PRÁCTICA: GENERADOR CHACHA20

Objetivo: Implementar el Generador ChaCha20 usado en SSL/TLS.

Desarrollo:

Implementa el generador ChaCha20 según la descripción incluida en las transparencias pero siguiendo el RFC7539 disponible en https://datatracker.ietf.org/doc/html/rfc8439.

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] ← 0x61707865, S[1] ← 0x3320646e
  S[2] ← 0x79622d32, S[3] ← 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];
}
```

Ejemplo:

Entrada:

- Clave de 256 bits en forma de 8 palabras en hexadecimal=
  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
- Contador de 32 bits en forma de 1 palabra en hexadecimal = 01:00:00:00
- Nonce aleatorio de 96 bits en forma de 3 palabras en hexadecimal = 00:00:00:09: 00:00:00:4a: 00:00:00:00

Salida:

- Estado inicial=
  61707865 3320646e 79622d32 6b206574
  03020100 07060504 0b0a0908 0f0e0d0c
  13121110 17161514 1b1a1918 1f1e1d1c
  00000001 09000000 4a000000 00000000
- Estado final tras las 20 iteraciones=
  837778ab e238d763 a67ae21e 5950bb2f
  c4f2d0c7 fc62bb2f 8fa018fc 3f5ec7b7
  335271c2 f29489f3 eabda8fc 82e46ebd
  d19c12b4 b04e16de 9e83d0cb 4e3c50a2
- Estado de salida del generador=
  e4e7f110 15593bd1 1fdd0f50 c47120a3
  c7f4d1c7 0368c033 9aaa2204 4e6cd4c3
  466482d2 09aa9f07 05d7c214 a2028bd9
  d19c12b5 b94e16de e883d0cb 4e3c50a2