# Módulo II

## Comunicaciones seguras

- Tema 3: Protección de la confidencialidad
- Tema 4: Cifrados de clave secreta
- Tema 5: Distribución de claves

# Comunicaciones Seguras

En:

- SSL/TLS 

- TOR 

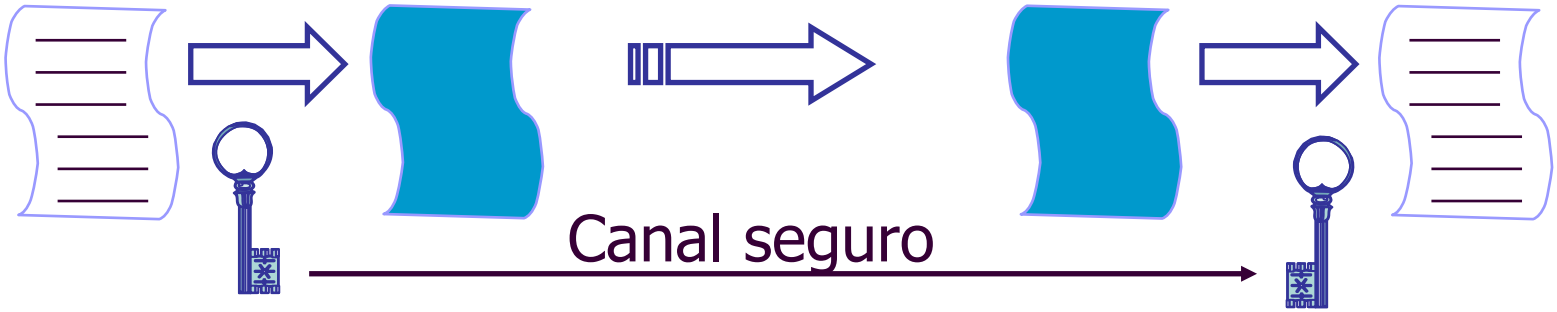
- GPS 

- Telefonía Móvil 

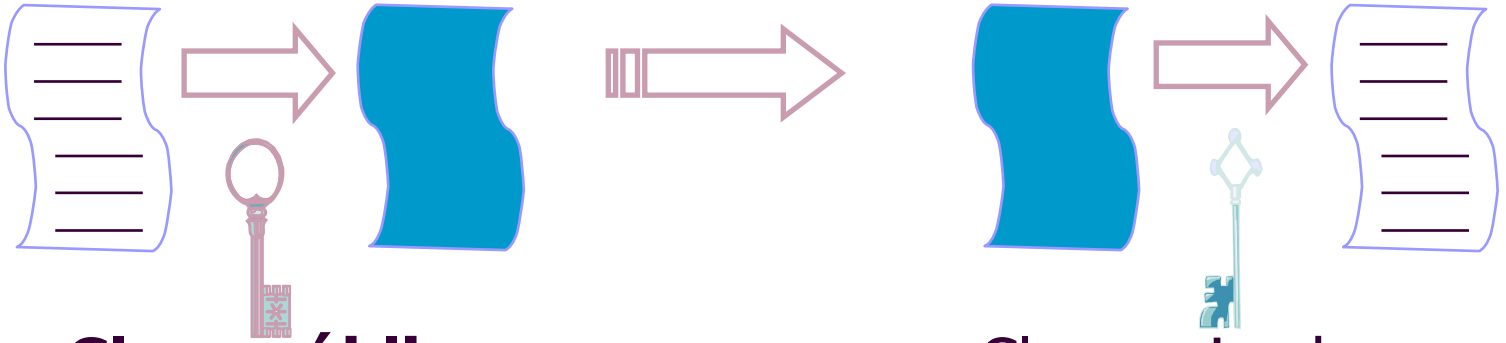
- Smartcards 

# Criptografía Moderna

– **Criptosistemas Simétricos o de Clave Secreta:**

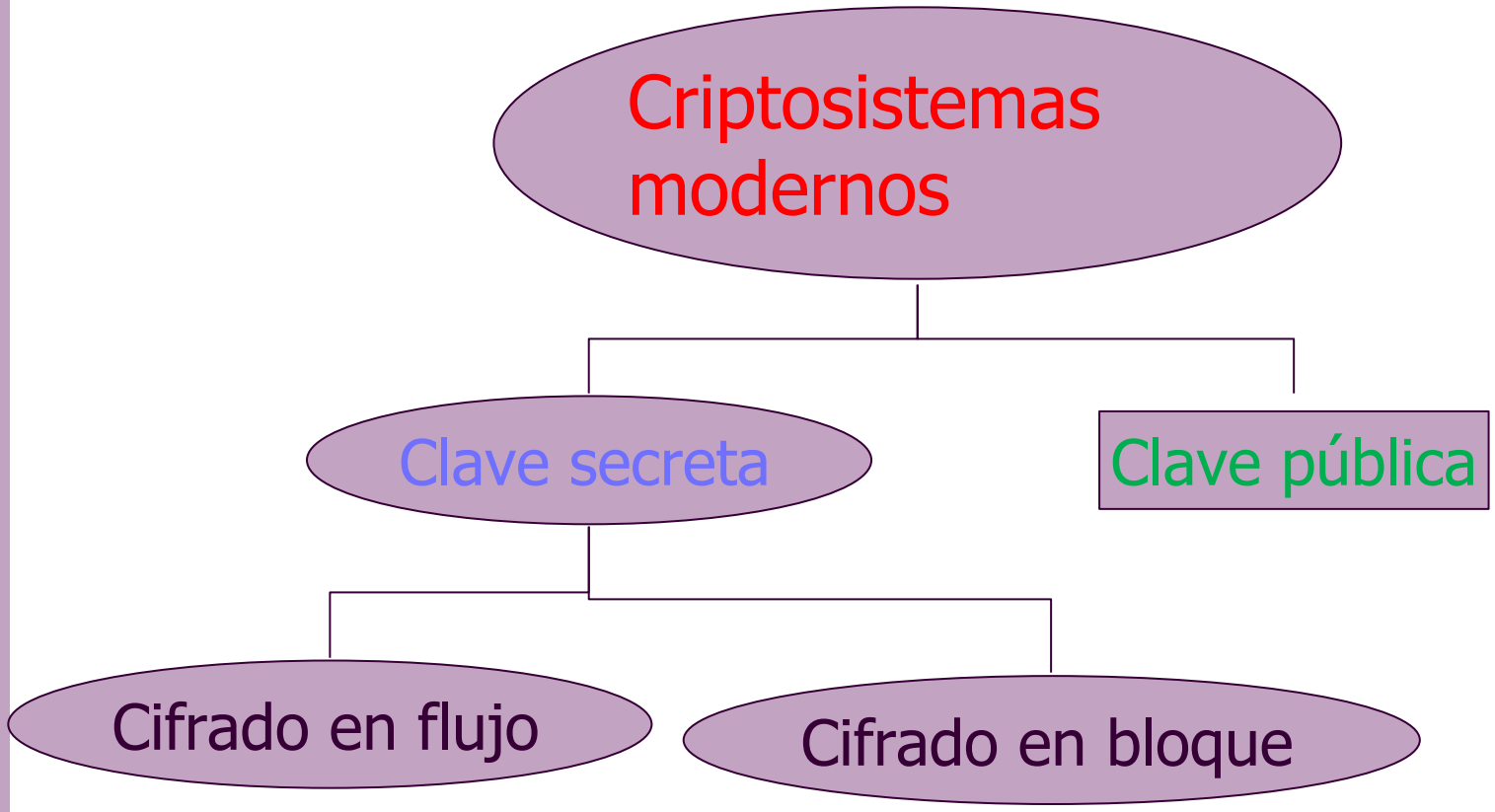



– **Criptosistemas Asimétricos o de Clave Pública:**

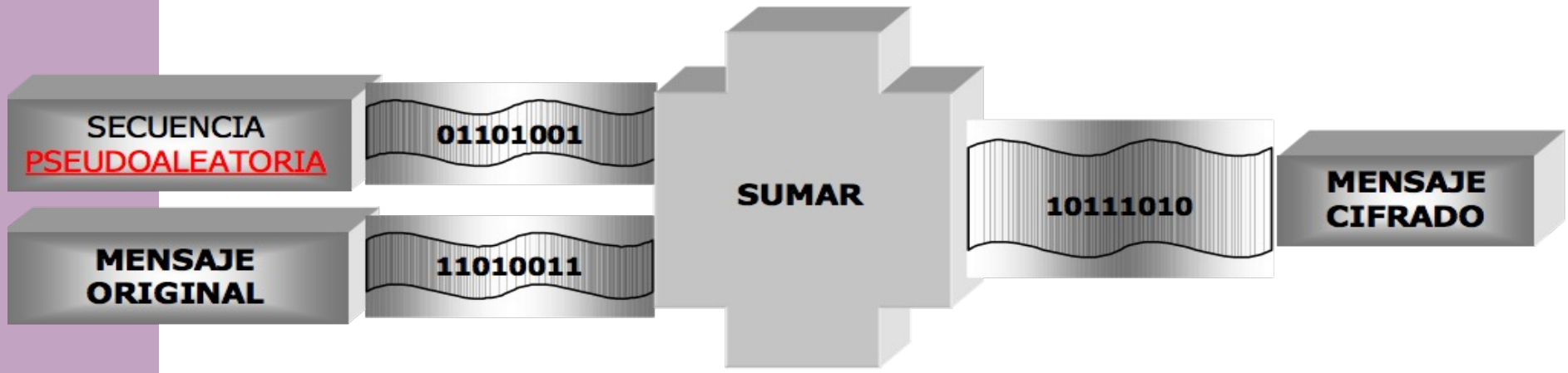


**Clave pública** Clave privada

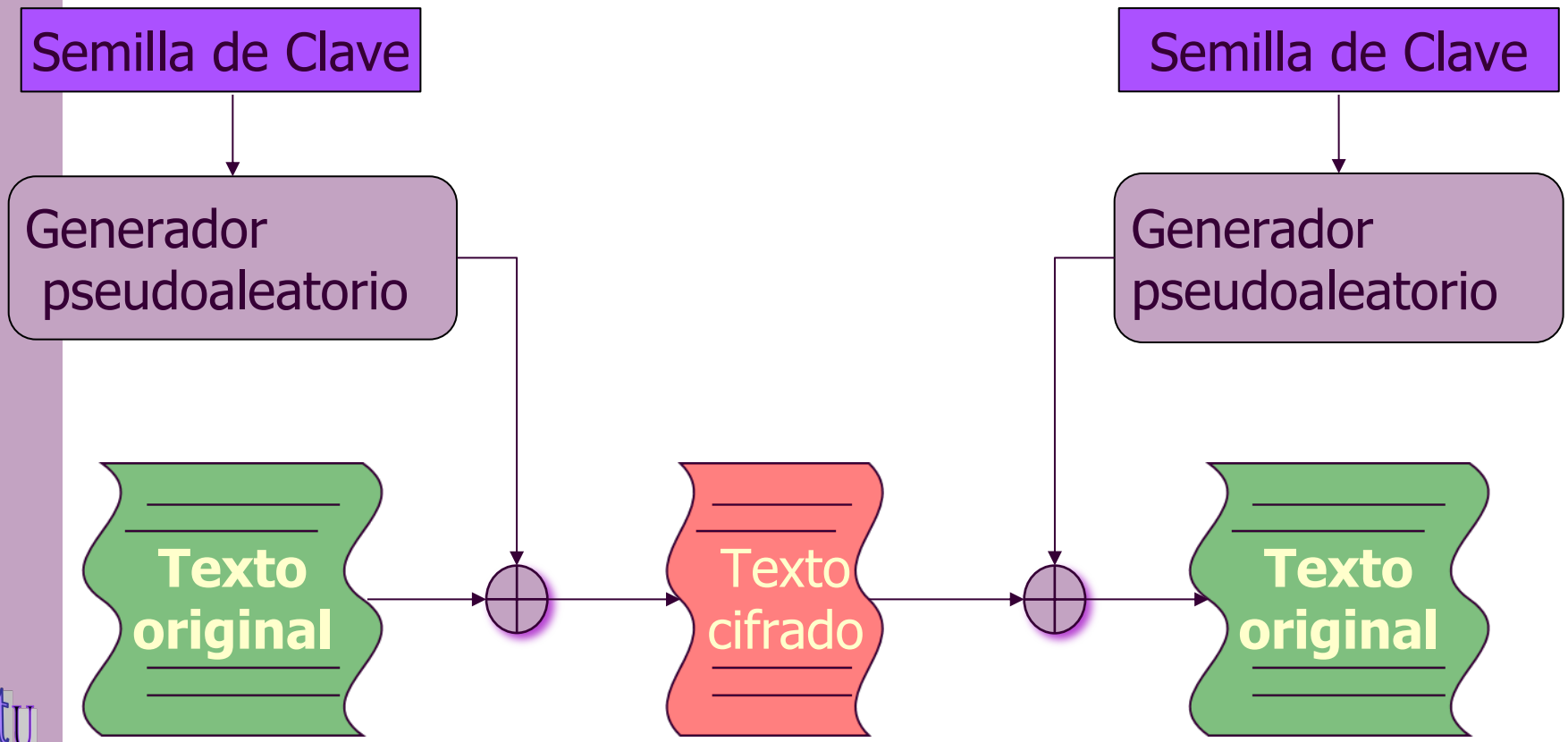
# Protección de la confidencialidad



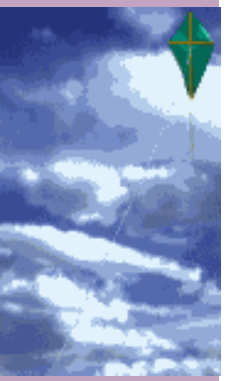
# Cifrado en Flujo



# Cifrado en Flujo

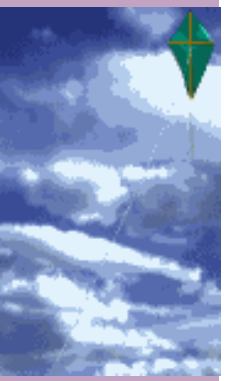


# Cifrado en Flujo



- El emisor usa una semilla secreta y un algoritmo determinista (generador pseudoaleatorio) para generar una secuencia binaria.
- Con la secuencia generada y el texto en claro expresado en binario se realiza una **XOR** bit a bit.
- Realizando la misma operación con el texto cifrado y la misma secuencia pseudoaleatoria se recupera el texto en claro.

# Generadores Pseudoaleatorios



1. ChaCha20

2. C/A

3. Snow3G

4. ZUC

5. Crypto-1

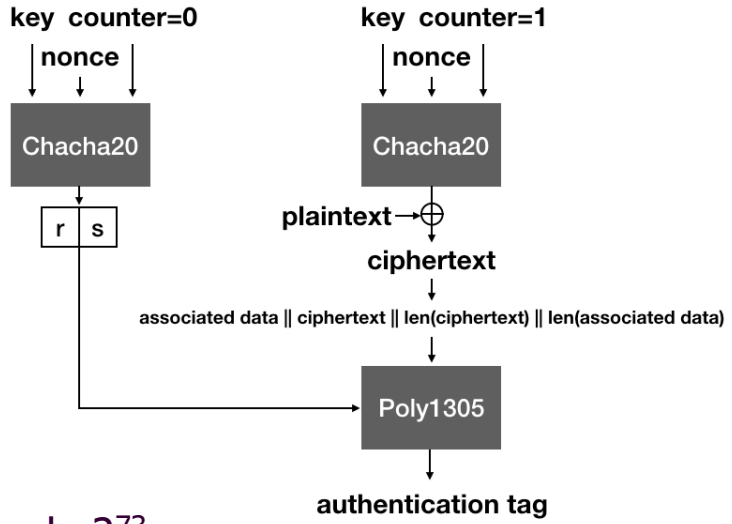
P. Caballero Gil


# Cifrado en SSL/TLS

| Cifrado | | | Versión del Protocolo | | | | | |
|---|---|---|---|---|---|---|---|---|
| Tipo | Algoritmo | Fortaleza nominal (bits) | SSL 2.0 | SSL 3.0 note 1 note 2 note 3 | TLS 1.0 note 1 note 3 | TLS 1.1 note 1 | TLS 1.2 note 1 | TLS 1.3[14] |
| Cifrado por bloques | AES GCM[15] note 4 | 256, 128 | N/D | N/D | N/D | N/D | Seguro | Seguro |
| | AES CCM[16] note 4 | | N/D | N/D | N/D | N/D | Seguro | Seguro |
| | AES CBC note 5 | | N/D | N/D | depende | depende | depende | N/D |
| | Camellia GCM[17] note 4 | 256, 128 | N/D | N/D | N/D | N/D | Seguro | N/D |
| | Camellia CBC[18] note 5 | | N/D | N/D | depende | depende | depende | N/D |
| | ARIA GCM | 256, 128 | N/D | N/D | N/D | N/D | Seguro | N/D |
| | ARIA CBC | | N/D | N/D | depende | depende | depende | N/D |
| | SEED CBC[19] note 5 | 128 | N/D | N/D | depende de las mitigaciones | depende de las mitigaciones | depende de las mitigaciones | N/D |
| | 3DES EDE CBC note 5 note 6 | 112 note 7 | Inseguro | Inseguro | Inseguro | Inseguro | Inseguro | N/D |
| | GOST 28147-89 CNT[13] note 6 | 256 | N/D | N/D | Inseguro | Inseguro | Inseguro | N/D |
| | IDEA CBC note 5 note 6 note 8 | 128 | Inseguro | Inseguro | Inseguro | Inseguro | N/D | N/D |
| | DES CBC note 5 note 6 note 8 | 56 | Inseguro | Inseguro | Inseguro | Inseguro | N/D | N/D |
| | | 40 note 9 | Inseguro | Inseguro | Inseguro | N/D | N/D | N/D |
| | RC2 CBC note 5 note 6 | 40 note 9 | Inseguro | Inseguro | Inseguro | N/D | N/D | N/D |
| Cifrador de flujo | ChaCha20+Poly1305[24] note 4 | 256 | N/D | N/D | N/D | N/D | Seguro | Seguro |
| | RC4 note 10 | 128 | Inseguro | Inseguro | Inseguro | Inseguro | Inseguro | N/D |
| | | 40 | Inseguro | Inseguro | Inseguro | N/D | N/D | N/D |

# Cifrado ChaCha20 de SSL/TLS

- Cifrado en flujo desarrollado en 2008 por Daniel **Bernstein**.
- Es un generador pseudoaleatorio de tipo **ARX**, e.d. con operaciones: **Add** de suma de 32 bits con acarreo (mod $2^{32}$), **Rotate** (rotaciones) y **XOR** o sumas binarias XOR.
- El generador se alimenta de:
  - Una **constante** de 128 bits
  - una clave secreta de **256 bits**,
  - un **nonce** pseudoaleatorio de 96 bits y
  - un **contador** de 32 bits




- Produce 512 bits de secuencia cifrante.
- La secuencia cifrante tiene un **periodo** máximo de $2^{73}$
- Se suele combinar con el código de autenticación de mensajes MAC, **Poly1305**.
- Es rápido, no está patentado y hay implementaciones de dominio público.
- Es la base de la función hash BLAKE, finalista en el concurso NIST de hash.

# Cifrado ChaCha20 de SSL/TLS


## Cipher Suites

**# TLS 1.3 (server has no preference)**

TLS_AES_128_GCM_SHA256 (0x1301) ECDH x25519 (eq. 3072 bits RSA) FS

TLS_AES_256_GCM_SHA384 (0x1302) ECDH x25519 (eq. 3072 bits RSA) FS

TLS_CHACHA20_POLY1305_SHA256 (0x1303) ECDH x25519 (eq. 3072 bits RSA) FS

**# TLS 1.2 (suites in server-preferred order)**

TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c) ECDH x25519 (eq. 3072 bits RSA) FS

TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca9) ECDH x25519 (eq. 3072 bits RSA) FS

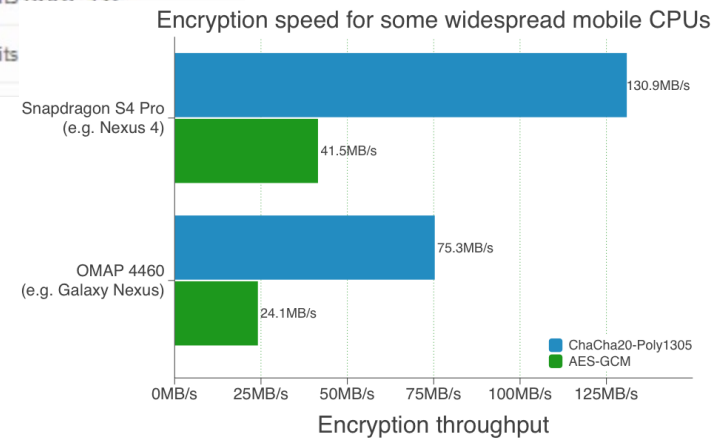
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b) ECDH x25519 (eq. 3072 bits RSA) FS

TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 (0xc024) ECDH x25519 (eq. 3072 bits RSA) FS

TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 (0xc023) ECDH x25519 (eq. 3072 bits RSA) FS

TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc00a) ECDH x25519 (eq. 3072 bits RSA) FS

TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (0xc009) ECDH x25519 (eq. 3072 bits




# Cifrado ChaCha20 de SSL/TLS

- Las operaciones se realizan sobre un **estado interno de 512 bits, e.d. 16 palabras de 32 bits dispuestas como matrices 4×4**:

| | | | |
|---|---|---|---|
| 0 | 1 | 2 | 3 |
| 4 | 5 | 6 | 7 |
| 8 | 9 | 10 | 11 |
| 12 | 13 | 14 | 15 |

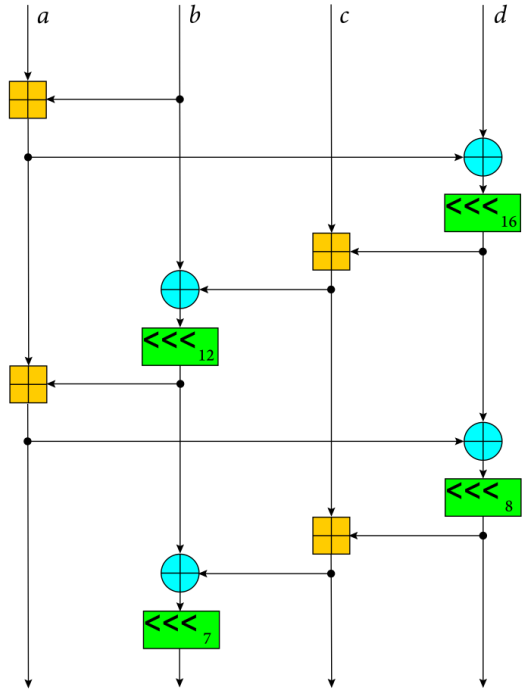
- El **estado inicial** se compone de: una clave de 256 bits, un contador de 32 bits, un nonce de 96 bits, y una constante de 128 bits, organizado como: 8 palabras de clave Key, 1 palabra contador Counter, 3 palabras de Nonce y 4 palabras constantes (expa, nd 3, 2-by y te k):

| "expa" | "nd 3" | "2-by" | "te k" |
|---|---|---|---|
| Key | Key | Key | Key |
| Key | Key | Key | Key |
| Counter | Nonce | Nonce | Nonce |

# Cifrado ChaCha20 de SSL/TLS

- La operación principal es Quarter-Round **QR(a,b,c,d)**, que toma una entrada de 4 palabras, y la actualiza como salida de 4 palabras de forma que:

| | | |
|---|---|---|
| a=a+b | → | d=(d ⊕ a)<<16 |
| c=c+d | → | b =(b ⊕ c)<<12 |
| a=a+b | → | d=(d ⊕ a)<<8 |
| c=c+d | → | b=(b ⊕ c)<<7 |




- Hay 20 iteraciones de QR(a,b,c,d), t.q. en las:
  - **impares** se aplica sobre las 4 **columnas**
  - **pares** se aplica sobre las 4 **diagonales**

# Implementaciones del Cifrado ChaCha20

**Algorithm 1** ChaCha20 Stream Cipher:
$C = \text{ChaCha20-SC}(K, N, P)$.

**Input:** $K \in \{0,1\}^{256}$, $N \in \{0,1\}^{96}$, $P \in \{0,1\}^*$
**Output:** $C \in \{0,1\}^{|P|}$

**for** $i \leftarrow 0$ **to** $\lceil |P|/512 \rceil - 1$ **do**
  /* Init State */
  $S[0] \leftarrow$ 0x61707865, $S[1] \leftarrow$ 0x3320646e
  $S[2] \leftarrow$ 0x79622d32, $S[3] \leftarrow$ 0x6b206574
  $S[4..11] \leftarrow K$ {Set Key}
  $S[12] \leftarrow i$ {Set Counter}
  $S[13..15] \leftarrow N$ {Set Nonce}
  $S' \leftarrow S$ {Save Initial State}
  **for** $n \leftarrow 0$ **to** 9 **do** {10 Double Rounds}
    /* Column Round */
    $S[0,4,8,12] \leftarrow \text{QR}(S[0], S[4], S[8], S[12])$
    $S[1,5,9,13] \leftarrow \text{QR}(S[1], S[5], S[9], S[13])$
    $S[2,6,10,14] \leftarrow \text{QR}(S[2], S[6], S[10], S[14])$
    $S[3,7,11,15] \leftarrow \text{QR}(S[3], S[7], S[11], S[15])$
    /* Diagonal Round */
    $S[0,5,10,15] \leftarrow \text{QR}(S[0], S[5], S[10], S[15])$
    $S[1,6,11,12] \leftarrow \text{QR}(S[1], S[6], S[11], S[12])$
    $S[2,7,8,13] \leftarrow \text{QR}(S[2], S[7], S[8], S[13])$
    $S[3,4,9,14] \leftarrow \text{QR}(S[3], S[4], S[9], S[14])$
  **end for**
  $k_i^N \leftarrow S \boxplus S'$ $\{\forall 0 \le x \le 15 : S[x] \boxplus S'[x]\}$
  $c_i \leftarrow p_i \oplus k_i^N$ {Encrypt}
**end for**
**return** $C$

```c
#define ROTL(a,b) (((a) << (b)) | ((a) >> (32 - (b))))
#define QR(a, b, c, d) (            \
    a += b,  d ^= a,  d = ROTL(d,16),   \
    c += d,  b ^= c,  b = ROTL(b,12),   \
    a += b,  d ^= a,  d = ROTL(d, 8),   \
    c += d,  b ^= c,  b = ROTL(b, 7))
#define ROUNDS 20

void chacha_block(uint32_t out[16], uint32_t const in[16])
{
    int i;
    uint32_t x[16];

    for (i = 0; i < 16; ++i)
        x[i] = in[i];
    // 10 loops × 2 rounds/loop = 20 rounds
    for (i = 0; i < ROUNDS; i += 2) {
        // Odd round
        QR(x[0], x[4], x[ 8], x[12]); // column 0
        QR(x[1], x[5], x[ 9], x[13]); // column 1
        QR(x[2], x[6], x[10], x[14]); // column 2
        QR(x[3], x[7], x[11], x[15]); // column 3
        // Even round
        QR(x[0], x[5], x[10], x[15]); // diagonal 1 (main diagonal)
        QR(x[1], x[6], x[11], x[12]); // diagonal 2
        QR(x[2], x[7], x[ 8], x[13]); // diagonal 3
        QR(x[3], x[4], x[ 9], x[14]); // diagonal 4
    }
    for (i = 0; i < 16; ++i)
        out[i] = x[i] + in[i];
}
```

https://datatracker.ietf.org/doc/html/rfc8439

# Entrada de Datos en el Cifrado ChaCha20

Key= 00:01:02:03: 04:05:06:07: 08:09:0a:0b: 0c:0d:0e:0f: 10:11:12:13: 14:15:16:17: 18:19:1a:1b: 1c:1d:1e:1f

Nonce= (00:00:00:09:00:00:00:4a:00:00:00:00)

Block Count = 1

Se usa el formato Little-endian (bytes en orden inverso):

ChaCha state with the key setup:

61707865 3320646e 79622d32 6b206574
03020100 07060504 0b0a0908 0f0e0d0c
13121110 17161514 1b1a1918 1f1e1d1c
00000001 09000000 4a000000 00000000

# Ejemplo de QR en Cifrado ChaCha20

$a = 11111111_{Hex}$
$b = 01020304_{Hex}$
$c = 9b8d6f43_{Hex}$
$d = 01234567_{Hex}$

$a = a + b = 11111111_{Hex} + 01020304_{Hex} = 12131415_{Hex}$;
$d = d \wedge a = 01234567_{Hex} \wedge 12131415_{Hex} = 13305172_{Hex}$= 0001 0011 0011 0000 0101 0001 0111 0010;
$d = d <<< 16$= 01010001011100100001001100110000 = $51721330_{Hex}$;

c = c + d= 9b8d6f43 + 51721330 = ECFF8273;
b= b ^ c= 01020304 ^ ECFF8273 = EDFD8177 = 1110 1101 1111 1101 1000 0001 0111 0111;
b= b <<< 12= 11011000000101110111111011011111 = D8177EDF;

a = a + b= 12131415 + D8177EDF = EA2A92F4;
d= d ^ a= 51721330 ^ EA2A92F4 = BB5881C4= 10111011010110001000000111000100;
d= d <<< 8= 01011000100000011100010010111011 = 5881C4BB;

c = c + d= ECFF8273 + 5881C4BB = 14581472E;
b= b ^ c= D8177EDF ^ 14581472E = 9D9639F1= 10011101100101100011100111110001;
b= b <<< 7= 11001011000111001111100011001110 = CB1CF8CE;

# Ejercicio de Cifrado ChaCha20

Dada una entrada del ChaCha20, con valores:

Clave = 00:00:00:00: 00:00:00:00: 00:00:00:00: 00:00:00:00: FF:FF:FF:0F: FF:FF:FF:F0: FF:FF:0F:FF: FF:FF:F0:FF

Contador = 01:00:00:00

Nonce aleatorio = 02:00:00:00: 03:00:00:00: 04:00:00:00

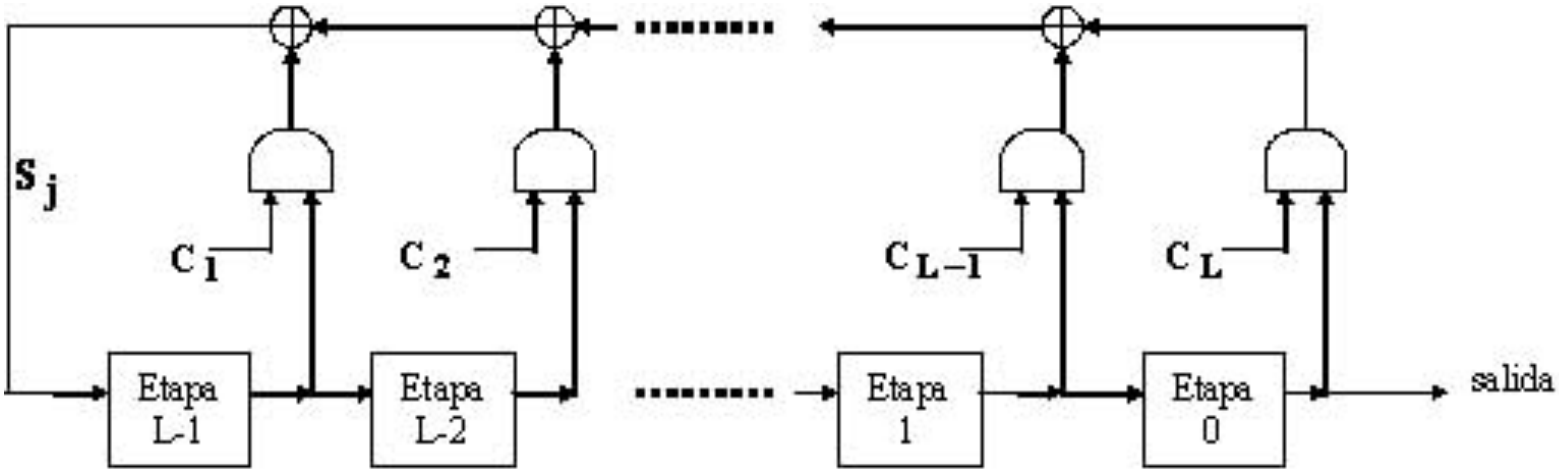
¿Cuál es la 1ª palabra de la salida de la 1ª iteración de QR aplicada sobre la 1ª columna?

Sabiendo que las palabras constantes son (expa, nd 3, 2-by y te k)= (61707865, 3320646e, 79622d32, 6b206574)

# Registro de Desplazamiento con Realimentación Lineal

•En GPS, Bluetooth, telefonía móvil y cifrados militares se usan cifrados en flujo basados en RDRLs o Linear Feedback Shift Registers (LFSRs)

•Generador de secuencia binaria pseudoaleatoria $s_j$.




•Cada etapa de la estructura se inicializa con un bit.

•A cada golpe de reloj, se realimenta con un bit producido mediante XORs definidos por la estructura.

# RDRL o LFSR

**Polinomio de realimentación:**

$C(x)=1+c_1x+c_2x^2+\ldots+c_Lx^L$ con $c_i$ binarios

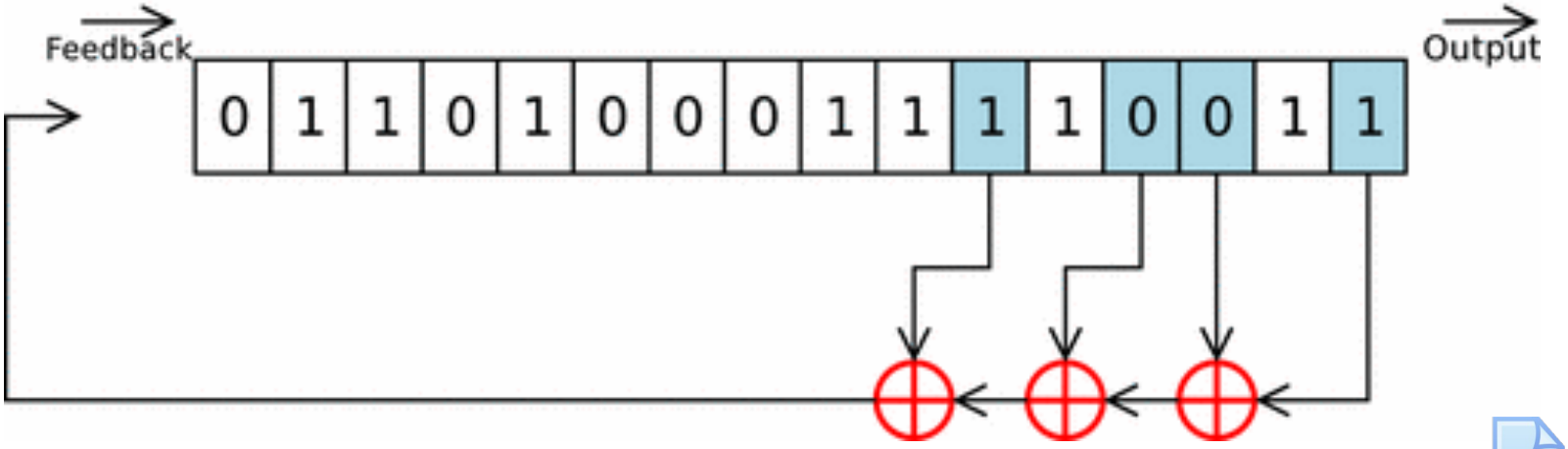
•El polinomio de realimentación

determina el periodo de la secuencia.

•El periodo máximo es $2^L-1$

| $m$ | Polinomios primitivos | $m$ | Polinomios primitivos |
|---|---|---|---|
| 1 | $x+1$ | 13 | $x^{13}+x^4+x^3+x+1$ |
| 2 | $x^2+x+1$ | 14 | $x^{14}+x^{10}+x^6+x+1$ |
| 3 | $x^3+x+1$ | 15 | $x^{15}+x+1$ |
| 4 | $x^4+x+1$ | 16 | $x^{16}+x^{12}+x^3+x+1$ |
| 5 | $x^5+x^2+1$ | 17 | $x^{17}+x^3+1$ |
| 6 | $x^6+x+1$ | 18 | $x^{18}+x^7+1$ |
| 7 | $x^7+x^3+1$ | 19 | $x^{19}+x^5+x^2+x+1$ |
| 8 | $x^8+x^4+x^3+x^2+1$ | 20 | $x^{20}+x^3+1$ |
| 9 | $x^9+x^4+1$ | 21 | $x^{21}+x^2+1$ |
| 10 | $x^{10}+x^3+1$ | 22 | $x^{22}+x+1$ |
| 11 | $x^{11}+x^2+1$ | 23 | $x^{23}+x^5+1$ |
| 12 | $x^{12}+x^6+x^4+x+1$ | 24 | $x^{24}+x^7+x^2+x+1$ |

(Polinomio primitivo: irreducible cuyas raíces son elementos primitivos, e.d. con potencias que generan todo el cuerpo de Galois)

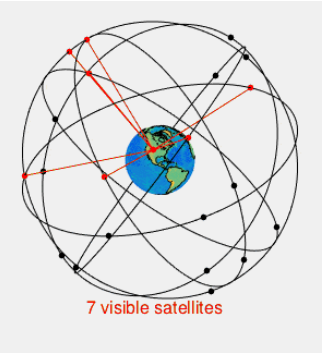



Ej: polinomio de realimentación: $1 + x^{11} + x^{13} + x^{14} + x^{16}$

P. Caballero Gil

# Comunicaciones en GNSS

- Un Sistema global de navegación por satélite (Global Navigation Satellite System, GNSS) es una constelación de 18-30 satélites que transmite señales para el posicionamiento y localización.
- Actualmente, 4 sistemas forman parte de GNSS, transmitiendo diferentes señales:
  - **GPS, de los Estados Unidos de América:**
    - **L1C/A, P(Y), L1C, L2C y L5**
  - **GLONASS, de la Federación Rusa:**
    - **L1OF, L2OF, L2OC, L1OC y L3OC**
  - **BDS o BeiDou, de China:**
    - **B1I, B1C, B2I, B2A y B3I**
  - **Galileo, de Europa:**
    - **E1 y E5a/b**
- El 87% de las señales se generan usando **LFSRs**.



# Comunicaciones en GPS

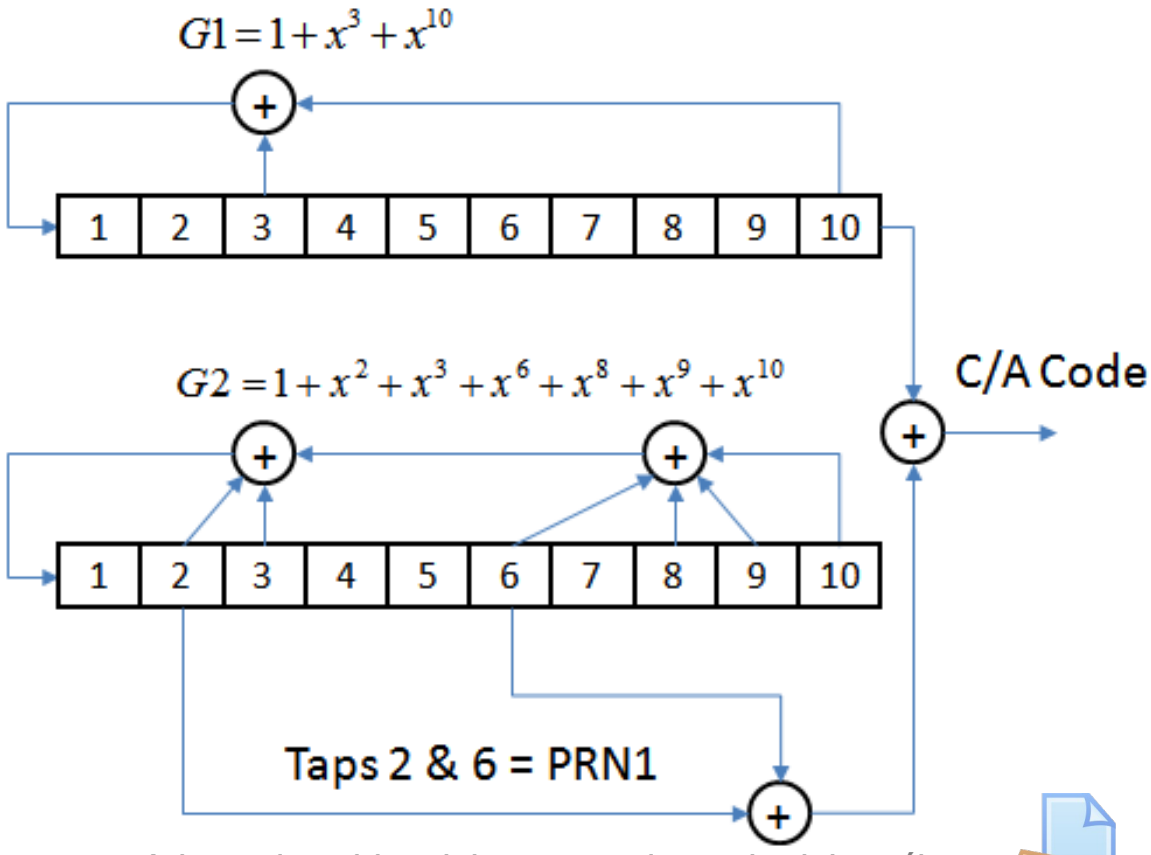

- **GPS L1C/A** (Código de Adquisición Aproximativa): Generador para uso civil. Utiliza 2 LFSRs de longitud 10 y estados iniciales 111…..1.
- **GPS P(Y)** (Cifrado de Precisión): Generador para uso militar. Genera una secuencia cifrante de periodo $2{,}35*10^{14}$ que se repite cada 266 días (con reset cada domingo), y en la que cada satélite usa un fragmento diferente. Utiliza 4 LFSRs de longitud 12.
- Hay otras 3 señales para uso civil, pero todavía no plenamente operativas: GPS L1C, GPS L2C y GPS L5, de las que GPS L2C y GPS L5 también usan LFSRs.

- No basadas en LFSRS son **GPS L1C, BDS B1C y Galileo E1.**
- Las secuencias generadas con GPS L1C/A y GPS P(Y) son **secuencias de Gold**, (como **BDS B1I, BDS B2I, BDS B2A, BDS B3I** y **Galileo E5a/b**).

# GPS C/A

- Genera una secuencia de periodo $2^{10}$-1=1023 que se repite cada milisegundo, y que es diferente para cada satélite.
- Genera una **secuencia Gold,** con baja correlación cruzada con otras generadas con el mismo generador, lo que es útil cuando varios dispositivos transmiten en el mismo rango de frecuencia.

| PRN ID | G2 Taps | PRN ID | G2 Taps |
|---|---|---|---|
| 1 | 2 & 6 | 17 | 1 & 4 |
| 2 | 3 & 7 | 18 | 2 & 5 |
| 3 | 4 & 8 | 19 | 3 & 6 |
| 4 | 5 & 9 | 20 | 4 & 7 |
| 5 | 1 & 9 | 21 | 5 & 8 |
| 6 | 2 & 10 | 22 | 6 & 9 |
| 7 | 1 & 8 | 23 | 1 & 3 |
| 8 | 2 & 9 | 24 | 4 & 6 |
| 9 | 3 & 10 | 25 | 5 & 7 |
| 10 | 2 & 3 | 26 | 6 & 8 |
| 11 | 3 & 4 | 27 | 7 & 9 |
| 12 | 5 & 6 | 28 | 8 & 10 |
| 13 | 6 & 7 | 29 | 1 & 6 |
| 14 | 7 & 8 | 30 | 2 & 7 |
| 15 | 8 & 9 | 31 | 3 & 8 |
| 16 | 9 & 10 | 32 | 4 & 9 |




(El par de celdas del 2º LFSR depende del satélite, como en BDS B1I y BDS B2I.)

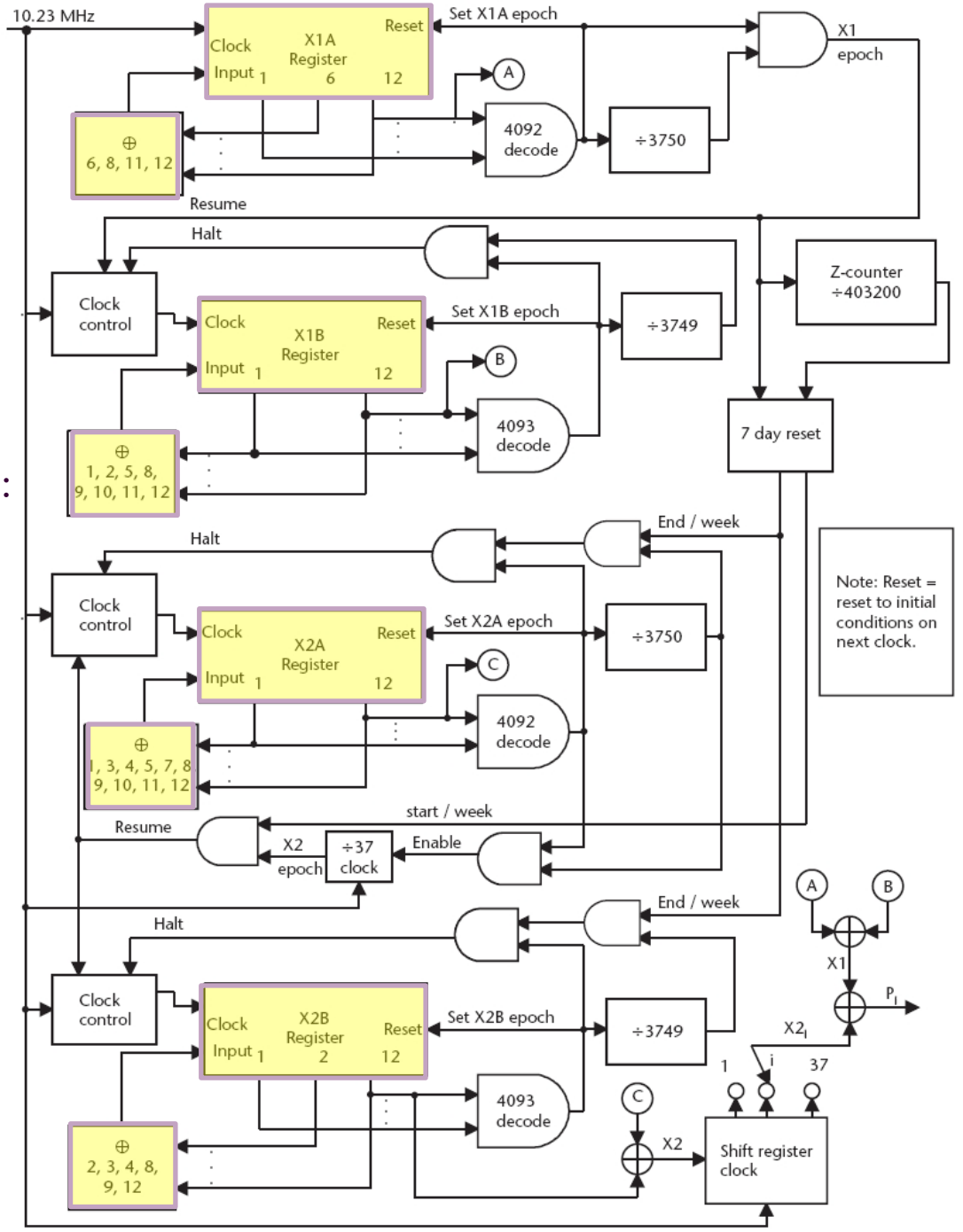
# GPS P(Y)

GPS

Estados iniciales de los RDRLs:

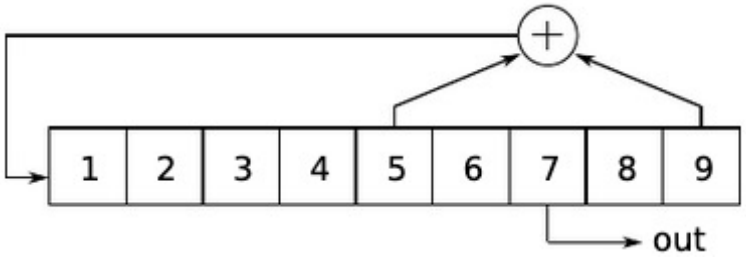
- De X1A: 001001001000
- De X1B: 010101010100
- De X2A: 100100100101
- De X2B: 010101010100

https://www.gps.gov/technical/icwg/IS-GPS-200K.pdf

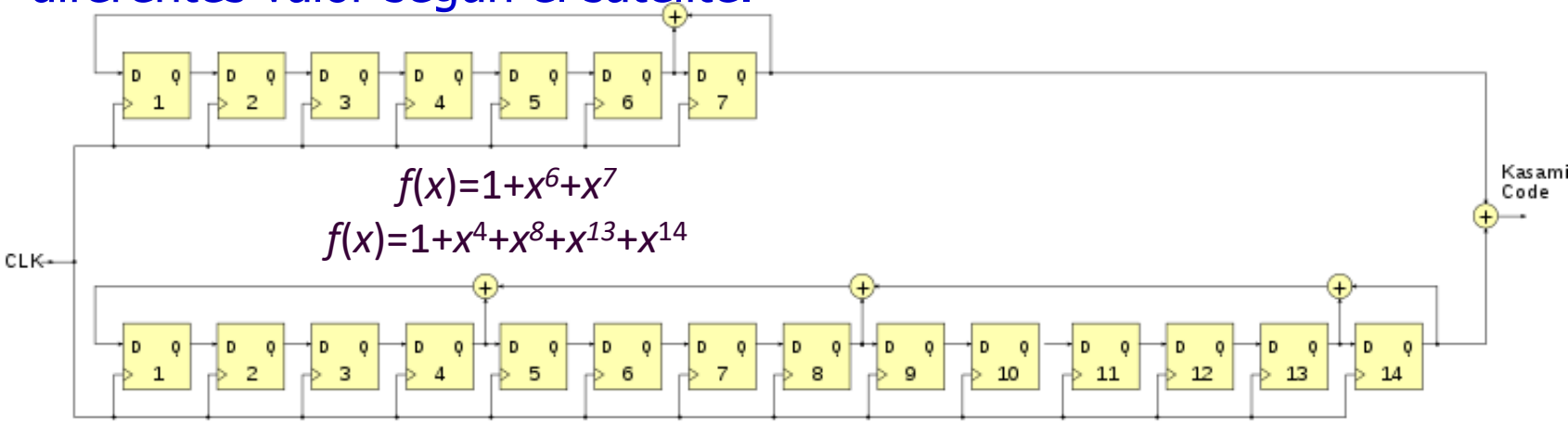


# Comunicaciones en GLONASS

- **GLONASS L1OF y GLONASS L2OF:** Generadores que utilizan 1 LFSR de longit




- **GLONASS L2OC:** Generador para uso civil. Genera una **secuencia Kasami** (como **GLONASS L1OC y GLONASS L3OC**). Utiliza 2 LFSRs de longitudes 7 y 14, y el estado inicial de SR2 toma diferentes valor según el satélite.



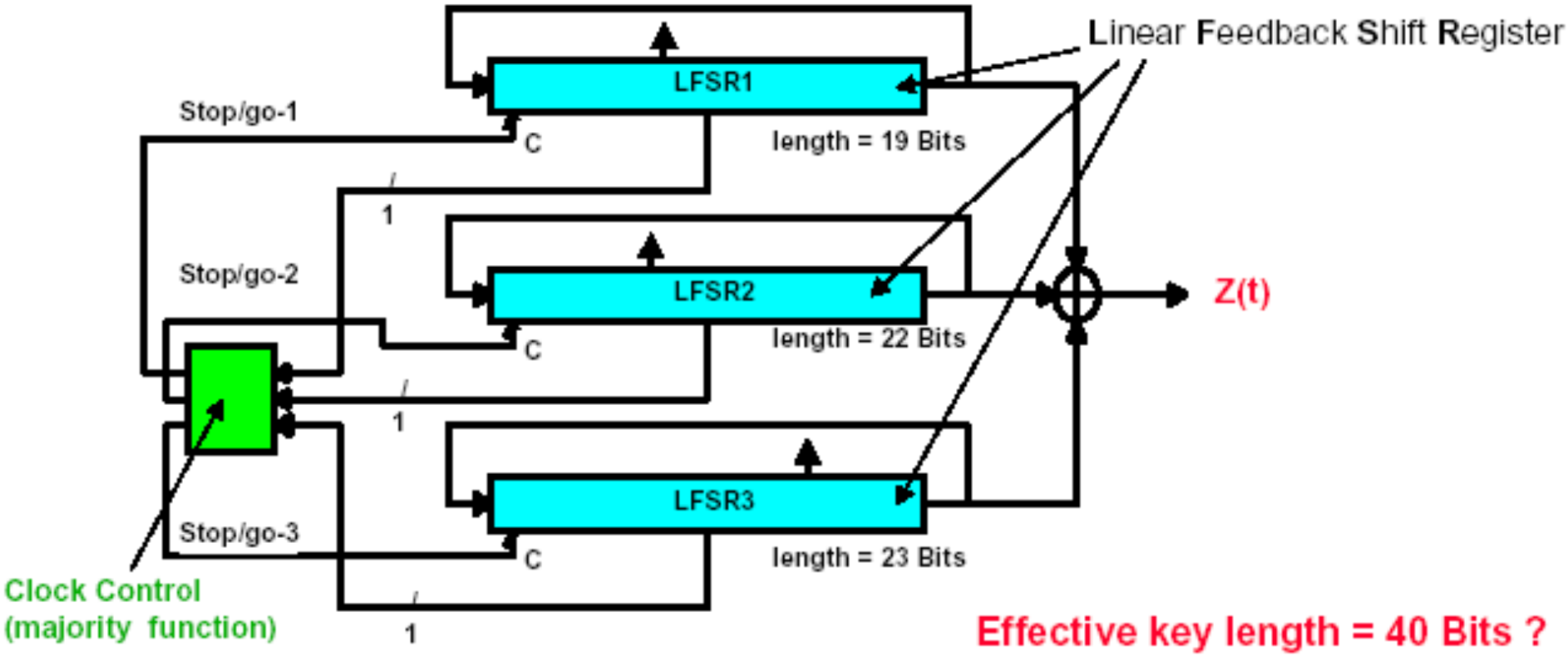
# Cifrado en Telefonía Móvil

- 2G o GSM (**A5**) desde 1999
- 3G o UMTS (**Kasumi y SNOW3G**) desde 2002
- 4G o LTE (**SNOW3G y AES**) desde 2010
- 5G (**SNOW3G, ZUC y AES**) desde 2020

movistar vodafone orange™ yoigo

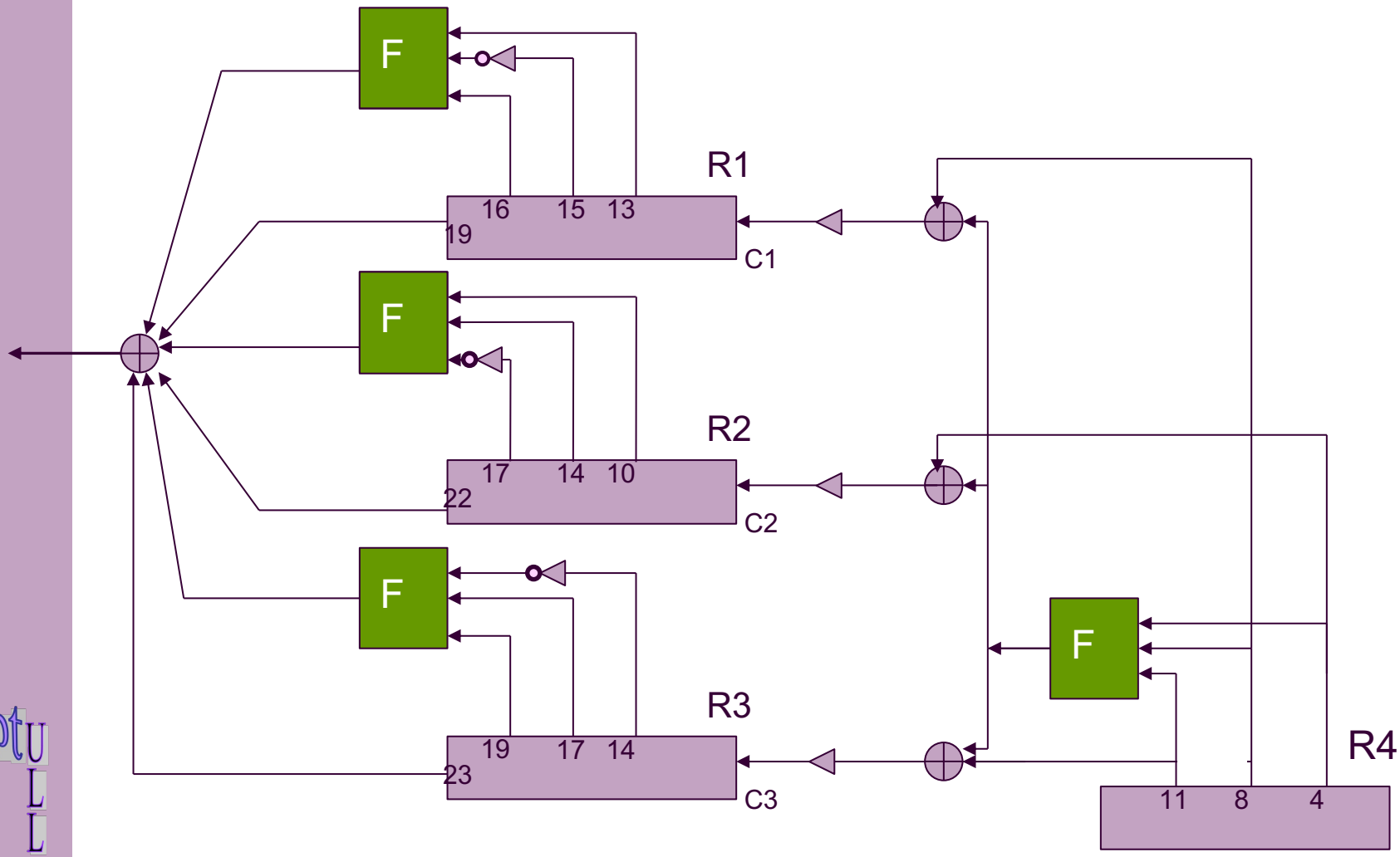
http://www.movistar.es/particulares/coberturas/movil/5G/

# Generador A5/1 de GSM




Effective key length = 40 Bits ?

# Generador A5/2 de GSM



# Cifrado en 3G (UMTS), 4G (LTE) y 5G

Para confidencialidad en UMTS:

- UEA1 = KASUMI
- UEA2 = **SNOW 3G**

Para confidencialidad en LTE:

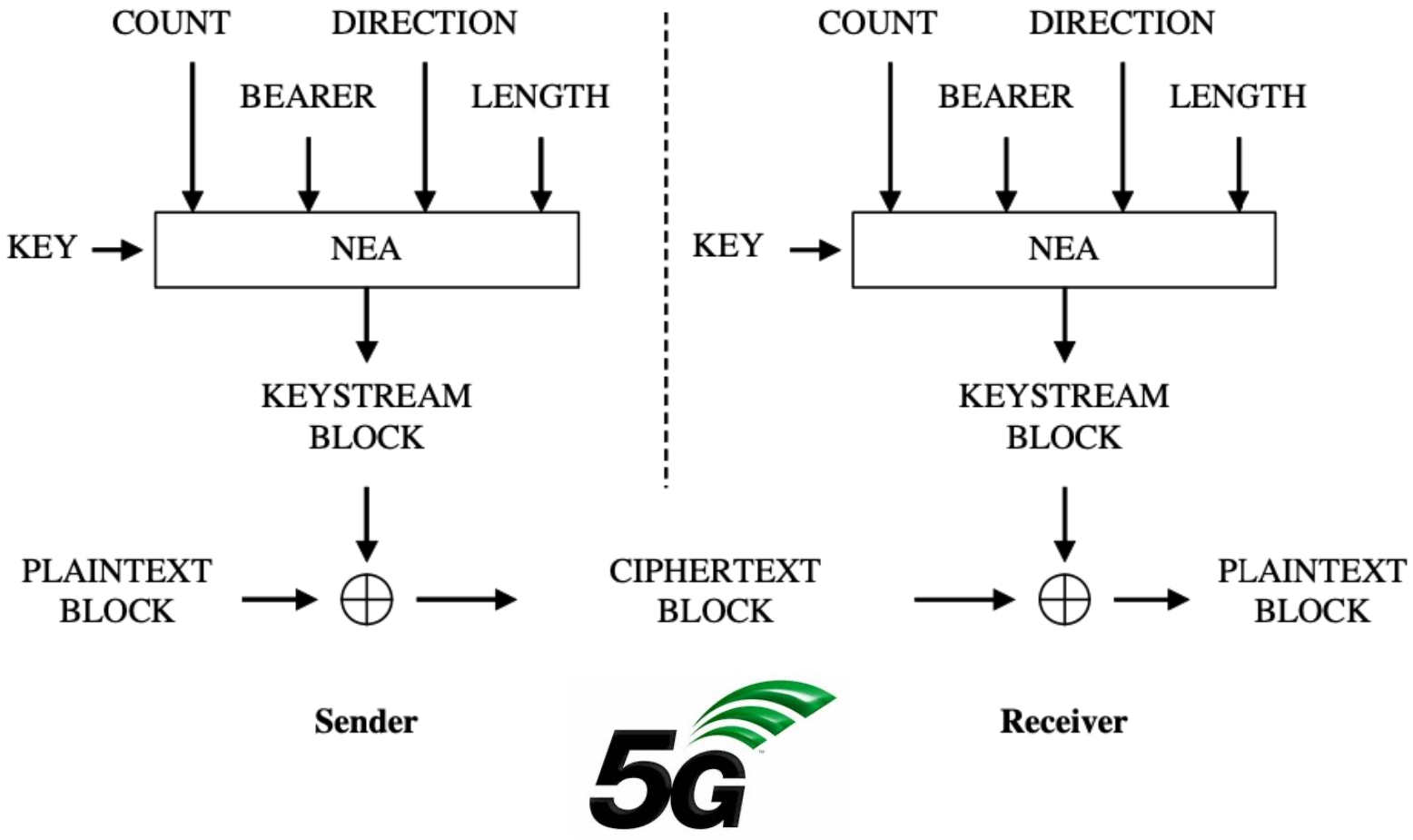
- 128-EEA1 = **SNOW 3G**
- 128-EEA2 = AES
- 128-EEA3 = **ZUC**

Para confidencialidad e integridad:

- NEA1 = NIA1 = **SNOW 3G**
- NEA2 = NIA2 = AES
- NEA3 = NIA3 = **ZUC**

# Cifrado en 5G

Entrada: Clave de 128 bits, Contador de 32 bits, Identidad del portador de 5 bits, Dirección de la transmisión de 1 bit, y Longitud de la secuencia cifrante necesaria.



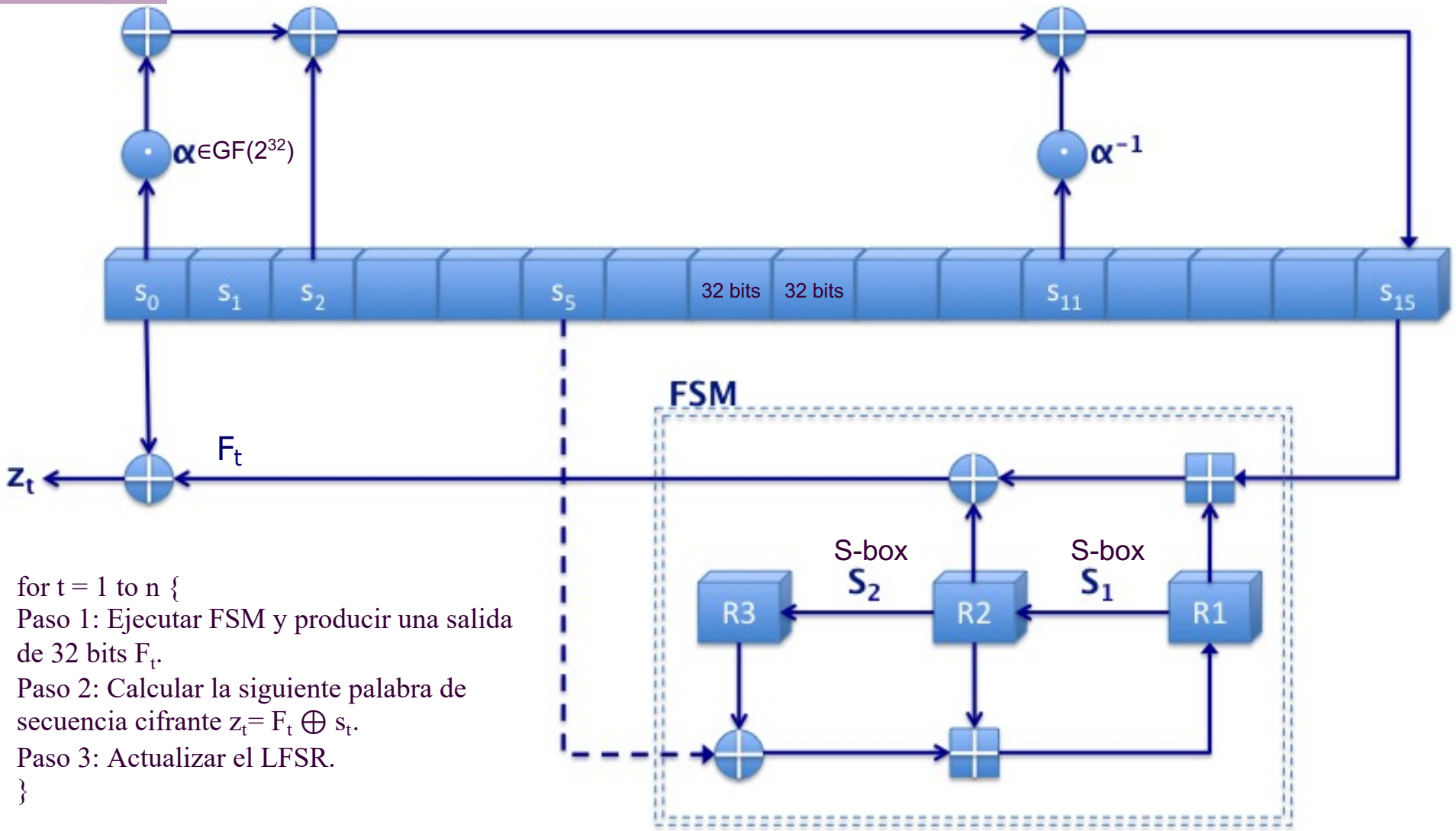

https://www.etsi.org/deliver/etsi_ts/133500_133599/133501/15.02.00_60/ts_133501v150200p.pdf

# Generador SNOW 3G de 3G, 4G y 5G

- SNOW 3G consta de dos partes:
  - un LFSR y
  - una máquina de estados finitos FSM.
- El LFSR tiene longitud 16 y **cada elemento $s_0$, $s_1$, $s_2$,..., $s_{15}$, contiene 4 bytes, e.d. 32 bits**.
- La realimentación se define con un polinomio primitivo sobre GF($2^{32}$), e implica dos multiplicaciones, una por una constante α de GF($2^{32}$) y otra por la inversa de α.

$$s_{t+16} = \alpha s_t \oplus s_{t+2} \oplus \alpha^{-1} s_{t+11}, \forall t \geq 0$$

# Generador SNOW 3G de 3G, 4G y 5G




for t = 1 to n {
Paso 1: Ejecutar FSM y producir una salida de 32 bits $F_t$.
Paso 2: Calcular la siguiente palabra de secuencia cifrante $z_t = F_t \oplus s_t$.
Paso 3: Actualizar el LFSR.
}

# SNOW 3G

- La **FSM** es la parte no lineal del generador y se alimenta de dos valores de entrada procedentes del LFSR correspondientes a los elementos en las posiciones 5 y 15.
- La FSM está formada por tres registros de 32 bits, R1, R2 y R3, y dos cajas de sustitución o S-boxes, S1 y S2, que se usan para actualizar los registros R2 y R3.
- En la FSM, las **S-Boxes** S1 y S2 toman 32 bits y producen 32 bits tras varias combinaciones de una S-box básica sobre cada uno de los 4 bytes de entrada. La caja S1 se basa en la S-box usada en el cifrado AES, mientras que la S-box S2 fue diseñada para SNOW 3G.
- En la FSM, las operaciones de mezcla de la FSM consisten en una **XOR bit a bit, y una suma de enteros módulo $2^{32}$**.

# SNOW 3G

- El LFSR puede usarse en dos modos de operación:
  - de inicialización (desechando la salida de 32 iteraciones)
  - de generación de secuencia cifrante.
- SNOW 3G se inicializa usando:
  - una clave de 128 bits como 4 palabras de 32 bits: $k_0$, $k_1$, $k_2$, $k_3$
  - un vector de inicialización IV de 128 bits como 4 palabras de 32 bits: $IV_0$, $IV_1$, $IV_2$, $IV_3$
  - Una palabra 1 de todo unos: 0xffffffff
  - R1 = R2 = R3 = 0

$$s_{15} = \mathbf{k_3} \oplus \mathbf{IV_0} \qquad s_{14} = \mathbf{k_2} \qquad s_{13} = \mathbf{k_1} \qquad s_{12} = \mathbf{k_0} \oplus \mathbf{IV_1}$$

$$s_{11} = \mathbf{k_3} \oplus \mathbf{1} \qquad s_{10} = \mathbf{k_2} \oplus \mathbf{1} \oplus \mathbf{IV_2} \qquad s_{9} = \mathbf{k_1} \oplus \mathbf{1} \oplus \mathbf{IV_3} \qquad s_{8} = \mathbf{k_0} \oplus \mathbf{1}$$

$$s_{7} = \mathbf{k_3} \qquad s_{6} = \mathbf{k_2} \qquad s_{5} = \mathbf{k_1} \qquad s_{4} = \mathbf{k_0}$$

$$s_{3} = \mathbf{k_3} \oplus \mathbf{1} \qquad s_{2} = \mathbf{k_2} \oplus \mathbf{1} \qquad s_{1} = \mathbf{k_1} \oplus \mathbf{1} \qquad s_{0} = \mathbf{k_0} \oplus \mathbf{1}$$

https://www.gsma.com/security/wp-content/uploads/2019/05/snow3gspec.pdf

# Implementación de SNOW 3G

- Las dos multiplicaciones implicadas en el LFSR se corresponden con productos de polinomios en módulo $x^8+x^7+x^5+x^3+1$, pero pueden implementarse como desplazamientos de bytes, y XORs con el byte $A9_{16}=10101001_2$.
- Implican aplicar operación distributiva sobre los dos bytes multiplicandos, usando para ello el byte de menor peso, y luego para cada bit 1 de ese byte, desplazar a izquierda el otro byte, de forma que cada vez que su bit más significativo antes del desplazamiento sea 1, hay que hacer, tras el desplazamiento, una XOR bit a bit con el byte $A9_{16}=10101001_2$.
- El nº de desplazamientos a realizar viene dado por la posición de los bits iguales a 1 en el 1er byte multiplicando.

# Multiplicación en SNOW 3G

Los bytes se representan en forma de polinomios.

Ej: $57_{(16}$= 0101 0111$_{(2}$= $x^6+x^4+x^2+x+1$ $83_{(16}$= 1000 0011$_{(2}$= $x^7+x+1$

Multiplicación en GF($2^8$ )

Consiste en multiplicar los polinomios (en módulo 2), y el resultado reducirlo módulo el polinomio **$x^8+x^7+x^5+x^3+1$** (que se corresponde con el byte **10**1 ).

Ej: {57} . {83}

$$(5\ 7)_{16} \otimes (8\ 3)_{16} = (C\ 1)_{16} \rightarrow$$
$$(x^6 + x^4 + x^2 + x\ +1) \otimes (x^7 + x\ +1) =$$
$$x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 \equiv$$
$$(x^7 + x^4 + x^2 + x) \bmod (x^8 + x^7 + x^5 + x^3 + 1)$$

**Ej: 01010111x10000011=**01010111x00000001+01010111x00000010+01010111x10000000=

01010111+10101110+01010111x10000000=01010111+10101110+01101111=**10010110=96**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 01010111 | 10101110 | 01011100+<br>10101001=<br>11110101 | 11101010+<br>10101001=<br>01000011 | 10000110 | 00001100+<br>10101001=<br>10100101 | 01001010+<br>10101001=<br>11100011 | 11000110+<br>10101001=<br>01101111 |

# Multiplicaciones en SNOW 3G

If el bit más más significativo de la entrada V es 1, then

$$MULx(V, c) = (V \ll 1) \oplus c,$$

else

$$MULx(V, c) = V \ll 1.$$

---

If $i = 0$, then

$$MULxPOW(V, i, c) = V,$$

else

$$MULxPOW(V, i, c) = MULx(MULxPOW(V, i - 1, c), c).$$

# Multiplicaciones en SNOW 3G

$MUL_\alpha(c) =$

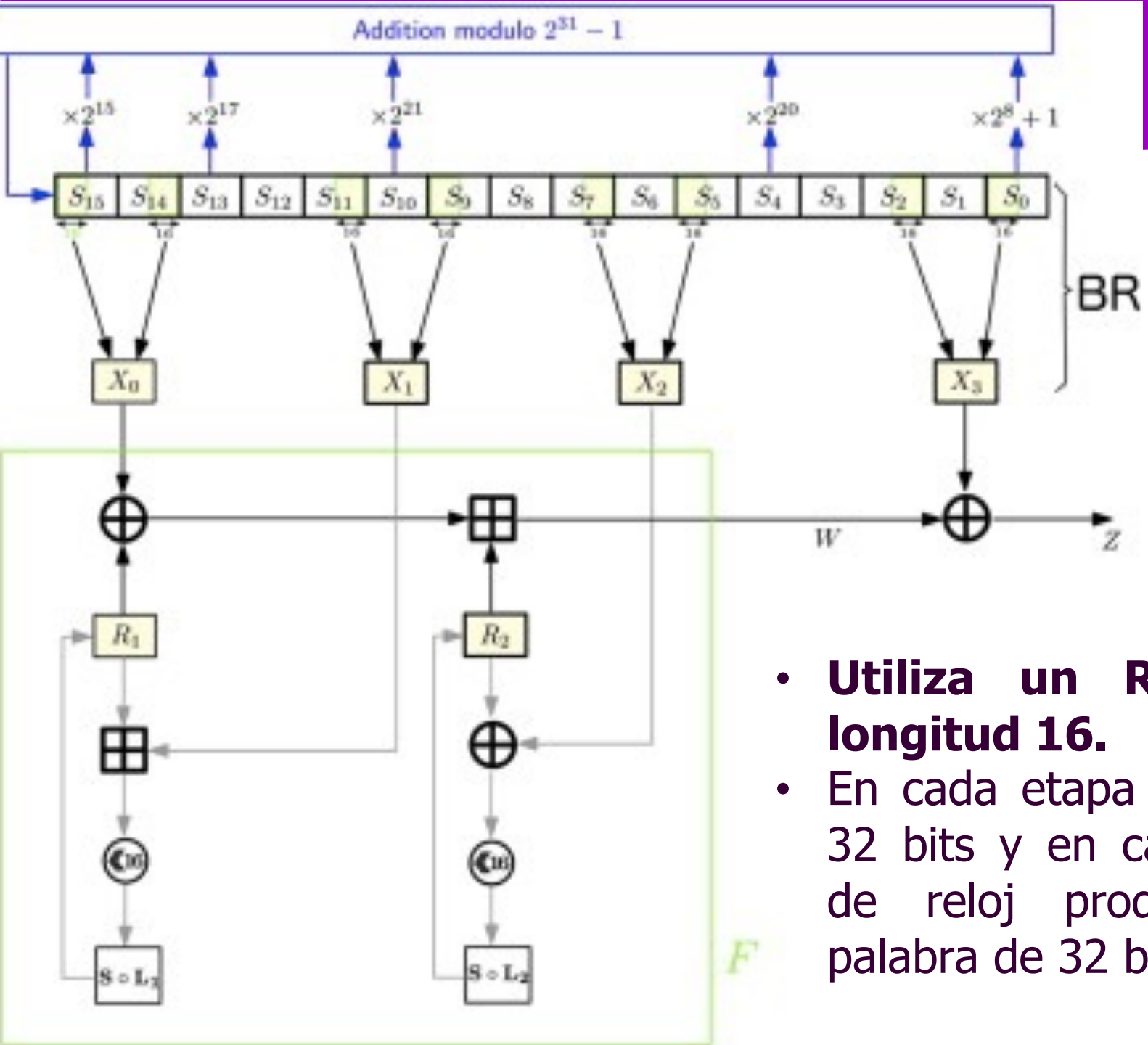
(MULxPOW(c, 23, 0xA9) || MULxPOW(c, 245, 0xA9) || MULxPOW(c, 48, 0xA9) || MULxPOW(c, 239, 0xA9)).

---

$DIV_\alpha(c) =$

(MULxPOW(c, 16, 0xA9) || MULxPOW(c, 39, 0xA9) || MULxPOW(c, 6, 0xA9) || MULxPOW(c, 64, 0xA9)).

# ZUC



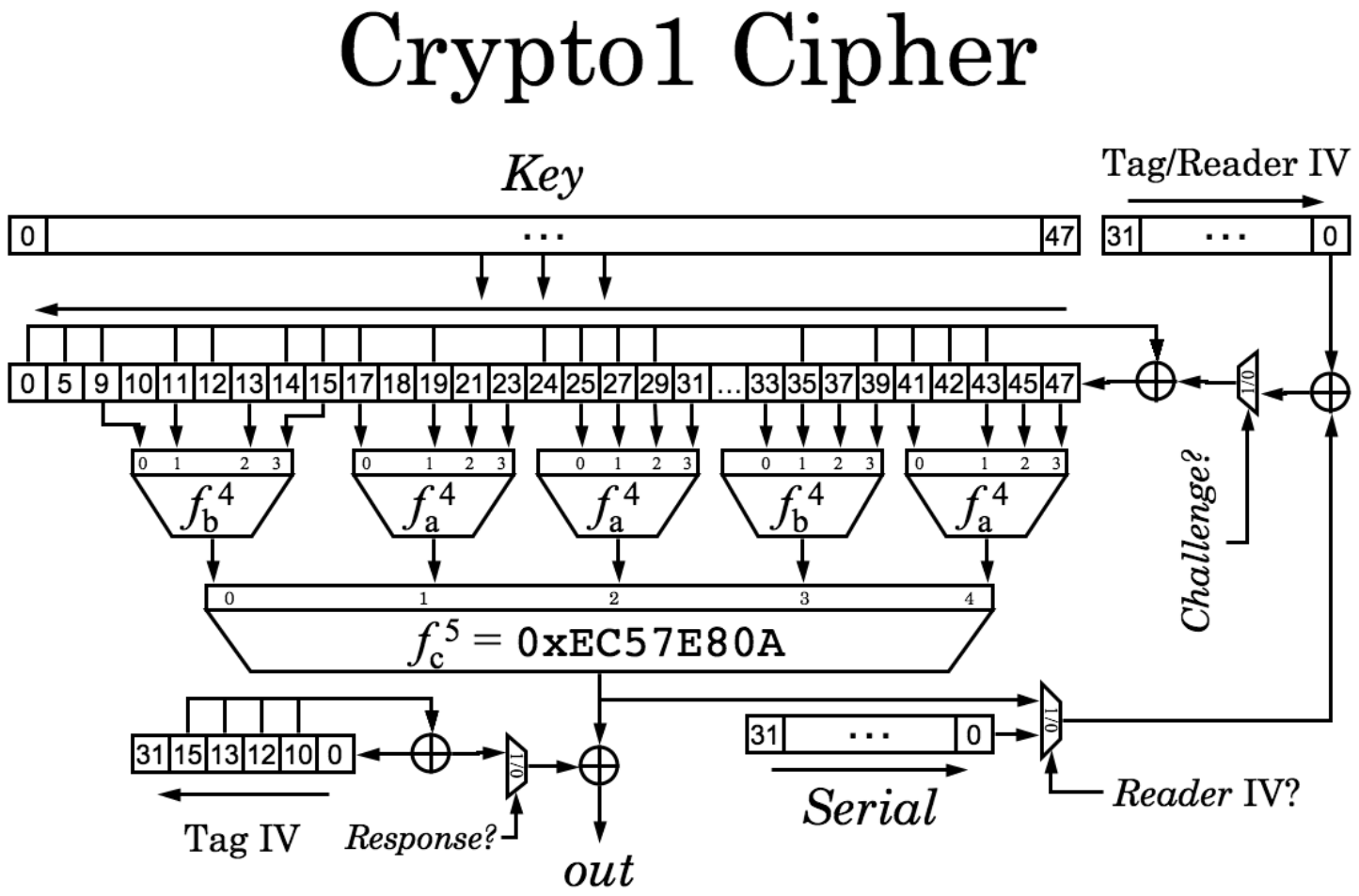

- **Utiliza un RDRL de longitud 16.**
- En cada etapa almacena 32 bits y en cada golpe de reloj produce una palabra de 32 bits

# Inseguridad de 4G vs 5G



- En 2018 usando una herramienta llamada **LTEInspector** se atacó en 4G:
  - Conexión: cuando se asocia el dispositivo de un usuario con la red.
  - Desconexión: cuando se apaga el dispositivo y se desconecta de la red.
  - Búsqueda: cuando se realiza una llamada y se busca al dispositivo en la red.
- El ataque más peligroso a 4G es de **suplantación o spoofing** porque un atacante puede conectarse sin tener credenciales legítimas.
- Otro ataque a 4G permite **bloquear** a un dispositivo el acceso a la red, **rastrear** a un usuario por las antenas que usa, o enviar alertas falsas.
- **5G se empezó a implantar en 2020.**
- La mayor diferencia entre 5G y 4G es la velocidad, pero tb en seguridad:
  - soporta claves de 256 bits para versiones futuras,
  - comprueba integridad detectando cambios no autorizados de los datos del usuario,
  - hay cambios en la autenticación,
  - protege la confidencialidad de los mensajes iniciales entre dispositivo y red,
  - permite reautenticación al moverse entre diferentes redes,
  - se puede usar la clave pública de la red para cifrar parte del identificador de abonado,
  - permite verificar la presencia, lo que resulta útil para roaming y prevención de fraudes,
  - se puede utilizar autenticación mutua basada en certificados,
  - protege el perímetro de la red doméstica en las interconexiones con otras redes.

# Generador Crypto-1




$f_a^4$ = 0x9E98 = (a+b)(c+1)(a+d)+(b+1)c+a

$f_b^4$ = 0xB48E = (a+c)(a+b+d)+(a+b)cd+b

Tag IV ⊕ Serial is loaded first, then Reader IV ⊕ NFSR