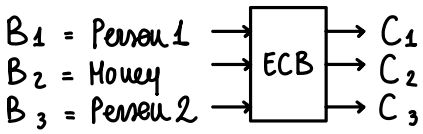
ATTACK TO ECB: BLOCK REPLAY

Let's consider a money transfer system in which banks agree on a standard message format to perform money transfer.
Suppose that the structure is Person1, Money, Person2 and that this message is splitted in 3 blocks encrypted with ECB such that:



$\rightarrow C_1 \| C_2 \| C_3$ to the bank

Let's now suppose that another transaction is performed so:

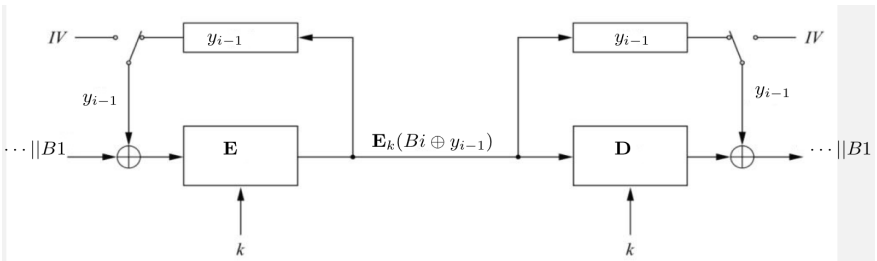
$\tilde{P}_1, \tilde{m}, \tilde{P}_2 \rightarrow \tilde{B}_1 \| \tilde{B}_2 \| \tilde{B}_3 \rightarrow \tilde{C}_1 \| \tilde{C}_2 \| \tilde{C}_3$

If an attacker performs traffic analysis and get $C_1 \| C_2 \| C_3$ and $\tilde{C}_1 \| \tilde{C}_2 \| \tilde{C}_3$ he can perform a replay attack reusing the valid encrypted blocks. For example sending $C_1 \| \tilde{C}_2 \| C_3$. This message will be considered valid!
A way to prevent this is adding to the message an identifier or a timestamp.

! The attacker just replay the ciphertext, he doesn't know anything about the key or plaintext !

CBC, Cipher Block Chaining:

The message P is divided in blocks, then each block is XORed with the ciphertext of the previous block before being encrypted. The first block is XORed with an IV that is generally a NONCE and if it is random makes the cipher NON DETERMINISTIC / PROBABILISTIC.



ENCRYPTION:

$y_1 = \text{Enc}_k(B1 \oplus IV)$
If $j > 1$ then $y_j = \text{Enc}_k(Bj \oplus y_{j-1})$.

DECRYPTION:

$B1 = \text{Dec}_k(y_1) \oplus IV$
If $j > 1$ then $B_j = \text{Dec}_k(y_j) \oplus y_{j-1}$.

Initialization vector must be sent along with C to permit decryption at the receiver

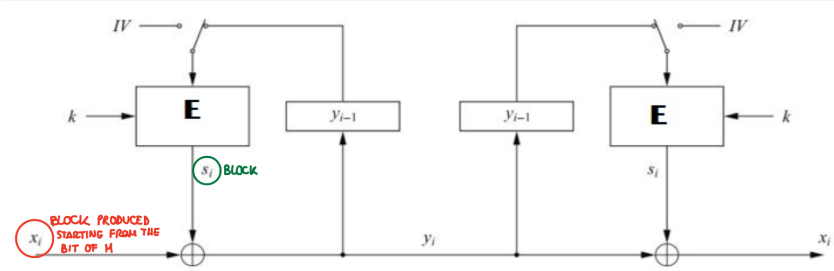
- Advantages: Allows probabilistic encryption. Blocks depend upon each other so filtering/modification attacks are detected. Deciphering can be done in parallel.

About IV:

The CBC, CFB, and OFB modes require an initialization vector as input, in addition to the plaintext. An IV must be generated for each execution of the encryption operation, and the same IV is necessary for the corresponding execution of the decryption operation. Therefore, the IV, or information that is sufficient to calculate the IV, must be available to each party to the communication.
The IV need not be secret, so the IV, or information sufficient to determine the IV, may be transmitted with the ciphertext.
For the CBC and CFB modes, the IVs must be unpredictable. In particular, for any given plaintext, it must not be possible to predict the IV that will be associated to the plaintext in advance of the generation of the IV.

CFB, Cipher FeedBack mode (PRNG):

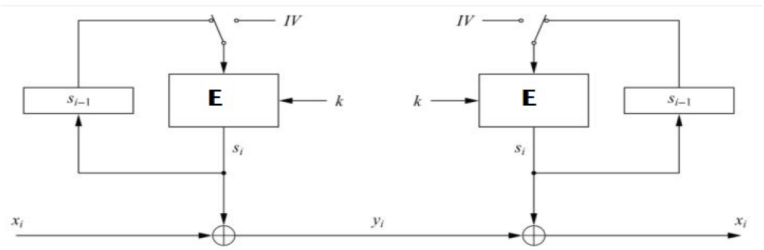
In this mode the block cipher is used to construct a stream cipher.



Here we have just a XOR between the block of the message x_i and the encryption of the previous block (STREAM CIPHER).
In fact the block cipher is used just to generate the keystream to XOR with the message's pieces x_i.

OFB, Output FeedBack mode (PRNG):

Also here the block cipher is used to construct a stream cipher.



$S_0 = IV$
$S_1 = f(S_0)$
...
$S_M = f(S_{M-1})$
where $f(x) = \text{Enc}_K(x)$ and
$C = x_n \oplus S_M \ldots x_1 \oplus S_1$

LIKE LEHMER GENERATOR WHERE
$S_i = S_{i-1} \cdot a + b \pmod N$

- **CTR, Counter Mode:** the plaintext P is divided into N blocks $P = B_N \| \dots \| B_1$, then other N blocks called COUNTERS $T_N, \dots, T_1$ are created to be encrypted producing O_j that will be xored with B_i (works like a stream cipher).

CIPHERING
$$O_j = Enc_K(T_j) \quad j = 1 \dots N$$
$$C_j = B_j \oplus O_j \quad j = 1 \dots N$$

DECIPHERING
$$O_j = Enc_K(T_j) \quad j = 1 \dots N$$
$$B_j = C_j \oplus O_j \quad j = 1 \dots N$$

Usually the counters $T_N, \dots, T_1$ are obtained from the initial block T1 by increment or partial increment. For example $T1 = IV \| Q$, where $Q \in \mathbb{Z}_{2^m}$ and $T_j = IV \| (Q + j - 1)$ and IV is a nonce. Counters can be created by a LFSR from an initial state T1.

> It is important to prevent two counters from being equal. Otherwise you risk a KPA attack: if $T1 = Tj$ and $(Ct_1, B1)$ known to the cryptanalyst then he also easily gets Bj.
>
> To avoid two counters being the same, you should choose the size m of Q to be able to encrypt all N blocks of our message before changing the nonce IV, ie $N \leq 2^m$.
>
> It is not necessary for the first counter $T1$ to be secret. The sender could send $T1$ together with the first encrypted block Ct_1 to the recipient.

Advantage: CTR IS PARALLELIZABLE

- **GCM, Galois Counter Mode:** GCM is a combination of CTR mode by using an IV, and a MAC. A tag t of 128 bits is produced to allow the receiver to check the authenticity of the message. The used block cipher must use 128 bit blocks. It is called Galois because blocks are regarded as number of the finite galois field $GF(2^{128})$ using a $G(x) = x^{128} + x^7 + x^2 + x + 1$.

> The IVs in GCM must fulfill the following "uniqueness" requirement:
>
> **The probability that the authenticated encryption function ever will be invoked with the same IV and the same key on two (or more) distinct sets of input data shall be no greater than 2^{-32}.**
>
> Compliance with this requirement is crucial to the security of GCM. Across all instances of the authenticated encryption function with a given key, if even one IV is ever repeated, then the implementation may be vulnerable to the forgery attacks that are described in Ref [5] and summarized in Appendix A. In practice, this requirement is almost as important as the secrecy of the key.

The message P is divided in blocks $B_N \| \dots \| B_1$ of 128 bits. As in CTR a set of counters T_j are produced and encrypted obtaining O_j. Then $C_j = B_j \oplus O_j$

Moreover a tag is produced.

How is produced the tag (MAC):

$H = Enc_k(0^{128})$;
$g_0 = AAD \otimes H$; where $\otimes$ is the Galois multiplication in $GF(2^{128})$.
$g_j = (g_{j-1} \oplus C_j) \otimes H$ for $j = 1, \cdots, N$;
$L = [len(AAD)]_{64} || [len(P)]_{64}$;
$t = ((g_N \oplus L) \otimes H) \oplus Enc_k(T0)$.

the receiver gets:

$(C_N, \dots, C_1, t, AAD)$

Where AAD are Additional AuthN Data.

NOTE 5.1.15

From the mathematical point of view the tag **t** is basically the evaluation of H in a polynomial whose coefficients are the blocks of the ciphertext. Namely,

$$\mathbf{t} = C_1 H^{N+1} + C_2 H^N + \cdots + C_{N-1} H^3 + C_N H^2 + g_0 H^N + LH + Enc_k(T0)$$

($T0 = Enc_K(00\dots00)$, 128 bits)

Poly whose coeff. are from the ciphertext

The receiver decrypt as in CRT mode and by using the AAD a tag $\mathbf{t}'$ is computed via the above algorithm 5.1.14. If $\mathbf{t} = \mathbf{t}'$ then the receiver is assured that the ciphertext (and AAD) were not manipulated in transit and that only the sender could have generated the message.

In practice, the string AAD might include IP addresses and parameters in a network protocol. So the tag is used for the authentication of the AAD.

- **RESOLVING THE EQUATION $aX = 1 \pmod n$:**

If it is possible to find such a $\forall a \neq 0$ then $\mathbb{Z}_n$ is a FIELD and this happens $\iff$ n is a PRIME NUMBER

When n is not prime, I get a SET OF INVERTIBLE REMAINDERS called UNITS. All the remainders in $\mathbb{Z}_n$ form a MODULAR RING and if n is prime, the modular ring is a FIELD.

The number of elements of the set of units are $\#\mathbb{Z}_n^* = \phi(n)$ where ϕ is the EULER'S FUNCTION. If $n = p$ is prime $\#\mathbb{Z}_p^* = \phi(p) = p-1$. The EULER function is multiplicative for co-prime numbers:

e.g.

$$\phi(45) = \phi(5\cdot 9) = \phi(5)\cdot\phi(9) = 4\cdot\phi(9) = 4\cdot 6 = 24$$

$\phi(5)$: prime $\Rightarrow 5-1$ — $\phi(9)$: would be $\phi(3\cdot 3)$ but they're not CO-PRIME! — $\mathbb{Z}_9 = \{0,1,2,3,4,5,6,7,8\}$ MUST PICK THE NUMBERS NOT CO-PRIME WITH 9

$\Rightarrow$ the number of invertible coefficients in $\mathbb{Z}_{45}$ are 24

- **EUCLIDIAN EXTENDED ALGORITHM:** defines an empiric method to find the inverse of a in $\mathbb{Z}_b$ in the eqz $a\cdot x = 1 \pmod b$ considering $r = b - a\cdot q$ the remainder in the division of a by b.

GENERAL RULE

$$(a,b) \xrightarrow{q} (r,a)$$
$$(\tilde y - q\tilde x, \tilde x) \xleftarrow{q} (\tilde x, \tilde y)$$

e.g. $29\cdot x = 1 \pmod{45}$

YOU WILL GET THE PAIR (0,1) ONLY IF a IS INVERTIBLE

$$(29,45) \xrightarrow{1} (16,29) \xrightarrow{1} (13,16) \xrightarrow{1} (3,13) \xrightarrow{4} (1,3) \xrightarrow{3} (0,1) \parallel \text{STOP}$$

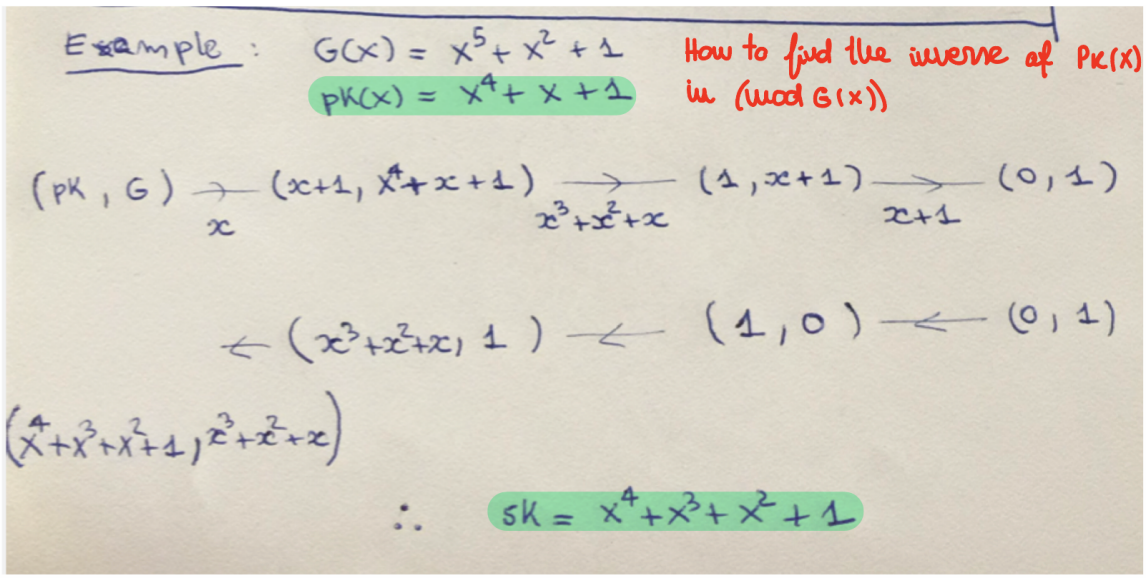
SAME q BACKWARDS

$$(14,-9) \xleftarrow{1} (-9,5) \xleftarrow{1} (5,-4) \xleftarrow{1} (-4,1) \xleftarrow{4} (1,0) \xleftarrow{3} (0,1)$$

$(1,0)$: $\tilde y - q\tilde x, \tilde x$ — $(0,1)$: $\tilde x, \tilde y$

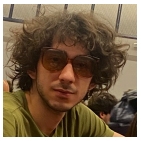
the inverse of $a = 29$ is 14 in $\mathbb{Z}_{45}$, let's check it: $29\cdot 14 = 406 = 1 \pmod{45}$ ✓

This method can be used to find the inverse of a poly in the Galois field:



- This rule comes from an Euclide observation: $d = g.c.d.(a,n) = g.c.d.(a,r)$ with $n = a\cdot q + r$ annuntiation that reduces a lot the complexity passing from a number a to the remainder, smaller number.

DIM CHE AL MOMENTO NON HO CAPITO



Now since $g.c.d.(a,n) = g.c.d.(a,r) = d$ we can say that:

$$d = r\tilde x + a\tilde y = (n - aq)\tilde x + a\tilde y = a(\tilde y - q\tilde x) + n\tilde x = ax + ny$$ ✓

$(\tilde y - q\tilde x)$ is the EEA $(\tilde x, \tilde y) \to (\tilde y - q\tilde x, \tilde y)$

- **Faster way to find the inverse by diScala:** general formula:

$$\begin{matrix} A & C \\ B & D \end{matrix} \xrightarrow{A>B} A = Bq + r \rightarrow \begin{matrix} r & C - q \cdot D \\ B & D \end{matrix}$$

$$\searrow^{B>A} \quad B = Aq + r \rightarrow \begin{matrix} A & C \\ r & D - q \end{matrix}$$

example $29 \cdot x = 1 \pmod{45}$

$$\begin{matrix} 29 & 1 \\ 45 & 0 \end{matrix} \rightarrow \begin{matrix} 29 & 1 \\ 16 & -1 \end{matrix} \rightarrow \begin{matrix} 13 & 2 \\ 16 & -1 \end{matrix} \rightarrow \begin{matrix} 13 & 2 \\ 3 & -3 \end{matrix} \rightarrow \begin{matrix} 1 & 14 \\ 3 & -3 \end{matrix} \rightarrow \begin{matrix} 1 & 14 \\ 0 & -45 \end{matrix}$$

$45 = 29 \cdot 1 + 16$ $\quad 29 = 16 \cdot 1 + 13$

- **Complexity:** $(a, n) \xrightarrow{q_i} (x_i, y_j) \xrightarrow{q_n} (0, 1)$ how many steps are needed based on a and n?

The WORST CASE is the one for which all the $q = 1$. $\quad \begin{cases} x_i = y_{i+1} - x_{i+1} \cdot (1) \\ y_i = x_{i+1} \end{cases} \Rightarrow x_{i+2} = x_i + x_{i+1}$

that follows the FIBONACCI SEQUENCE with a $\log(n)$ COMPLEXITY

- **The Chinese Remainder Theorem (CRT):**

The CRT allows us to compute $\mathbb{Z}_{n \cdot m}$ regarding its elements as PAIRS. It defines an ISOMORPHISM between $\mathbb{Z}_{n \cdot m}$ and $\mathbb{Z}_n \times \mathbb{Z}_m$.

(ISOMORPHISM: is a 1 to 1 correspondence between the elements of two sets)

For example: $\mathbb{Z}_{24} \cong \mathbb{Z}_3 \times \mathbb{Z}_8$:

0	1	2	3	4	5	6	7	8	9	10	...	14	18	19	20	21	22	23
(0,0)	(1,1)	(2,2)	(0,3)	(1,4)	(2,5)	(0,6)	(1,7)	(2,0)	(0,1)	(1,2)	...	(2,1)	(0,2)	(1,3)	(2,4)	(0,5)	(1,6)	(2,7)

to build the table the pairs in $\mathbb{Z}_3 \times \mathbb{Z}_8$ increment by 1 in both position in the modular rings $\mathbb{Z}_3$ and $\mathbb{Z}_8$.
So generally given $a \in \mathbb{Z}_{24} \longrightarrow (a \pmod 3, a \pmod 8) \in \mathbb{Z}_3 \times \mathbb{Z}_8$ eg. $17 = 2 \bmod 3 = 1 \bmod 8 \Rightarrow 17 \rightarrow (2,1)$

This is useful to simplify some operations like the inverse computation:

e.g.

$17^{-1} = ? \longrightarrow f(17) = (2,1)$ and now I find the inverse of 2 in $\mathbb{Z}_3$ and the inverse of 1 in $\mathbb{Z}_8$

- $2^{-1} \pmod 3 = 2$
- 1 is the inverse of himself

$\Rightarrow (2,1)^{-1} = (2,1) \Rightarrow 17^{-1} = 17 \pmod{24}$ ✓

- Let's now consider $F(X, Y)$, if we consider the canonical base $(1,0), (0,1)$ then

$$F(x,y) = F(X(1,0) + Y(0,1)) \underset{\text{LINEARITY}}{=} F(X(1,0)) + F(Y(0,1)) = X\underbrace{F(1,0)}_{A} + Y\underbrace{F(0,1)}_{B} = \dots$$

compute A and B:

- A is a value such that is a multiple of 8 and gives 1 mod 3: $A = 8 \cdot K \underset{\downarrow \bmod 3}{=} 1 \pmod 3$
- $B = 3K = 1 \pmod 8 \longrightarrow K = 3$ WORKS $\Rightarrow B = 9$ $\qquad 2K = 1 \Rightarrow A = 16$ WORKS

$\dots = 16 \cdot X + 9 \cdot Y$

- Let's verify it: $f(2,5) = 16 \cdot 2 + 9 \cdot 5 = 32 + 45 = 77 \longrightarrow (77 \bmod 3, 77 \bmod 8) = (2,5)$ ✓

BUT $77 \notin \mathbb{Z}_{24}$! Then we can see that if I add multiples of $24 = 3 \cdot 8$ I will have the same result.

$\longrightarrow 77 - 3 \cdot 24 = 5 \in \mathbb{Z}_{24} \longrightarrow (5 \bmod 3 = 2, 5 \bmod 8 = 5)$ ✓

- This isomorphism can be done with 3 factors. First choose three modular ring and three coprime numbers for each ring:

$\mathbb{Z}_3$	$\mathbb{Z}_5$	$\mathbb{Z}_7$
1	4	2

now we want to find $x \in \mathbb{Z}_{105}$ such that $\begin{cases} x = 1 \pmod 3 \\ x = 4 \pmod 5 \\ x = 2 \pmod 7 \end{cases}$

We want a map: $\mathbb{Z}_3 \times \mathbb{Z}_5 \times \mathbb{Z}_7 \xrightarrow{f} \mathbb{Z}_{105}$, $(r, s, t) \rightarrow (x)$ $\Rightarrow f(r,s,t) = r\underbrace{f(1,0,0)}_{A} + s\underbrace{f(0,1,0)}_{B} + t\underbrace{f(0,0,1)}_{C}$

$A = \overbrace{7\cdot 5}^{35}\cdot K = 1 \pmod 3$ ↓ mod 3 → $2K = 1 \bmod 3 \Rightarrow A = 70$ WORKS

$B = \overbrace{3\cdot 7}^{21} K = 1 \bmod (5)$ ↓ mods → $1K = 1 \bmod (5) \to B = 21$

$C = 15K = 1 \pmod 7$ ↓ → $K = 1 \Rightarrow C = 15$

$\mathbb{Z}_3 \times \mathbb{Z}_5 \times \mathbb{Z}_7 \to \mathbb{Z}_{105}$

The map is $f(r,s,t) = 70r + 21s + 15t$ $\Rightarrow$ $f(1,4,2) = 70 + 21\cdot 4 + 15\cdot 2 = 184 = 79 \pmod{105}$

$79 = (79\,\%\,3, 79\,\%\,5, 79\,\%\,7) = (1,4,2)$ $\mathbb{Z}_{105} \to \mathbb{Z}_3 \times \mathbb{Z}_5 \times \mathbb{Z}_7$

• Relationship between Euler function Φ and CRT:

Given $(x,y) \in \mathbb{Z}_3 \times \mathbb{Z}_8$, then the correspondent value in $\mathbb{Z}_{24}$ is INVERTIBLE $\iff$ x is invertible in $\mathbb{Z}_3$ and y in $\mathbb{Z}_8$.

From this we can say that $\Phi(p\cdot q) = \Phi(p)\cdot\Phi(q)$ where $\mathbb{Z}_{p\cdot q} \leftrightarrow \mathbb{Z}_p \times \mathbb{Z}_q$ is an ISOMORPHISM.
Then Φ is multiplicative if p and q are COPRIME.

So if $n = p_1^{r_1}\cdot p_2^{r_2}\cdot \ldots \cdot p_k^{r_k}$ $\Rightarrow$ $\Phi(n) = \Phi(p_1^{r_1})\cdot \ldots \cdot \Phi(p_k^{r_k})$

But what if we have a prime number?

$\Phi(p^s) = ?$ where p is prime and $s \in \mathbb{N}$, $x \in \mathbb{Z}_{p^s}$ is invertible mod N $\iff$ x is coprime with N

For example $\Phi(7^5)$, we must find the number of coprimes in

All that column contains 7^4 elements so:

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

General $\Phi(p^s) = p^s - p^{s-1} = p^{s-1}(p-1)$

1	2	3	4	5	6	7
8	9	..	..	..	..	$2\cdot 7$
.						$3\cdot 7$
.						.
.						.
.						.
-	-	-	-	-	-	$7^4\cdot 7$

All this column is not coprime with 7 because they have at least one factor equal to 7

The use of this is related to the computation of the order of an element:

$g \in \mathbb{Z}_n$, g is invertibe. If I start computing $g, g^2, g^3, \ldots g^r = 1$ at some point I will find in this sequence $g^r = 1$ and r is called PERIOD or ORDER of g.

• One way functions and trapdoors:

A function $f: D \to T$ is one-way if for $x \in D$ the value $f(x)$ can be computed efficiently but for any $y \in T$ it is not computationally feasible to find $x \in D$ such that $f(x) = y$.

One-way functions f can be obtained by using a secure block-cipher $(\mathsf{Enc}, \mathsf{Dec})$.

Example 1) :

$$f : \mathcal{K} \to \mathcal{C}$$

from the key space to the ciphertext space is defined as follows: pick a random plaintext P_0 and put

$$f(k) := \mathsf{Enc}_k(P_0)$$

So, since encryption should be efficient we get that computing $f(k)$ is easy. But finding k for a given C is cryptanalysis which should be hard.

Example 2) :

$$f : \mathcal{P} \to \mathcal{C}$$

from plaintext space to ciphertext space defined as

$$f(P) := \mathsf{Enc}_k(P)$$

here the key k is regarded as a **trapdoor** i.e. the special information which turns to easy inversion. TRAPDOOR = special information that permits the inversion in an easy way

- **Discrete Logarithm:** it's a typical example of one way function. It's the power map $f: \mathbb{Z}_n \to \langle \alpha \rangle$ for which $\gamma \to \alpha^\gamma$ where α has order n.

Given this function the problem is to find x from β where $f(x) = \alpha^{x} = \beta$ (x: SECRET KEY; α: GENERATOR; β: PUBLIC KEY)

This is the one-way func. used by Diffie-H

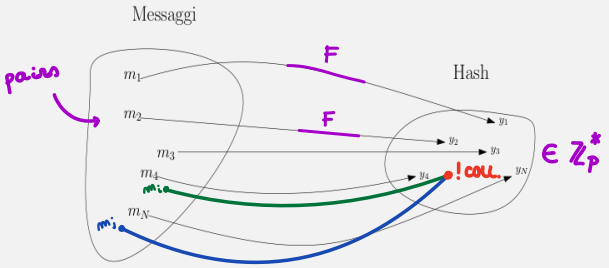
The best way to solve this problem is looking for collisions. Let's introduce $F(s,t) = \alpha^s \beta^t$. A collision happens when two arguments goes in the same result (if this happens the function is not INJECTIVE).
So the first step is to find a collision such that:

$$\alpha^a \beta^b = \alpha^A \beta^B \xrightarrow{\beta = \alpha^x} \alpha^a \alpha^{xb} = \alpha^A \alpha^{xB} \to \alpha^{a+xb} = \alpha^{A+xB} \to a+bx = A+Bx \pmod{r} \to (B-b)x = a-A \pmod{r}$$

(r: ORDER OF α)

We just need to solve the modular equation (SIMPLE!) to find the key. The real problem is find a collision...
To do so we use the **birthday paradox attack**:

Assume a digest of n bit. With N messages $x_1, \cdots x_N$ we can form $\frac{N(N-1)}{2}$ pairs which are candidates for a collision. So it is natural to guess that the probability λ_N of a collision between two of these $\frac{N(N-1)}{2}$ pairs of messages would be related to the numbers $2^{\frac{n}{2}}$ and 2^n.



Here the equation relating λ_N and the digest length:

$O(\sqrt{n})$ ← of the order of $\sqrt{p}$

$$N \approx 2^{\frac{n+1}{2}} \cdot \sqrt{\ln\left(\frac{1}{1-\lambda_N}\right)}$$

n = # bits of p

$n = \log_2(p)$

The above equation follows from the following discussion.
A collision between the N messages is produced when the restriction of the **Hash** function to the set of N messages is no more injective. So the probability $1 - \lambda_N$ of **no collision** is the quotient between the number of injective functions and the number of all possible functions:

- **Hash functions:** Hash functions-such as MD5, SHA-1, SHA-256, SHA-3, and BLAKE2 -comprise the cryptographer's Swiss Army Knife: they are used in digital signatures, public-key encryption, integrity verification, message authentication, password protection, key agreement protocols, and many other cryptographic protocols.

An hash function has in input a sequence of values in $\mathbb{Z}_2^*$ and output a fixed length sequence of $n = 160, 256, 512$ bits. HASH: $\mathbb{Z}_2^* \to \mathbb{Z}_2^n$

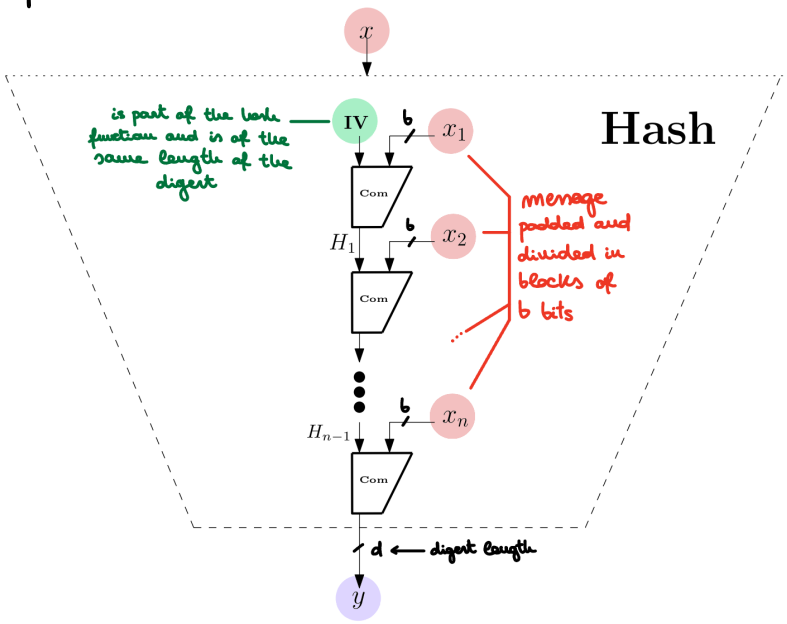
Hash properties

MAIN FEATURES

- one-way (EASY TO COMPUTE BUT DIFFICULT TO INVERT)
- Collision Resistance (COMPUTATIONALLY INFEASIBLE TO FIND TWO ARGUMENTS THAT ARE MAPPED IN THE SAME VALUE)
- Second Preimage resistance or weak collision (take $x \in \mathbb{Z}_2^*$; $f(x) = y$ => Find $s | f(s) = y$ => Knowing x e y find s)

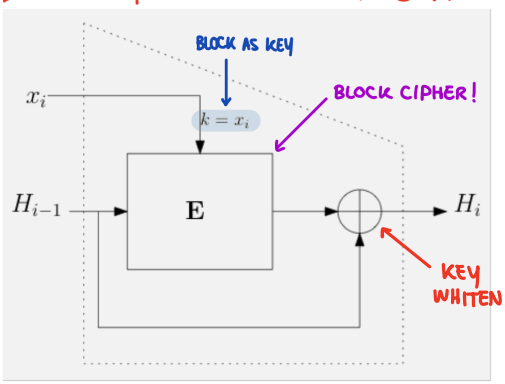
(x is a preimage of y and s is a second preimage of y)

- **Compression and Merkle-Damgard**: it is a method to construct an hash from a compression function Com. An IV is used. The message x is divided in blocks $x_1 x_2 \dots x_n$ which go through the loop:



The Com function can be implemented with a block cipher in which the block x_i are used as keys and there is a concatenation of block's encryption

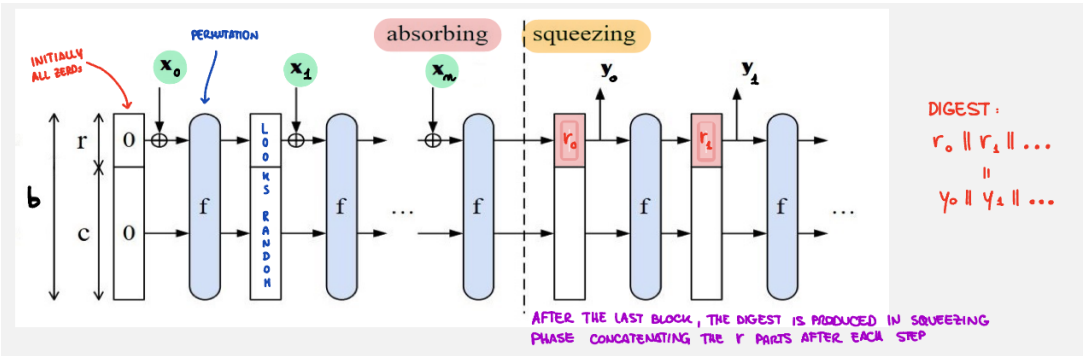
DAVIES-MEYER CONSTRUCTION OF COM:



$\mathrm{Enc}_{x_i}(H_{i-1}) \oplus H_{i-1}$ (Cf. Salsa20 & Chacha20)

- **Permutations and Sponge**:

Another way to construct an hash function is using a permutation function $f: \mathbb{Z}_2^b \to \mathbb{Z}_2^b$. It is a bijection (both injective and surjective), fast to compute. It is not necessarily a one way function but it MUST LOOK RANDOM. An example is the SPONGE FUNCTION:



The input **x** is divided in blocks of **x**i bits and there's a padding. The number **b = r + c** is called width, sum of the capacity **c** and the nitrate **r**. The state of the sponge is the content of a register of b bits.
f is a permutation function.
There are 2 phases:

- **absorbing:** the input r-bits of block xi are cored with the first r bits of the state. Then f is used to permute the state. This is done for all the blocks.
- **Squeezing:** the output is formed y = y0 || y1 || … by using the r bits of each yi

- **Definition of MAC (Msg AuthN Code)**:

DEFINITION 4.1 A message authenticatign code (or MAC) consists of three probabilistic polynomial-time algorithms (Gen, Mac. Vrfy) such that:
1. The key-generation algorithm Gen takes as input the security parameter 1" and outputs a key k with |k| ≥ n.

2. The tag-generation algorithm Mac takes as input a key k and a message m € {0, 1)*, and outputs a tag t. Since this algorithm may be random-ized, we write this as t + Mack (m).

3. The deterministic verification algorithm Vrfy takes as input a key k, a message m, and a tagt. It outputs a bit b, with b = I meaning valid and b = 0 meaning invalid. We write this as b:= Vrfy, (m, t).

It is required that for every n, every key k output by Gen(1"), and every m € {0, 1)*, it holds that Vrfy, (m, Mack (m)) = 1.
If there is a function I such that for every k output by Gen(1), algorithm Mack is only defined for messayes m € {0, 1}e(*), then we call the scheme a fixed-length MAC for messages of length e(n).

- Message authN experiment $MAC\text{-}forge_{\mathcal{A},\Pi}(n)$:
  1) A key is generated by running $Gen(1^n)$
  2) The adversary $\mathcal{A}$ is given input 1^n and oracle access to $MAC_K(\cdot)$.
     $\mathcal{A}$ eventually outputs (m,t). Let Q be the set of all queries that $\mathcal{A}$ asked its oracle.
  3) $\mathcal{A}$ succeeds if: a) $Vrfy_K(m,t) = 1$
     b) $m \notin Q$

- Given a message m, $MAC_K(m) := hash(K \| m) = t$

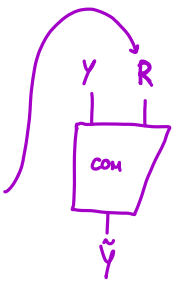
If $MAC_K(m) = hash(K \| m) = y$ with Merkel-Dagmar we have the pair (m, y) VALID PAIR

the goal is to construct another valid pair $(\tilde{m}, \tilde{y})$ from (m, y)

If I'm an $\mathcal{A}$ and I take $m \| R$ then even not knowing the key I will work on $K \| m \| R$ (length of one block)

$\Rightarrow$ $\tilde{m} = m \| R$ $\tilde{y} = hash(m \| R) = \tilde{y}$ and $(\tilde{m}, \tilde{y})$ will be also VALID!

LENGTH ATTACK

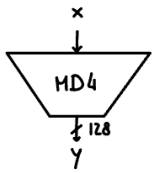


- MD4:

2 Overview and Design Goals

The first design goal is, of course, *security*: it should be computationally infeasible to find two messages M_1 and M_2 that have the same message digest. Here we define a task to be "computationally infeasible" if it requires more than 2^{64} operations.

The MD4 output digest has 128 bits.
MD4 is based in Merkle-Damgård idea.
There is a padding: a single "1" bit is appended to the message, and then enough zero bits are appended so that the length in bits of the padded message becomes congruent to 448 modulo 512. Then the 64 bit representation of the length of the pre-padded original message is appended. If the length of the original message needs more than 64 bits then only the 64 lower-order bits are used.



$x \| 100\ldots0 \| +64\,bits$

$\equiv 448 \bmod 512$

$448 + 64 = 0 \bmod 512$

$\Rightarrow x \leq 2^{64}$

The blocks are of 512 bits divided in 16 words (32 bits).
The main register (called 4-word buffer by Rivest) has 4 words A B C D , see bellow figure.

Another design goal of MD4 was to favor little-endian architectures.

- MD4 WHITE PAPER:

Step 3. Initialize MD buffer

A 4-word buffer (A, B, C, D) is used to compute the message digest. Here each of A, B, C, D is a 32-bit register. These registers are initialized to the following values (in hexadecimal, low-order bytes first):

word A: 01 23 45 67
word B: 89 ab cd ef
word C: fe dc ba 98
word D: 76 54 32 10

} IV = 128 bits

Step 4. Process message in 16-word blocks

We first define three auxiliary functions that each take as input three 32-bit words and produce as output one 32-bit word.

$$f(X,Y,Z) = XY \vee (\neg X)Z$$
$$g(X,Y,Z) = XY \vee XZ \vee YZ$$
$$h(X,Y,Z) = X \oplus Y \oplus Z$$

In each bit position f acts as a conditional: **if** x **then** y **else** z. In each bit position g acts as a majority function: if at least two of x, y, z are one, then g has a one in that position. The function h is the bit-wise *xor* or *parity* function. It is interesting

to note that if the bits of X, Y, and Z are independent and unbiased, the each bit of $f(X,Y,Z)$ is independent and unbiased, and similarly for $g(X,Y,Z)$ and $h(X,Y,Z)$.

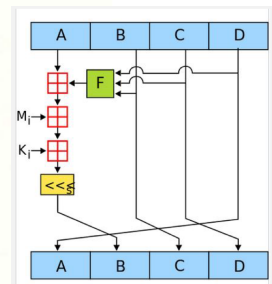
MD4 utilizes two "magic constants" in rounds two and three. The round two constant is $\sqrt{2}$ and the round 3 constant is $\sqrt{3}$. (See Knuth [6, page 660].) Here are their values in octal and hex (with high-order digits given first).

	Octal	Hex
Round 2 constant ($\sqrt{2}$):	013240474631	5A827999
Round 3 constant ($\sqrt{3}$):	015666365641	6ED9EBA1

two CONSTANTS are used

Do the following:

```
For i = 0 to N/16-1 do        /* process each 16-word block */
        Set X[j] to M[i*16+j], for j = 0, 1, ..., 15.
        Save A as AA, B as BB, C as CC, and D as DD.

        [Round 1]
          Let [A B C D i s] denote the operation
                A = (A + f(B,C,D) + X[i]) <<< s  .
          Do the following 16 operations:
                [A B C D 0  3]
                [D A B C 1  7]
                [C D A B 2  11]
                [B C D A 3  19]
                [A B C D 4  3]
                [D A B C 5  7]
                [C D A B 6  11]
                [B C D A 7  19]
                [A B C D 8  3]
                [D A B C 9  7]
                [C D A B 10 11]
                [B C D A 11 19]
                [A B C D 12 3]
                [D A B C 13 7]
                [C D A B 14 11]
                [B C D A 15 19]

        [Round 2]
          Let [A B C D i s] denote the operation
                A = (A + g(B,C,D) + X[i] + 5A827999) <<< s .
          Do the following 16 operations:
                [A B C D 0  3]
                [D A B C 4  5]
                [C D A B 8  9]
                [B C D A 12 13]
                [A B C D 1  3]
                [D A B C 5  5]
                [C D A B 9  9]
                [B C D A 13 13]
                [A B C D 2  3]
                [D A B C 6  5]
                [C D A B 10 9]
                [B C D A 14 13]
                [A B C D 3  3]
                [D A B C 7  5]
                [C D A B 11 9]
                [B C D A 15 13]

        [Round 3]
          Let [A B C D i s] denote the operation
                A = (A + h(B,C,D) + X[i] + 6ED9EBA1) <<< s .
          Do the following 16 operations:
                [A B C D 0  3]
                [D A B C 8  9]
                [C D A B 4  11]
                [B C D A 12 15]
                [A B C D 2  3]
                [D A B C 10 9]
                [C D A B 6  11]
                [B C D A 14 15]
                [A B C D 1  3]
                [D A B C 9  9]
                [C D A B 5  11]
                [B C D A 13 15]
                [A B C D 3  3]
                [D A B C 11 9]
                [C D A B 7  11]
                [B C D A 15 15]

        Then perform the following additions:
                A = A + AA
                B = B + BB
                C = C + CC
                D = D + DD
        (That is, each of the four registers is incremented by the value
        it had before this block was started.)

        end /* of loop on i */
```



One MD4 operation : MD4 consists of 48 of these operations, grouped in three rounds of 16 operations. F is a nonlinear function; one function is used in each round. M_i denotes a 32-bit block of the message input, and K_i denotes a 32-bit constant, different for each operation.

Step 5. Output

The message digest produced as output is A, B, C, D. That is, we begin with the low-order byte of A, and end with the high-order byte of D.

This completes the description of MD4.

4 WORDS 16 WORDS

COMP — comp is Hayer Devis that uses a BLOCK CIPHER with a key of 16 WORDS

4 WORDS

16 words

4 words — enc

- SHA-0-1-2 Read them on Crypto 8 because prof didn't add anything.

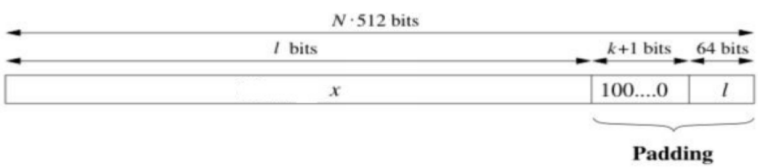
Let **Hash** be the hash function with digest of 512 bit obtained by using the permutation $f : \mathbb{Z}_2^{1024} \to \mathbb{Z}_2^{1024}$ given by

$$f(b_1, b_2, b_3, \cdots, b_{1024}) = (b_{1024}, b_1, b_2, \cdots, b_{1023})$$

SHA1 PADDING

sha1 padding uses permutation function

and using the sponge with $b = 1024, r = 512$ and the padding:



- **Permutation Based Crypto**: $f : \mathbb{Z}_2^b \to \mathbb{Z}_2^b$

1) f is not one-way 2) f is invertible and both f and f^{-1} are computationally feasible
3) f should be random

Now fixed $r < b$ $\quad f_r(x) = \text{Trunc}_r(f(x))$ first r bits of $f(x)$ $\quad f_r(x) : \mathbb{Z}_2^b \to \mathbb{Z}_2^r$

Difficult Problem is: Given $y \in \mathbb{Z}_2^r$ find $x \in \mathbb{Z}_2^b$ such that $f_r(x) = y$

$f = \left(\begin{bmatrix} \quad \end{bmatrix}^{r \text{ bits}}_{c \text{ bits}} \right)$ $\quad f_r\left(\begin{bmatrix} \quad \end{bmatrix}^{r \text{ bits}}_{c \text{ bits}} \right) = [\ ]$ r bits $\Rightarrow$ you have 2^c possible choices for c bits.

- **Hellman & Rainbow Tables**: Given y find x such that $f(x) = y$

1) **PRECOMPUTATION**: before try to solve the equation, when you just know the function f. Let's choose a domain A and a set of elements of it and a number of iterations t.

$$\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{bmatrix} \xrightarrow{f} \begin{bmatrix} f(a_1) \\ f(a_2) \\ \vdots \\ f(a_m) \end{bmatrix} \xrightarrow{f} \begin{bmatrix} f(f(a_1)) \\ f(f(a_2)) \\ \vdots \\ f(f(a_m)) \end{bmatrix} \xrightarrow{t \text{ iterations}} \begin{bmatrix} f^t(a_1) \\ f^t(a_2) \\ \vdots \\ f^t(a_m) \end{bmatrix}$$

Now I keep only this last column $\quad \begin{bmatrix} f^t(a_1) \\ f^t(a_2) \\ \vdots \\ f^t(a_m) \end{bmatrix}$ THIS IS A HELLMAN TABLE

2) **ATTACK**: Given y I check if y is in the values that I kept:

- If $y = f^t(a_i) \Rightarrow f^{t-1}(a_i) = x$ ✓
- If NOT, I compute $f(y) = \tilde{y}$ and check again ... after u iterations I will find

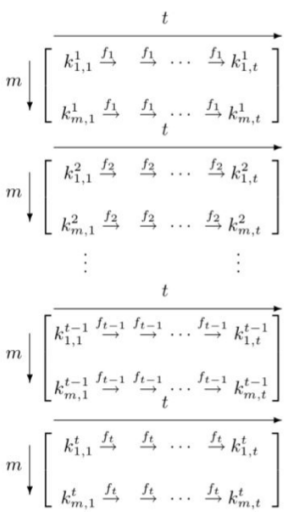
$$f^u(y) = f^t(a_i) \Rightarrow x = f^{t-u-1}(a_i)$$ ✓

- $R_i : A \to A \quad i = 1, \dots, t$ are bijections. $\quad R_i \circ f = f_i$

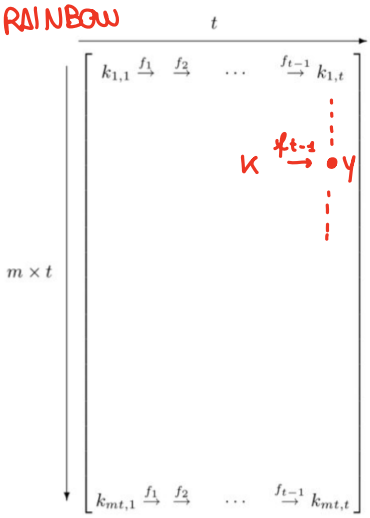
I can construct multiples Hellman tables of m elements, each of one using a different f_i.

A **Rainbow table** instead starts with $m \cdot t$ random elements and for each iteration a different f_i is used.

HELLMAN



RAINBOW



Now if you find y in the last column of a rainbow table:

$$y = f_{t-1}(x) = R_{t-1}(f(x))$$

$$\Rightarrow y = f(\underbrace{R_{t-1}(f_{t-2}(x))}_{x}) \to f_{t-1}(y) = f_{t-1}(f_{t-2}(x))$$

$$\Rightarrow R_{t-2}(x) = x$$ ✓

If y not in the last column $f_{t-1}(y) \xrightarrow{\text{NOT IN LAST COL}} f_{t-2}(f_{t-1}(y)) \xrightarrow{\text{NOT IN LAST COL}} f_{t-3}(f_{t-2}(f_{t-1}(y)))$...

WORST CASE:

- Hellman takes t iterations for t tables $= t^2$
- Rainbow tables worst case $\frac{t(t-1)}{2}$

Another disadvantage in H.T. is that if two elements are the same, also the next to them will be the same. In Rainbow this doesn't happen because we change f_i

- **KEY EXCHANGE:**

Key-Exchange Protocol

is a protocol Π i.e. a set of instructions for the parties say Alice and Bob, that starting from a security parameter n allows them to compute keys k_A and k_B.
The correctness requirement is that

$$\mathsf{k}_A = \mathsf{k}_B$$

so we can speak simply of *the* key $\mathsf{k} = \mathsf{k}_A = \mathsf{k}_B$ as the exchanged key.

- **DIFFIE-HELLMAN:**

Alice and **Bob** agree to use a prime number p and an element g of $\mathrm{GF}^*(p)$.

Alice		**Bob**
Pick RND $A \in \{1, \cdots, p-1\}$		
Compute $g^A \pmod p$		
	$\xrightarrow{g^A}$	
		Pick RND $B \in \{1, \cdots, p-1\}$
		Compute $g^B \pmod p$
	$\xleftarrow{g^B}$	
$\mathsf{k}_A = (g^B)^A$		$\mathsf{k}_B = (g^A)^B$

you need a huge prime to avoid the attacks of bruteforce

So $\mathsf{k} = \mathsf{k}_A = \mathsf{k}_B$ is the session key between **Alice** and **Bob**.

A and B are the private keys whilst g^A, g^B are the public keys.

The prime number is generated in multiple steps:

- First of all a number generator generates q which has an high probability of being prime since by the prime number theorem $\#Pr \approx n \log(n)$.
  To check if $q \in \mathbb{Z}^N$ is prime:
  $$q^{N-1} \pmod N \overset{?}{=} 1 \begin{cases} \text{NO} \rightarrow \text{REPEAT} \\ \text{YES} \rightarrow \text{PRIME!} \end{cases}$$
- Once we got q prime we generate $p = 2q + 1$

- **RSA:**