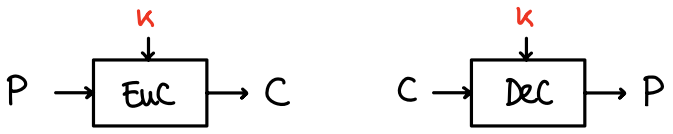
CRYPTOGRAPHY NOTES 23/24

PROF: DI SCALA

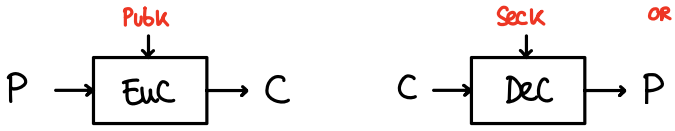
NICCOLO' LENTINI

- **Introduction to cryptography**: cryptography is the art of keeping messages secure and it's practiced by cryptographers. Cryptanalysts are practitioners of cryptanalysis, the art and science of breaking ciphertext.

- **Basics**: Encipher & Decipher (symmetric):



Encipher & Decipher (asymmetric):



OR VICEVERSA

NOTE: to use crypto, all parties must agree on all the elements defining the crypto system, that is the domain parameters of the scheme.

- **Kerchoffs principle**: the enemy knows the system (Shannon's Maxim). This means that security just depend on the secrecy of the key.

- TOY EXAMPLES:
  - VERNAM CIPHER or ⊕-Cipher or ONE-TIME-PAD: A e B agree on using $K = 01010$ as secret key. Now each P will be enc. as $C = Enc_K(P) = P \oplus K$ and since ⊕ is the inverse of itself $P = Dec_K(C) = C \oplus K$

  - CAESAR CIPHER or ⊞-cipher: the ⊞ operator means doing SUM and then mod by some number. For example mod 32. So $27 \boxplus 21 = 48 = 16 \pmod{32}$.
  In this case $C = Enc_K(P) = P \boxplus K$ and $P = Dec_K(C) = C \boxplus (-K)$

~~ MODULAR ARITHMETIC or CLOCK ARITHMETIC or DOING COMPUTATION WITH REMAINDERS:
Based on Theorem of Division that says:
given $b, N \in \mathbb{Z}$ $\exists\ q, r \in \mathbb{Z} \mid b = N \cdot q + r$ such that $0 \leq r \leq N-1$ => r is the result of the mod N

- e.g. mod 7: $10 \times 10 \times 10 \pmod 7$ = remainder of 10^3 divided by 7

getting the remainders already → $10 \times 10 \times 10 = 3 \times 3 \times 3 = 9 \times 3 = 2 \times 3 = 6$ ✓ :7

but in mod arithmetic I can replace any number with another number that gives the same remainder in the division ... for example $6 = -1$

Then I can do $10^{30} \pmod 7 = 10^{3 \cdot 10} = (10^3)^{10} = (-1)^{10} = 1$ ✓

Properties: if $a = b \pmod N$, $c = d \pmod N$ then:

- $a \pm c = b \pm d \pmod N$
- $a \cdot c = b \cdot d \pmod N$ $\quad \forall\ a,b,c,d \in \mathbb{Z}$

- e.g. Casting out Nines: it's a TEST to verify the correctness of a plain arithmetic operation you (mod 9) and check it's validity (mod 9).

DIM: $230178 \pmod 9 = 21 \pmod 9 = 3 \pmod 9$

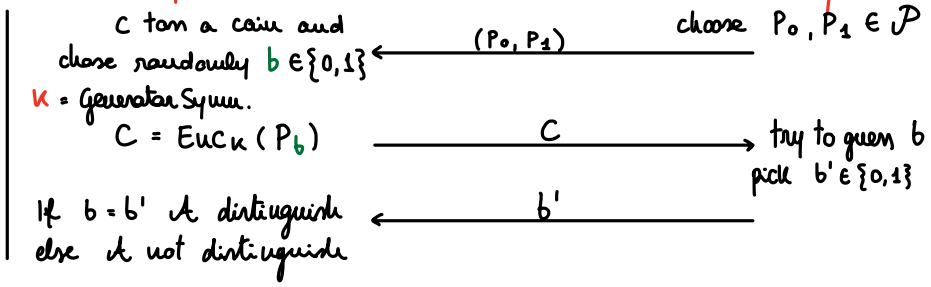
$8 + 7 \cdot 10 + 1 \cdot 10^2 + 0 \cdot 10^3 + 3 \cdot 10^4 + 2 \cdot 10^5 \overset{10^n = 1 \pmod 9}{=} 8 + 7 + 1 + 0 + 3 + 2 = 21 =$

$= 2 \cdot 10 + 1 = 2 + 1 = 3 \pmod 9$

- ROT CIPHER or CIRCULAR SHIFT $<<< r$: here $C = Enc_K(P) = P <<< K$ and $P = C <<< -K$

- **IND (INDISTINGUISHABILITY)**: is a property of an encryption scheme, for which the adversary is unable to distinguish pairs of cipher texts based on the message being encrypted.
  Defined with the IND experiment:

DEF// **IND-SECURE**: the symmetric crypto system $\Pi = (GEN, ENC, DEC)$ is IND-SECURE, if no adversary $\mathcal{A}$ can distinguish with probability better than $1/2$.

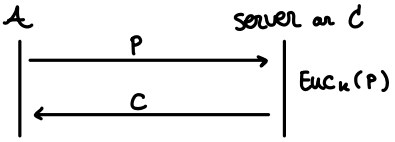


- **ATTACKS**: there are four general types of cryptanalytic attacks. Of course the cryptanalyst has full knowledge of the algorithms used.

1) **Ciphertext-Only-Attack (COA)**: the adversary has information on C only and wants to recover P or better the keys:
   GIVEN: $C_1 = Enc_K(P_1)$, $C_2 = \dots$ ecc...
   DEDUCE: $P_1, P_2, \dots$ or K or an algorithm to infer P from C

2) **Known-Plaintext attack (KPA)**:
   GIVEN: P_1, C_1; P_2, C_2 ecc...
   DEDUCE: K or an algorithm to infer P from C

3) **Chosen-Plaintext-attack (CPA)**: One of the most typical where $\mathcal{A}$ chose P and send it to $\mathcal{C}$ that encipher it.
   GIVEN: P_1, C_1; P_2, C_2; ecc...
   DEDUCE: K or an algorithm to infer P from C



The IND property can be also under chose attacks...

- **CPA-IND and Probabilistic Encryption (or non deterministic)**:

CPA-IND MEANS THAT THE CRYPTO PRIMITIVE IS PROBABILISTIC
REMEMBER

the CPA-IND experiment:
1) A key K is generated with GEN.
2) $\mathcal{A}$ chose two messages P_0, P_1 and send them to $\mathcal{C}$
3) $\mathcal{C}$ chose randomly $b \in \{0,1\}$ and generate $C = enc_K(P_b)$. Then send C to $\mathcal{A}$
4) $\mathcal{A}$ has ORACLE ACCESS to $Enc_K(.)$ and outputs $b' \in \{0,1\}$.
5) if $b' = b$ => $\mathcal{A}$ succeed

DEF// **CPA-IND SECURE**: the symmetric crypto system $\Pi = (GEN, ENC, DEC)$ is CPA-IND SECURE if no $\mathcal{A}$ can succeed with probability better than $1/2$.

! This system works because when $\mathcal{A}$ asks twice to encrypt the same P, then the cryptogram CHANGES!

time 0: $Enc_K(P) = C$
⋮
time n: $Enc_K(P) = \tilde{C}$
} PROBABILISTIC ENCRYPTION (CRYPTOSYSTEM)

VS

time 0: $Enc_K(P) = C$
⋮ =
time n: $Enc_K(P) = C$
} DETERMINISTIC CRYPTOSYSTEM

note that anyway $dec_K(C) = dec_K(\tilde{C})$

- **RANDOMNESS**: A pseudo-random sequence is a vague notion embodying the idea of a sequence in which each term is unpredictable to the uninitiated and whose digits pass a certain number of tests traditional with statisticians and depending somewhat on the uses to which the sequence is to be put.

- **PRNG (Pseudo Random Number Generator)**: A PRNG is a function $G : \mathbb{Z}_2^n \to \mathbb{Z}_2^{\ell(n)}$ with $\ell(n) > n$ (expansion factor) such that no adversary $\mathcal{A}$ succeed with probability $> 1/2$ the PRG IND experiment $PRG_{\mathcal{A},G(n)}$:

  a) A uniform bit $b \in \{0,1\}$ is chosen. If $b = 0$ then choose a uniform $r \in \{0,1\}^{\ell(n)}$. If $b = 1$ then choose a uniform $s \in \{0,1\}^n$ and set $r := G(s)$. (expands n bits in $\ell(n)$ bits)

  b) The adversary $\mathcal{A}$ is given r and outputs a bit b'.

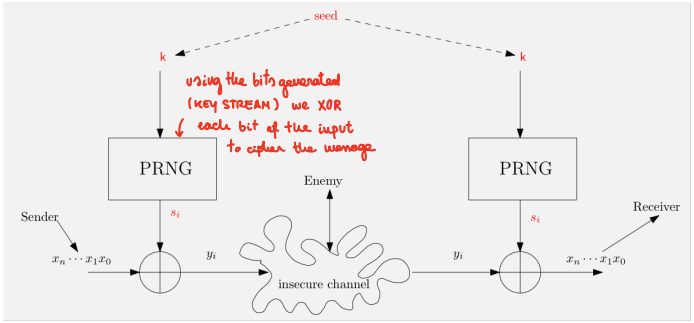
  c) The output of the experiment is defined to be 1 if $b' = b$ and 0 otherwise.

In this experiment $\mathcal{A}$ tries to understand if r is totally random or derives from a recursive function.

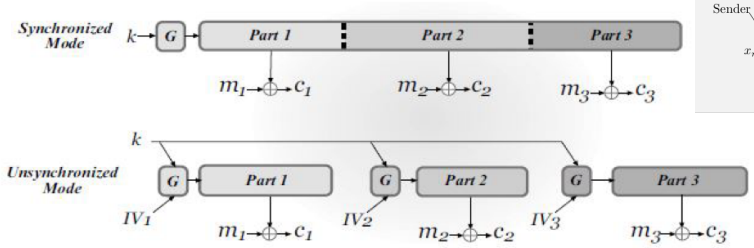
A PRNG is generally constructed from two algorithms (Init and GetBits). Init takes as input a seed s and an optional IV and outputs an initial state st_0. GetBits takes an input state st_i, outputs a bit y and update the state to st_{i+1}. The states st_i are the states in a big buffer or memory called "entropy pool".

$$St_0 = init(s, IV) \rightarrow \text{for } i = 1 \text{ to } \ell(n) \{ (y_i, st_i) = GetBits(st_{i-1}) \} \rightarrow \text{return } y_1 \dots y_{\ell(n)}$$

PRNG are used in stream cipher where sender and receiver are syncronized and share the same seed.



2.3 Stream Ciphers & Operation modes



2.3.1 Synchronized

Using (Init, GetBits) the parties construct $G(\mathsf{k}, 1^l)$ running l-times using a shared k. Concretely both begin by computing $\mathsf{st}_0 = \mathsf{Init}(\mathsf{k})$.
To encrypt the first message m_1 of length l_1 the sender runs GetBits l_1-times beginning at st_0 getting bits $y_1 \cdots y_{l_1} = \mathsf{pad}_1$ along with an updated state st_{l_1}. It sends $c_1 = \mathsf{Enc}_\mathsf{k}(m_1) = \mathsf{pad}_1 \oplus m_1 = \mathsf{pad}_1 \oplus m_1$.

Once the other party receive c_1 it runs GetBits l_1-times beginning at st_0 getting bits $y_1 \cdots y_{l_1} = \mathsf{pad}_1$ along with an updated state st_{l_1}. It uses pad_1 to recover $m_1 = c_1 \oplus \mathsf{pad}_1$.
Later the sender to encrypt a second message m_2 will run GetBits at the state st_{l_1} to obtain a second $\mathsf{pad}_2 = y_{l_1+1} \cdots y_{l_1+l_2}$ and an updated state $\mathsf{st}_{l_1+l_2}$ and so on.

NOTE 2.3.1

We remark that in this mode, the stream cipher does not need to use the initialization IV.

2.3.2 Unsynchronized

In this case ciphertexts are pairs :

$$\mathsf{Enc}_k(m_j) = \langle IV_j \,,\, G(k, IV_j, 1^{|m_j|}) \oplus m_j \rangle$$

where $G(k, IV_j, 1^{|m|})$ is the pad obtained by running $\mathsf{Init}(k, IV_j)$. Decryption is performed in the natural way.

NOTE 2.3.2

Tipical mistakes are related to the optional IV e.g. a fixed constant in soft or hardware. If IV is random then $\mathsf{Enc}_k(m) = \langle IV \,,\, G(k, IV, 1^{|m|}) \oplus m \rangle$ is CPA-secure. But not CCA-secure.

- **LEHMER GENERATOR** (linear Caugruental generator): One of the fiints PRNG is an applicatian af modular arithmetic. It uses:

$G : \mathbb{Z}_2^u \longrightarrow \mathbb{Z}_2^{\ell(u)}$ such that given $S_0 \in \mathbb{Z}_2^u$ (SEED) $\longrightarrow$ $G(S_0) = S_0 \,||\, S_1 \,||\, \dots \,||\, S_{\ell(u)}$

Given $S_0 \in \mathbb{Z}_2^u$ $\exists$ a, b, m where $S_1 = a \cdot S_0 + b \pmod m$

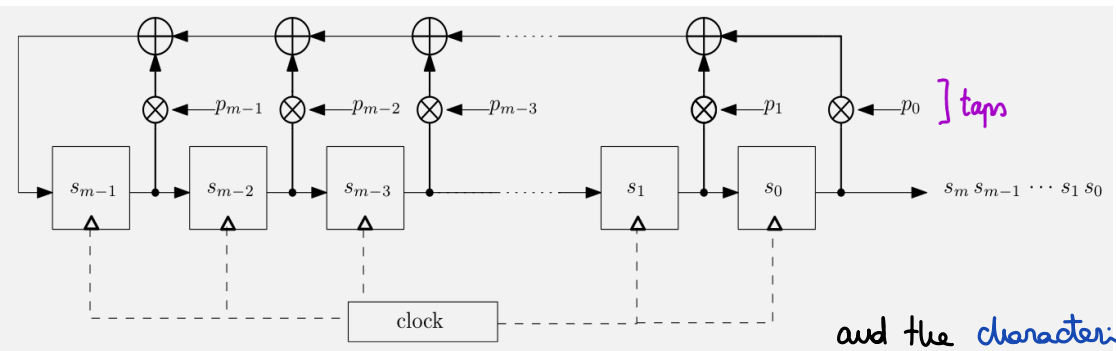
$S_2 = a \cdot S_1 + b \pmod m$

...

$$S_i = a \cdot S_{i-1} + b \pmod m$$

- **STREAM CIPHER : LFSR**

Linear feedback shift register is a shift register where state trauritian is linear.
The state s is a row vectar af m bits ($s \in \{0,1\}^m$ or $s \in \mathbb{Z}_2^m$).
The sequence af bits in the rightmast paritian af the state is the autput streaun (key streaun).
The bits in the LFSR state that influence its behaviar are called TAPS.
It is campased by a series af flipflop :



The output S_m is computed as a feedback :

$$S_m = f(s) = \sum_{j=0}^{m-1} S_j \cdot P_j$$

the feedback polynomial is:

$$P(x) = 1 + p_{m-1}x + \dots + p_1 x^{m-1} + p_0 x^m$$

and the characteristical polynomial is:

$$\chi_L(x) = x^m P\left(\frac{1}{x}\right) = p_0 + p_1 x + \dots + p_{m-1} x^{m-1} + p_m x^m$$

Here the involved linear map L is:

$$L = \begin{bmatrix} p_{m-1} & 1 & 0 & 0 & \cdots & 0 \\ p_{m-2} & 0 & 1 & 0 & \cdots & \vdots \\ \vdots & 0 & 0 & \ddots & & 0 \\ p_1 & \vdots & \vdots & & \cdots & 1 \\ p_0 & 0 & 0 & \cdots & & 0 \end{bmatrix}$$

Being the state $S = [s_{m-1}\, s_{m-2} \dots s_1\, s_0]$ the trauritian to the state S' is given by the multiplicatian:

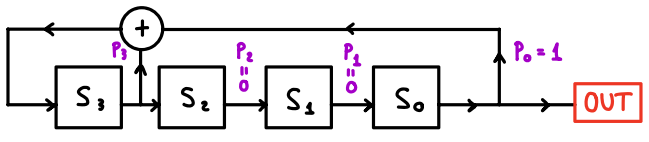
$$S' = S \cdot L$$

! if the initial state is all 0 then the state will always be 0

- EXAMPLE : $\chi_L(x) = x^4 + x^3 + 1$ $\Rightarrow$ $\chi_L(x) = p_0 + p_1 x + p_2 x^2 + p_3 x^3 + x^4$

$p_0 = 1$, $p_1 = 0$, $p_2 = 0$, $p_3 = 1$

then the LFSR will be :



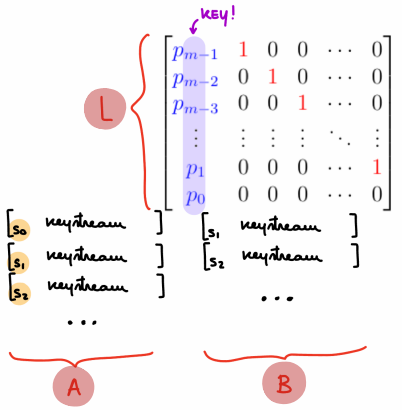
considering $S = 1000$ we have $L = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$

$$\Rightarrow S' = [1000] \cdot \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} = [1100]$$

If we continue the iterations, then we will find a repetition, where at a certain point the state will be the same of the starting point. The number of iterations to find the repetition defines the PERIOD.
In this case with starting S = 1000 after 9 iterations there's a repetition.
With this LFSR we have 4 bit state so the number of different states possible is $2^{m-1} = 15$.
The polynomial $p_0 \dots p_n$ that traverse all the possible states is defined as PRIMITIVE POLYNOMIAL (EXCEPT THE ZERO STATE)

- **Attack an LFSR:**

A cryptoanalyst is able to recover a sequence of the keystream S_i and wants to discover the key that represents the bits comparing the taps (i.e. the characteristical polynomial). This is a KPA attack.



$$\Rightarrow A \cdot \underbrace{L}_{\text{UNKNOWN}} = B \Rightarrow \boxed{L = A^{-1} \cdot B}$$

If we're producing a sequence long m ($S_m S_{m-1} \dots S_1 S_0$) then the possible combination for the characteristical polynomial are 2^m.

- **Attack to a 3bits LFSR:** $S = S_5 S_4 S_3 S_2 S_1 S_0 = 011101$ $\quad m = 2$

$P = P_2 P_1 P_0 = 011$

$$L = \begin{bmatrix} P_2 & 1 & 0 \\ P_1 & 0 & 1 \\ P_0 & 0 & 0 \end{bmatrix}$$

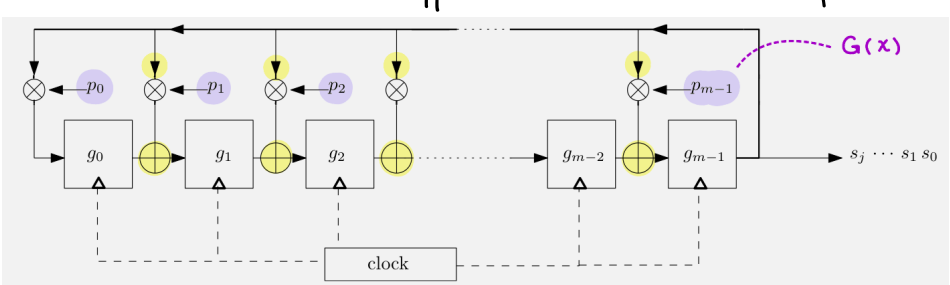
$$A = \begin{bmatrix} S_2 & S_1 & S_0 \\ S_3 & S_2 & S_1 \\ S_4 & S_3 & S_2 \end{bmatrix} \begin{bmatrix} S_3 & S_2 & S_1 \\ S_4 & S_3 & S_2 \\ S_5 & S_4 & S_3 \end{bmatrix} = B$$

$\Rightarrow$ Using the Gauss inversion to find A^{-1}

$$\underbrace{\begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{Id}$$

1. Reduce the matrix in a scaled (?) matrix
2. Reduce the elements above the pivot to 0
3. Make the pivot equal to 1

- **GALOIS LFSR:** the difference is in the way the register is updated.



Here the state $s = [g_0 g_1 \dots g_{m-1}]$ is regarded as a polynomial:

$$S(x) = g_0 + g_1 x + \dots + g_{m-2} x^{m-2} + g_{m-1} x^{m-1}$$

When the system is clocked the new state is: $S'(x) = x \otimes S(x)$
where $\otimes$ is the Galois cipher with $G(x) = X_L(x) = x^m + p_{m-1} x^{m-1} + \dots + p_1 x + p_0$ $\Rightarrow$ $x \times S(x) = g_0 x + g_1 x^2 + \dots + g_{m-2} x^{m-1} + g_{m-1} x^m$
and then get the remainder of the division by $X_L(x)$.
The trick to perform this division consists in replacing x^m with an equivalent value (mod G).

Since $G(x) = 0 \pmod G$ we have $x^m + p_{m-1}x^{m-1} + \dots + p_1 x + p_0 = 0 \pmod G$ and we can say $x^m = p_{m-1}x^{m-1} + \dots + p_1 x + p_0$

Now: $x \cdot S(x) = g_0 x + g_1 x^2 + g_2 x^3 + \dots + g_{m-1} x^m = g_0 x + g_1 x + \dots + g_{m-1}(p_{m-1}x^{m-1} + \dots + p_1 + p_0) =$

$= g_{m-1} \cdot p_0 + (g_0 + g_{m-1} \cdot p_0) x + \dots + (g_{m-2} + g_{m-1} \cdot p_{m-1}) x^{m-1}$

Here the linear map is $L = \begin{bmatrix} 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \\ \vdots & & & \ddots & & \vdots \\ 0 & & & & \ddots & 1 \\ p_0 & p_1 & p_2 & \cdots & & p_{m-1} \end{bmatrix}$ and the passage to a new stage S' is given by: $S \cdot L = S'$

- NOTE: LFSR are used in crypto inside crypto primitives, they are building blocks that don't work alone!

- **STREAM CIPHERS: ARX**

ARX architecture consists of a long chain of three simple operations:
- Addition
- Exclusive-or, producing the XOR $\oplus$
- Rotation $<<<$ of constant number of bits to the left

These three simple operation permits to have HIGH SPEED and SECURITY.

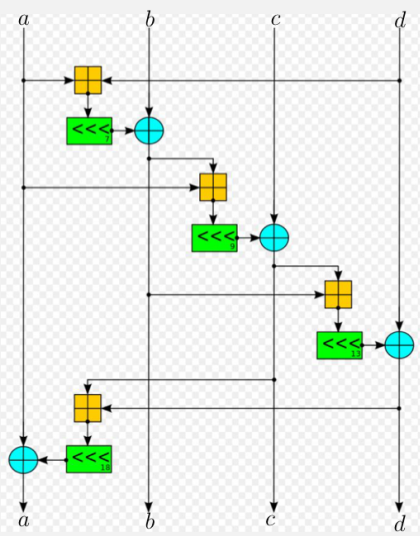
- **SALSA20**: expands a 256 bit key and a 64 bit nonce into a 2^{70} byte stream. It encrypts a b-byte plaintext by xoring the plaintext with the first b bytes of the stream and discarding the rest. It decrypts a b-byte ciphertext by xoring the ciphertext with the first b-byte of the stream.

The initial state is a 4×4 matrix of "words" of $32 = 2^5$ bits. So this matrix is $2^5 \cdot 2^4 = 64$ bytes. The diagonal is CONS filled. The key words contain the expansion of the 32-byte K. The words POS are used as a counter from 0 to $\lceil \frac{b}{64} \rceil - 1$ to construct the stream flow to encrypt a b-byte plaintext.

Cons	Key	Key	Key
Key	Cons	Nonce	Nonce
Pos	Pos	Cons	Key
Key	Key	Key	Cons

Each of these states, counted by the POS words, is scrambled 10 times each of 2 rounds using the quarter-round function $QR(a,b,c,d)$.

$$b \oplus = (a+d) <<< 7$$
$$c \oplus = (b+a) <<< 9$$
$$d \oplus = (c+b) <<< 13$$
$$a \oplus = (d+c) <<< 18$$



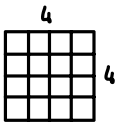
=> FIRST 2 ROUNDS —

QR(0,4,8,12)
QR(5,9,13,1)
QR(10,14,2,6)
QR(15,3,7,11)

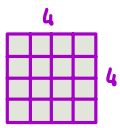
QR(0,1,2,3)
QR(5,6,7,4)
QR(10,11,8,9)
QR(15,12,13,14)

The output is the ADD in $\mathbb{Z}_{2^{32}}$, of the matrix after 20 rounds with the initial matrix. This last ADD is FUNDAMENTAL, otherwise since each round is invertible, it would be possible to go back to the initial matrix and recover the key!

- **EXAMPLE:**
  - length(M) ≤ 512

M →  to encrypt → C = M ⊕ keystream USING SALSA20

=> INITIAL STATE =  10 times 2 ROUNDS ~~~> QR FUNC. + LAST ADD

KEY STREAM  → C = M ⊕ keystream

- If length(M) > 512 then it's splitted in M_i of length = 512 on which the same operation is performed.

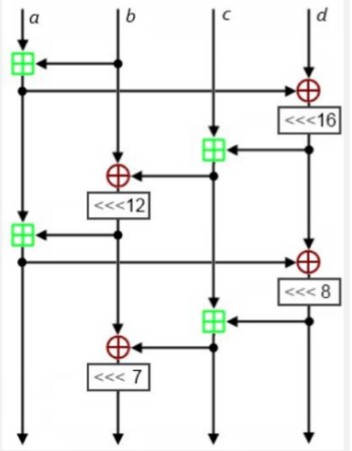
- **CHACHA20**: Also chacha20 expands a key of 256 bit to a stream of 2^{73} bits. The difference between chacha and salsa is in the QR function and the initial state. The main goal is to improve the diffusion of bits.

INITIAL STATE:

Cons	Cons	Cons	Cons
Key	Key	Key	Key
Key	Key	Key	Key
Pos	Pos	Nonce	Nonce

QR FUNCTION

$a += b$
$d \oplus= a$
$d = d <<< 16$
$c += d$
$b \oplus= c$
$b = b <<< 12$
$a += b$
$d \oplus= a$
$d = d <<< 8$
$c += d$
$b \oplus= c$
$b = b <<< 7$



FIRST TWO ROUNDS

QR(0, 4, 8, 12)
QR(1, 5, 9, 13)
QR(2, 6, 10, 14)
QR(3, 7, 11, 15)

QR(0, 5, 10, 15)
QR(1, 6, 11, 12)
QR(2, 7, 8, 13)
QR(3, 4, 9, 14)

- **SIMPLE PRNG: BLUM-BLUM-SHUB**

3.2.1 Blum-Blum-Shub PRNG

Also known as the $x^2 \pmod N$ generator. The stream of bits is $b_i = \text{parity}(x_i)$ where $N = p \cdot q$, p, q are two prime numbers and $p, q \equiv 3 \pmod 4$. The x_i sequence starts with a seed x_0 from $(\text{mod } N)$ and the sequences is recursively generated as :

$$x_{i+1} \equiv x_i^2 \pmod N$$

$x_0 \to b_0$ parity bit
$x_0 \to x_1 \to \dots$
$b_0 \quad b_1 \quad \dots$
stream of bits

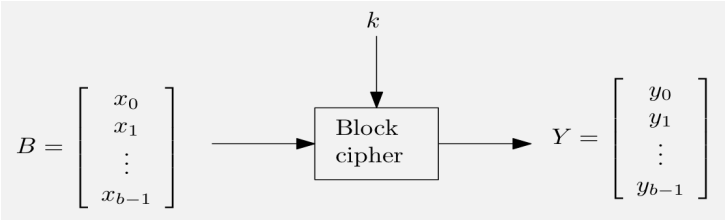
NOTE 3.2.2

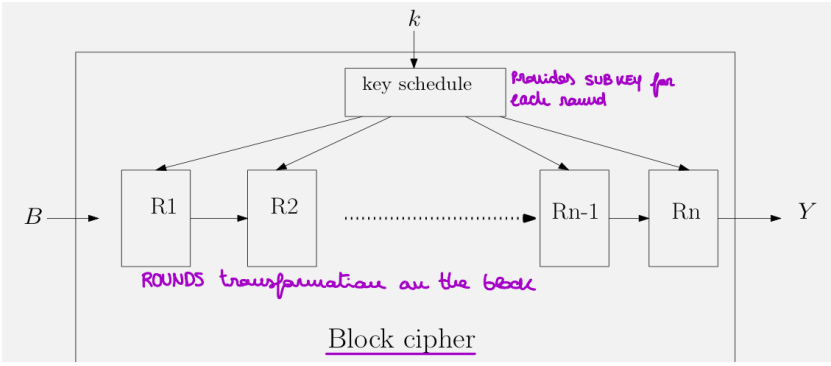
Here three properties of the PRNG [BBS86, pag. 373]:

1. Knowledge of N and x_{j_0} allows to compute the forward stream of bits $b_{j_0} b_{j_0+1} \cdots$.
2. Computing backward $\dots x_{j_0-2} x_{j_0-1}$ from x_{j_0} is equivalent to computing the factorization $N = p \cdot q$.
3. The factorization $N = p \cdot q$ to guess b_{j_0-1} from x_{j_0} with probability greater than $1/2$.

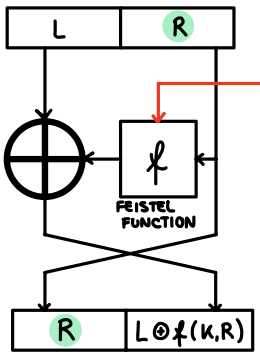
- if you know one of the sequence x_j since N is public you can compute all the sequence from the position j
- If somebody can extract the square root
- If somebody knows x_{10} and can extract x_9 with prob. $> 1/2$ then you have a great prob. of factorize N.

- **BLOCK CIPHERS**: Algorithms that take a KEY and a BLOCK and give as result another BLOCK.





- **FEISTEL ROUND**: one of the first block cipher that works as follow.



Just half of the block is transformed!

SUBKEY: different subkeys for different rounds following the key schedule.

Note that to decipher, the Feistel rounds are the same. It is only needed to invert the order of the subkeys delivered by the key schedule algorithm.

- **SPN (SUBSTITUTION - PERMUTATION NETWORK)**: Here we have a division of the 16 bit data block, into 4 bit sub-blocks each of that forms an input to a 4×4 S-box implemented with a lookup table. The most fundamental property of an S-box is that it is a NON LINEAR MAPPING.

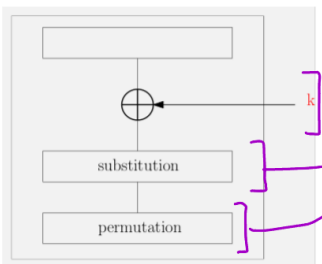
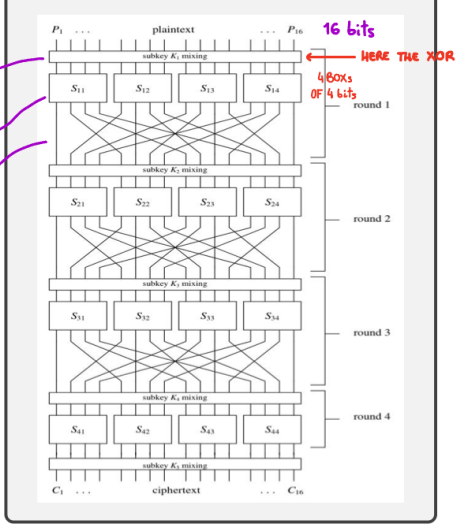


Figure 4.1.7: Toy SPN



TOY BLOCK CIPHER

S-BOX or LOOKUP TABLE

Figure 4.1.8: S-Box (Substitution) OR CONFUSION → you introduce some CAOS

input	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
output	E	4	D	1	2	F	B	8	3	A	6	C	5	9	0	7

SUBSTITUTION FOLLOWS THIS TABLE

e.g. 0000 → EEE0

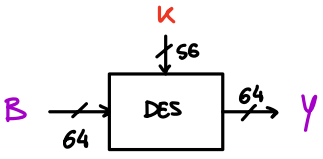
CONF. and DIFFUS. are terms introduced by SHANNON

Permutation of the wires!
input goes in output

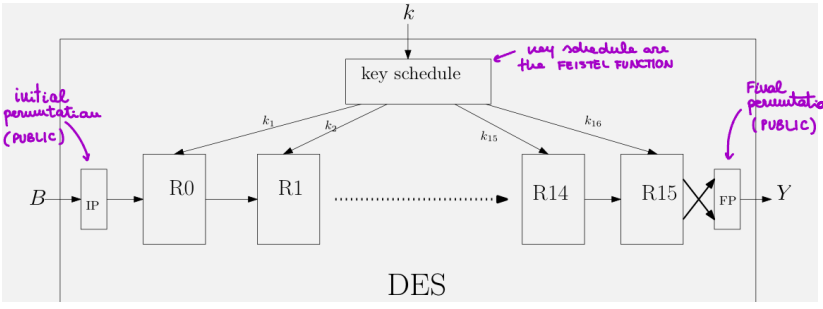
Figure 4.1.9: Permutation OR DIFFUSION (bits are spread casually → (diffusion of bits)

input	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
output	1	5	9	13	2	6	10	14	3	7	11	15	4	8	12	16

- **DES (DATA ENCRYPTION STANDARD):** It's a feistel cipher and it is BIT ORIENTED.
  It receives a 64 bit block and with a 56 bit key produces a 64 bit block.



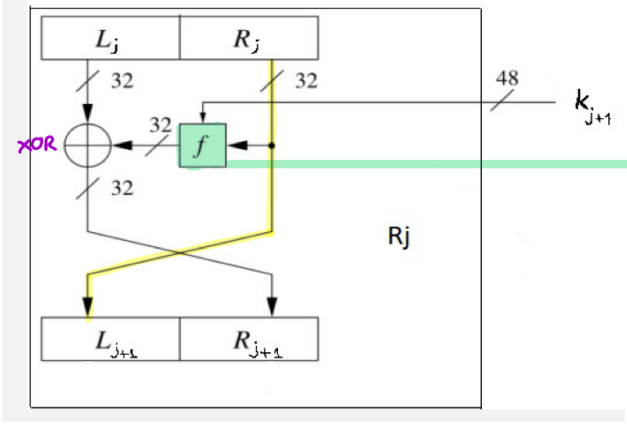
NOTE: the strength of the algorithm depends JUST ON THE KEY, on its length and secrecy. The rounds and the operations are public.

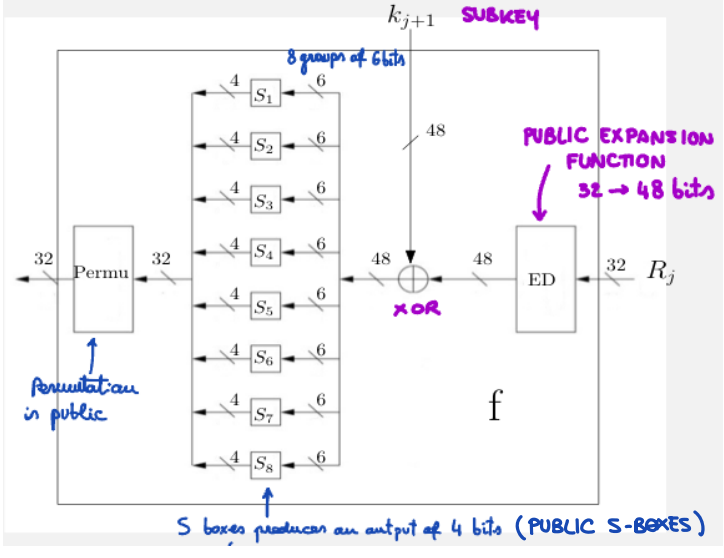


Besides the 16 rounds DES has an initial permutation IP and a block exchange between the last round and the final permutation $FP = IP^{-1}$.

DES rounds are feistel.

- **DES FEISTEL ROUND:**





to construct the S box has been followed some properties

NOTE 4.2.6

The eight S-box (substitution box) $S_1, \cdots, S_8$ are different from each other.

related with gaussian elimination algorithm that permits to solve a system in a fast way

S-BOX NON LINEARITY

S-boxes are lookup tables i.e. a map from input to output which uses the input as an index to lookup.

Here some of the guiding ideas about the design of the S-box:

1. No single output bit should be too close to a linear combination of the input bits.
2. If the lowest and the highest bits of the input are fixed and the four middle bits are varied, each of the possible 4-bit output values must occur exactly once.
3. If two inputs to an S-box differ in exactly one bit, their outputs must differ in at least two bits.
4. If two inputs to an S-box differ in the two middle bits, their outputs must differ in at least two bits.
5. If two inputs to an S-box differ in their first two bits and are identical in their last two bits, the two outputs must be different.
6. For any nonzero 6-bit difference between inputs, no more than 8 of the 32 pairs of inputs exhibiting that difference may result in the same output difference.
7. A collision (zero output difference) at the 32-bit output of the eight S-boxes is only possible for three adjacent S-boxes.

Here is S_1:

S_1	x0000x	x0001x	x0010x	x0011x	x0100x	x0101x	x0110x	x0111x	x1000x	x1001x	x1010x	x1011x	x1100x	x1101x	x1110x	x1111x
0yyyy0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0yyyy1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
1yyyy0	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
1yyyy1	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

How does it work?
e.g. 110101 → 0011 (3)
col
row

⚠ S-Boxes are the most crucial elements of DES because they introduce a non-linearity to the cipher:
$S(a) \oplus S(b) \neq S(a \oplus b)$

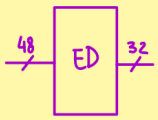
- **The expansor in DES:**

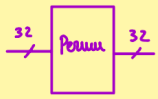
The permutation Perm of f introduce "Shannon's difussion". Namely, the output 4 bits of the S-box are permuted so they hit several S-boxes in the next round.
Thus, the combined effect of the expansor ED, the S-boxes and the permutation Perm assure that every bit of the output block y has a strong dependence on all bits of the key and all bits of the input block x. This is the so called avalanche effect.

Input 1 is connected with output 32 etc...

ED

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1





connect the input 1 to the output wire 16

Permu

16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

- **DOUBLE ENCRYPTION:** having a cipher Enc_K using a key $|K| = 56$ bits you can use another $\tilde{K}$ and compare the operations: $Enc_K(Enc_{\tilde{K}}(P)) = T_{K\tilde{K}}(P)$

Then you have double encryption. Even if we use two keys of 56 bits, the security level is not 112 bits.
This because if an attacker knows P_0 and C_0 where $C_0 = T_{K\tilde{K}}(P_0)$, he can discover the keys with at most $|K|+1 = |\tilde{K}|+1$ iterations (Meet In the Middle attack).

The attacker computes two tables, one with all the possible encryption from P_0 to $enc_{K_i} P_0$, and one with all the possible decryption from C_0 to $dec_{K_j} C_0$. So in total $2^{56} + 2^{56} = 2 \cdot 2^{56} = 2^{57}$ rounds
Then the attacker looks for a correspondence in the tables such that $enc_{K_i} P_0 = dec_{K_j} C_0$ to find the values of K and $\tilde{K}$.

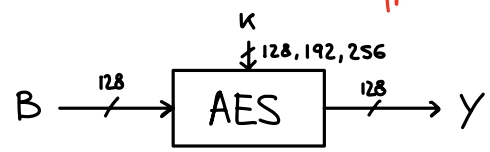
(HE LIKES THIS TERM) → SECURITY LEVEL IS ≈ 57 bit

- **TRIPLE ENCRYPTION:** triple application of the same cipher is instead considered secure. Usually implemented as EDE (ENC, DEC, ENC):

$$DES_{K_3} \circ DES^{-1}_{K_2} \circ DES_{K_1}$$

Here the security level is ≈ 112 bit

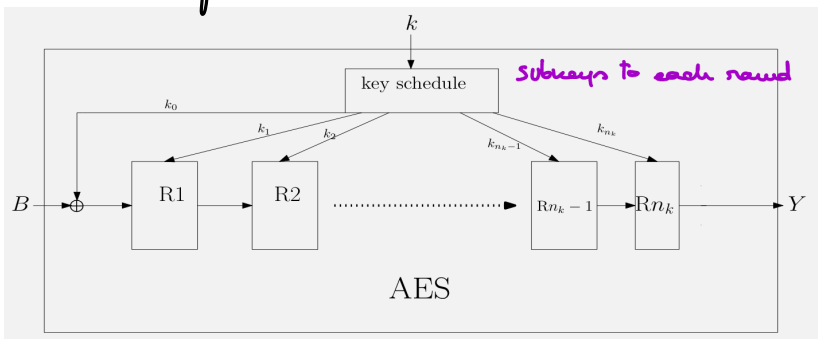
- **AES (Advanced Encryption Standard):** AES is a BYTE ORIENTED (the work unit is the byte) cipher.



keys of different length are which depend the number of rounds per encryption

$$K \begin{cases} 128\text{ bit} \cdots 10 \\ 192\text{ bit} \cdots 12 \\ 256\text{ bit} \cdots 14 \end{cases} \text{Rounds}$$

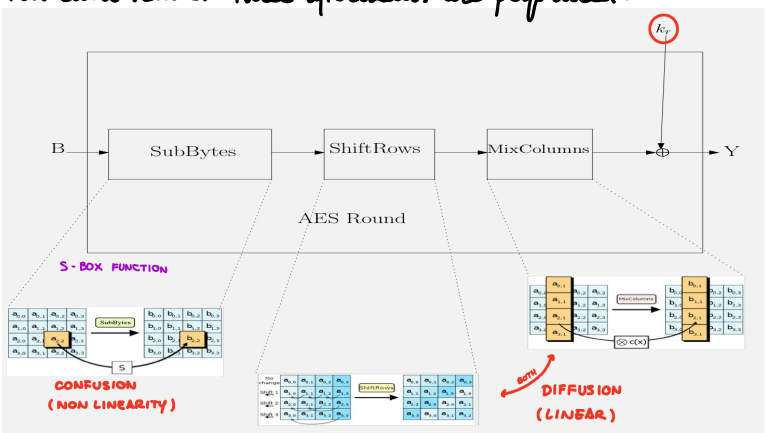
AES rounds for each block:



Each block is regarded as a 4×4 matrix of 16 bytes.

16 portions × 8 bits = 128 bits BLOCK

For each round three operations are performed:



- **SUBBYTES**: the Sbox function takes a bytes a_{ij} and returns another byte b_{ij}. It regards a_{ij} as a polynomial of degree 7: $a_{ij} = a_0 + a_1 x + \dots + a_7 x^7$. All the operations are performed in modular arithmetic with the polynomial $G(x) = x^8 + x^4 + x^3 + x + 1$.

So given a_{ij} is possible to construct the inverse $r = r_0 + r_1 x + \dots + r_7 x^7$ such that $r \cdot a_{ij} = 1 \pmod{G(x)}$

Then given $A = \begin{bmatrix} \dots \\ \dots \end{bmatrix}$ and $v = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$ (NOT RANDOM) $\quad S(a_{ij}) = A \cdot \begin{bmatrix} r_0 \\ \vdots \\ r_7 \end{bmatrix} + v = \begin{bmatrix} s_0 \\ \vdots \\ s_7 \end{bmatrix}$

The computation of the inverse is a HIGHLY NON-LINEAR computation since $(a+b)^{-1} \neq a^{-1} + b^{-1}$

- **SHIFT ROWS**: Just shifting rows

- **MIXCOLUMNS**: Also this operation is related to a modular polynomial $Q(z) = z^4 + 1$. Each column is regarded as a polynomial $a_0 + a_1 z + a_2 z^2 + a_3 z^3$ and the mixcolumns works as follow:

Pass from $\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \rightarrow \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$ $\quad b_0 + b_1 z + b_2 z^2 + b_3 z^3 = (3z^3 + z^2 + z + 2) \times (a_0 + a_1 z + a_2 z^2 + a_3 z^3) \pmod{z^4 + 1}$

attention! In the Galois field multiplication are not as in the real field: consider $3 = 00000011 = x + 1$

$$3 \cdot 3 = (x+1)(x+1) = x^2 + \overset{1+1=0}{x+x} + 1 = x^2 + 1 = 00000101 = 5 \quad \Rightarrow \underline{3 \times 3 = 5}$$

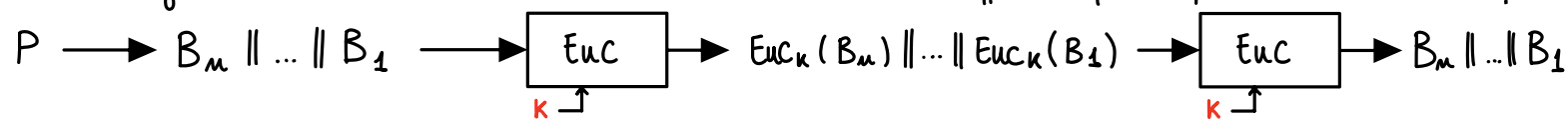
- **NOTE**: you can consider a ROUND as a single block cipher. So passing through different rounds means having a COMPOSITION OF BLOCK CIPHERS => MORE ROUNDS MEANS MORE SECURITY (that's why we use rounds).

- **BLOCK CIPHERS : OPERATION MODES**, how to use a block cipher.

A block cipher is much more than just an encryption algorithm. It can be used as a versatile building block with which a diverse set of cryptographic mechanisms can be realized.

- **ECB, Electronic Code Book** :

The message P is divided into blocks, and each block is encrypted separately with the same key K.



- **Advantages**: Synchronization is not necessary, blocks can be independently decrypted. Enc / Dec can be done in parallel. Errors on one block don't affect other blocks.

- **Disadvantages**: There is not integrity protection, attackers can invert / modify / filter blocks. There is a lack of DIFFUSION, because ECB encrypts identical plaintext blocks in identical ciphertext blocks. So if the key is not changed is possible to detect identical blocks. For this reason ECB doesn't have the IND-PROPERTY since an adversary can immediately guess the value of the coin.

ECB is a deterministic mode which leaves on the ciphertext too much information of the plaintext. It is not recommended for use in cryptographic protocols at all.

The name Electronic Codebook comes from the fact that given a key k each clear block is cipher into a unique cipher block. So we can imagine a huge book in which each line contains the pair clear block / cipher block. Such a book can be regarded as a "code".