stateless filtering has limitations regarding from lack of state itself!

SESSION FILTERING

Decision is made IN CONTEXT of a connection!
- if new connection, check packet filtering
- if existing connection, look in table and update it.

this kind of filtering heavily relies on a CONNECTION, so it wouldn't be possible to correctly filter ICMP or UDP transport packet.

[on linux we can see connection tables with command lsof]

IPtables does session filtering!

IPTABLES

chain: list of rules which match set of packets. The rules are ORDERED, and have a PATTERN and a TARGET
there are multiple chains.

There are many tables:
- filter (default)
- NAT
- mangle
- RAW

when we want to work with firewalls we'll work with filter tables.

we have BUILT-IN or USER-DEFINED chains.
we'll create

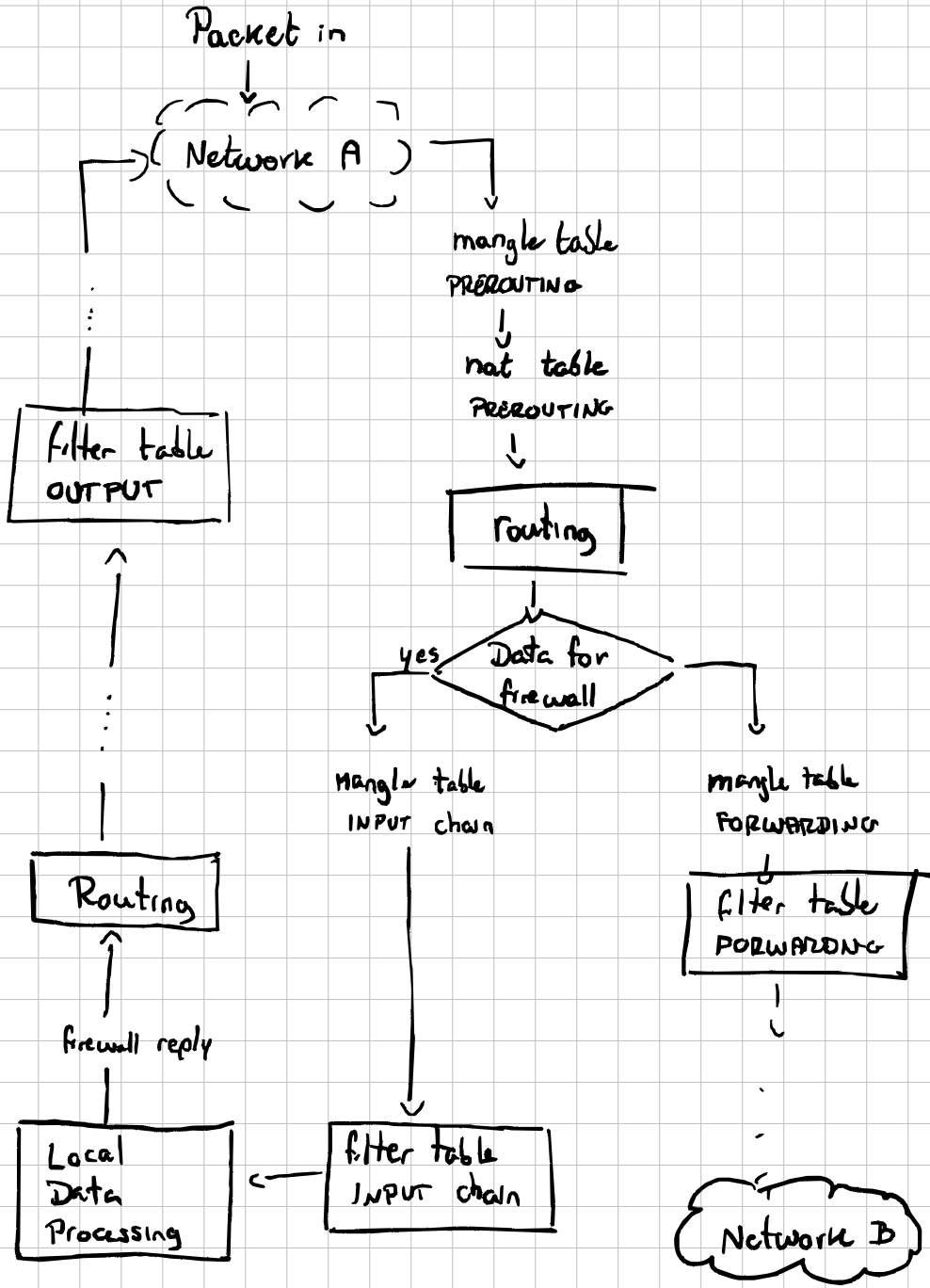
BUILT-IN CHAINS

- PREROUTING: packets handled before routing procedure (used with NAT)
- INPUT: packets destined to local sockets
- FORWARD: packets incoming to firewall, destination OUT of firewall [perimeter level firewall]
- OUTPUT: locally-generated packets
- POSTROUTING: for packets handled after routing (used with NAT)

built-in chan have a policy, default DROP, which is applied to packets that don't match any rule in chain

TARGET: we have ACCEPT or DROP, which are self explanatory. Others are QUEUE, RETURN or a name of a user-defined chain (subroutine calling)

[user-defined chains don't have a policy, IP no matching we'll RETURN back to previous chain. RETURN doesn't really make sense on built-in chains.


Packet in
Network A
mangle table
PREROUTING
nat table
PREROUTING
Routing
yes
Data for firewall
filter table
OUTPUT
Routing
mangle table
INPUT chain
mangle table
FORWARDING
filter table
FORWARDING
firewall reply
Local Data Processing
filter table
INPUT chain
Network B

Iptables supports EXTENDED MODULES, which are enrichment of the filtering techniques by adding extra rules.

Loading modules has syntax -m <MODULE_NAME> along with setting module options by using "--"

example "iptables -m state [!] --state $STATE"
- useful for session filtering.
where state can be: INVALID, ESTABLISHED, NEW, RELATED

modules can be loaded implicitly by using -p directly. (useful in case of TCP/UDP addressing)

EXTENDED MODULES

mac : filtering based on MAC address
ttl
tcp
udp
:

iptables options

COMMANDS
- -A append chain rule-specification
- -L lists chain
- -D deletes rule
- -F flushes all rules!

[man iptables for more]

PARAMETERS
- -p protocol
- -s source address/mask
- -i in-interface name

Some examples in slides no. 31

SECRET SHARING - SHAMIR

We want to share secret S with N different people, without anyone knowing the whole S. The secret can be reconstructed by FRAGMENT they hold.

there are $P = \{p_1, p_2, \dots, p_N\}$ fragments

we'll define a number $n < N$ such that secret can be reconstructed by having subset $S \subseteq P$ s.t $|S| = n$, whatever the fragment is.

Adversary can't access secret, even with INFINITE computational power.

For example, we may consider XOR-ing all fragments to reconstruct original secret S.

(k, n)
n: fragments
k: needed fragments to reconstruct secret

POLYNOMIAL INTERPOLATION
MODULAR ARITHMETIC & FINITE FIELDS

Th: given $r \in \mathbb{N}^+ : r > 1$ points of $\mathbb{R}^2$ there exists a unique polynomial of degree $r-1$ going exactly through r points

- Let p be a prime $(p > S, p > n)$, randomly choose $k-1$ integers in $[0, p)$: $a_1, a_2, \dots, a_{k-1}$, with $a_0 = S$
- Consider polynomial $P(x) = a_{k-1} X^{k-1} + a_{k-2} X^{k-2} + \dots + a_1 X + a_0$
- Let $S_i = p_i$ for $i = 1, \dots n$
  by construction $P(0) = S$

Polynomial is the one that will be used to generate n secrets!

Poly is k-degree and we'll produce n points

Lagrange formula to set poly $P(x)$ given k points