

SNOW-V

Estudo e Implementação em C e Python



Jhonatan Cléto

j256444@dac.unicamp.br

*Universidade Estadual de Campinas (Unicamp)
Instituto de Computação (IC)*

Projeto Final - MC889 • 13 de Julho de 2021

AGENDA

01. BACKGROUND

Contexto e objetivos do SNOW-V

02. DESIGN

Estrutura geral do cifrador e principais componentes

03. IMPLEMENTAÇÃO

Detalhes da Implementação em C e Python

04. RESULTADOS

Comparação de desempenho com outros cifradores

1.

BACKGROUND

Contexto

- 5G (2018)
- Mudanças na arquitetura do sistema
 - Virtualização dos nós da rede
 - Download a 20 Gbps (desejado)
- Mudanças no nível de segurança
 - Chaves com 256 bits
- Padrões atuais não se adequam às novas demandas

Objetivos

- ❑ Satisfazer a demanda da indústria
- ❑ Alta velocidade de encriptação em ambientes virtualizados
- ❑ Atualizar a arquitetura SNOW
- ❑ Nível de segurança equivalente ou superior ao AES-256

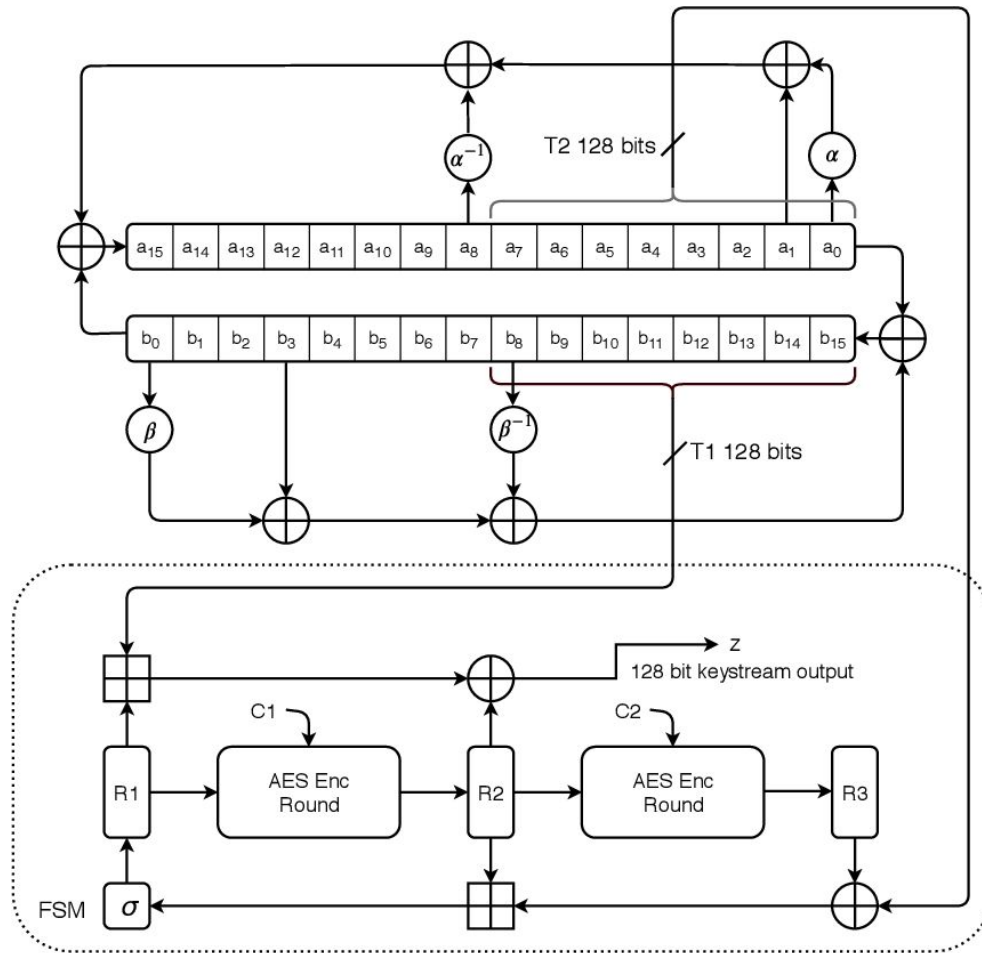
2. DESIGN

Arquitetura SNOW

- Cifrador de Fluxo (Stream Cipher)
- Duas estruturas principais
 - Linear Feedback Shift Register (LFSR)
 - Finite State Machine (FSM)
- Versões anteriores
 - SNOW 1.0 (NESSIE - 2000)
 - SNOW 2.0 (Correção de vulnerabilidades)
 - SNOW 3G (SAGE - Padronizado pelo 3GPP)

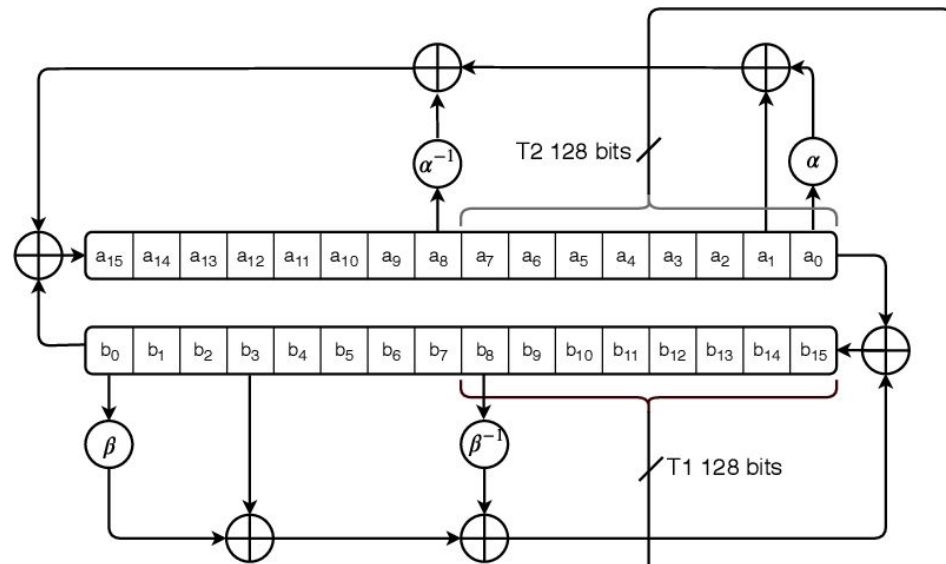
SNOW-V

- ❑ Atualização do design do SNOW 3G
- ❑ Estados com 896 bits
- ❑ Chave de 256 bits
- ❑ Planejado para utilizar instruções vetoriais
- ❑ Utiliza a função de rodada completa do AES



LFSR

- 2 LFSR (16x16 bits)
- 2 polinômios de geração
- Construção circular
 - Periodicidade: $2^{512}-1$
- Atualização de estado
 - 8 clocks nos LFSRS
 - T1 e T2 atualizados
- Implementação Eficiente em Software
 - Instruções vetoriais



$$g^A(x) = x^{16} + x^{15} + x^{12} + x^{11} + x^8 + x^3 + x^2 + x + 1 \in \mathbb{F}_2[x]$$

$$g^B(x) = x^{16} + x^{15} + x^{14} + x^{11} + x^8 + x^6 + x^5 + x + 1 \in \mathbb{F}_2[x]$$

$$a^{(t+16)} = b^{(t)} + \alpha a^{(t)} + a^{(t+1)} + \alpha^{-1} a^{(t+8)} \mod g^A(\alpha)$$

$$b^{(t+16)} = a^{(t)} + \beta b^{(t)} + b^{(t+3)} + \beta^{-1} b^{(t+8)} \mod g^B(\beta)$$

FSM

- Saída dada por:

$$z^{(t)} = (R1^{(t)} \boxplus_{32} T1^{(t)}) \oplus R2^{(t)}$$

- Atualização dos registradores:

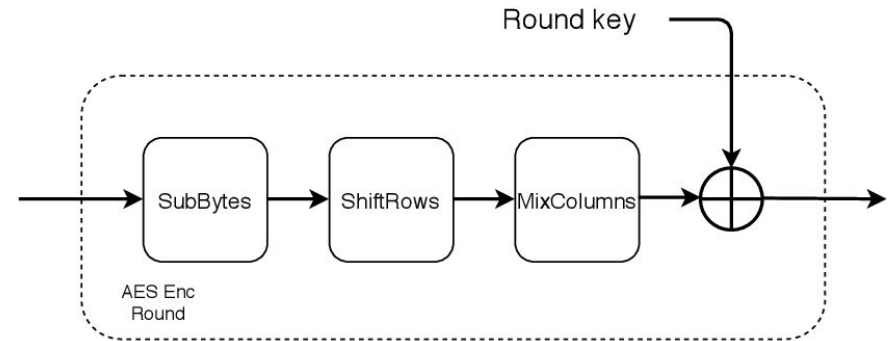
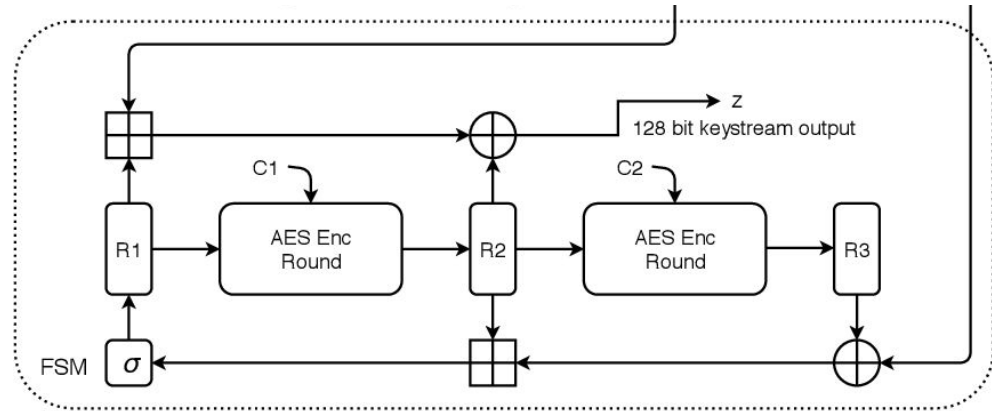
$$R1^{(t+1)} = \sigma(R2^{(t)} \boxplus_{32} (R3^{(t)} \oplus T2^{(t)})),$$

$$R2^{(t+1)} = AES^R(R1^{(t)}, C1),$$

$$R3^{(t+1)} = AES^R(R2^{(t)}, C2).$$

- σ é a permutação de bits:

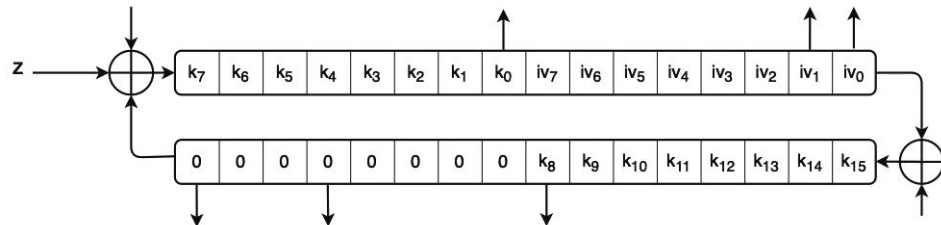
$$\sigma = [0, 4, 8, 12, 1, 5, 9, 13, 2, 6, 10, 14, 3, 7, 11, 15]$$



Inicialização

Algorithm 1 SNOW-V initialization

```
1: procedure INITIALIZATION( $K, IV$ )
2:    $(a_{15}, a_{14}, \dots, a_8) \leftarrow (k_7, k_6, \dots, k_0)$ 
3:    $(a_7, a_6, \dots, a_0) \leftarrow (iv_7, iv_6, \dots, iv_0)$ 
4:    $(b_{15}, b_{14}, \dots, b_8) \leftarrow (k_{15}, k_{14}, \dots, k_8)$ 
5:    $(b_7, b_6, \dots, b_0) \leftarrow (0, 0, \dots, 0)$ 
6:    $R1, R2, R3 \leftarrow 0, 0, 0$ 
7:   for  $t = 1 \dots 16$  do
8:      $T1 \leftarrow (b_{15}, b_{14}, \dots, b_8)$ 
9:      $z \leftarrow (R1 \boxplus_{32} T1) \oplus R2$ 
10:     $FSMupdate()$ 
11:     $LFSRupdate()$ 
12:     $(a_{15}, a_{14}, \dots, a_8) \leftarrow (a_{15}, a_{14}, \dots, a_8) \oplus z$ 
13:    if  $t = 15$  then  $R1 \leftarrow R1 \oplus (k_7, k_6, \dots, k_0)$ 
14:    if  $t = 16$  then  $R1 \leftarrow R1 \oplus (k_{15}, k_{14}, \dots, k_8)$ 
```



Inicialização

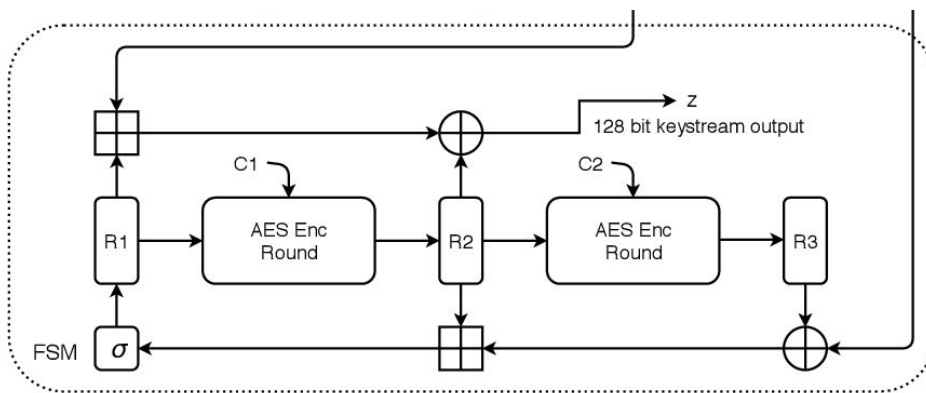
Algorithm 1 SNOW-V initialization

```

1: procedure INITIALIZATION( $K, IV$ )
2:    $(a_{15}, a_{14}, \dots, a_8) \leftarrow (k_7, k_6, \dots, k_0)$ 
3:    $(a_7, a_6, \dots, a_0) \leftarrow (iv_7, iv_6, \dots, iv_0)$ 
4:    $(b_{15}, b_{14}, \dots, b_8) \leftarrow (k_{15}, k_{14}, \dots, k_8)$ 
5:    $(b_7, b_6, \dots, b_0) \leftarrow (0, 0, \dots, 0)$ 
6:    $R1, R2, R3 \leftarrow 0, 0, 0$ 
7:   for  $t = 1 \dots 16$  do
8:      $T1 \leftarrow (b_{15}, b_{14}, \dots, b_8)$ 
9:      $z \leftarrow (R1 \boxplus_{32} T1) \oplus R2$ 
10:     $FSMupdate()$ 
11:     $LFSRupdate()$ 
12:     $(a_{15}, a_{14}, \dots, a_8) \leftarrow (a_{15}, a_{14}, \dots, a_8) \oplus z$ 
13:    if  $t = 15$  then  $R1 \leftarrow R1 \oplus (k_7, k_6, \dots, k_0)$ 
14:    if  $t = 16$  then  $R1 \leftarrow R1 \oplus (k_{15}, k_{14}, \dots, k_8)$ 

```

The diagram illustrates the FSM component. It shows a 3x3 grid of cells, likely representing a state or key schedule, with an arrow pointing to a register labeled R1. Below R1 is a function block labeled sigma, with an arrow pointing from R1 to it. The entire component is enclosed in a dashed box labeled FSM.



Algoritmo

Algorithm 2 SNOW-V algorithm

```
1: procedure SNOW-V( $K, IV$ )
2:   INITIALIZATION( $K, IV$ )
3:   while more keystream blocks needed do
4:      $T1 \leftarrow (b_{15}, b_{14}, \dots, b_8)$ 
5:      $z \leftarrow (R1 \boxplus_{32} T1) \oplus R2$ 
6:      $FSMupdate()$ 
7:      $LFSRupdate()$ 
8:     Output keystream symbol  $z$ 
```

Algorithm 3 LFSR update algorithm

```
1: procedure  $LFSRupdate()$ 
2:   for  $i = 0 \dots 7$  do
3:      $tmp_a \leftarrow b_0 + \alpha a_0 + a_1 + \alpha^{-1} a_8 \bmod g^A(\alpha)$ 
4:      $tmp_b \leftarrow a_0 + \beta b_0 + b_3 + \beta^{-1} b_8 \bmod g^B(\beta)$ 
5:      $(a_{15}, a_{14}, \dots, a_0) \leftarrow (tmp_a, a_{15}, \dots, a_1)$ 
6:      $(b_{15}, b_{14}, \dots, b_0) \leftarrow (tmp_b, b_{15}, \dots, b_1)$ 
```

Algorithm 4 FSM update algorithm

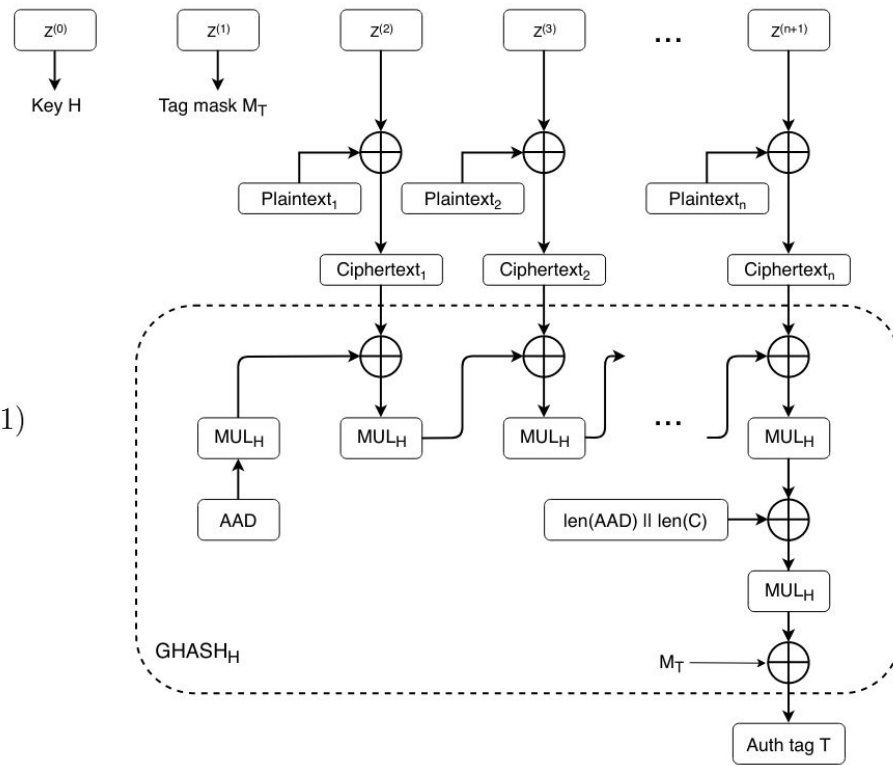
```
1: procedure  $FSMupdate()$ 
2:    $T2 \leftarrow (a_7, a_6, \dots, a_0)$ 
3:    $tmp \leftarrow R2 \boxplus_{32} (R3 \oplus T2)$ 
4:    $R3 \leftarrow AES^R(R2)$ 
5:    $R2 \leftarrow AES^R(R1)$ 
6:    $R1 \leftarrow \sigma(tmp)$ 
```

Modo AEAD

- Adaptação do GCM
 - Novo H para cada Key e IV
- Inicialização do LFSR B é diferente
 - Utiliza outra constante

$(b_7, b_6, \dots, b_0) = (6D6F, 6854, 676E, 694A, 2064, 6B45, 7865, 6C41)$

Keystream output z:



3. IMPLEMENTAÇÃO

Aspectos Gerais

- Implementação em C
 - Baseada na implementação de referência
 - Palavras de 8, 16 e 32 bits
 - Key - 32x8 bits
 - IV, Z - 16x8 bits
 - LFSRs - 16x16 bits
 - R_1, R_2, R_3 - 4x32 bits

Aspectos Gerais

- Implementação em Python
 - Orientada a objetos
 - Wrapper API
 - Chamadas a implementação em C
 - Parâmetros utilizando a classe bytes

Código C

```
// Saída de uma iteração do SNOW-V
// Produz um fluxo com 128 bits
void keystream(u8 *z)
{
    for (int i = 0; i < 4; i++)
    {
        u32 T1 = MAKEU32(B[2 * i + 9], B[2 * i + 8]);
        u32 v = (T1 + R1[i]) ^ R2[i];
        z[i * 4 + 0] = (v >> 0) & 0xff;
        z[i * 4 + 1] = (v >> 8) & 0xff;
        z[i * 4 + 2] = (v >> 16) & 0xff;
        z[i * 4 + 3] = (v >> 24) & 0xff;
    }
    fsm_update();
    lfsr_update();
}
```

Código C

```
void snowv_initialize(u8 *key, u8 *iv, int is_aead_mode)
{
    for (int i = 0; i < 8; i++)
    {
        A[i] = MAKEU16(iv[2 * i + 1], iv[2 * i]);
        A[i + 8] = MAKEU16(key[2 * i + 1], key[2 * i]);
        B[i] = 0x0000;
        B[i + 8] = MAKEU16(key[2 * i + 17], key[2 * i + 16]);
    }
    ...
    for (int i = 0; i < 16; i++)
    {
        u8 z[16];
        keystream(z);
        ...
        if (i == 15)
            for (int j = 0; j < 4; j++)
                R1[j] ^= MAKEU32(MAKEU16(key[4 * j + 19],
                                         key[4 * j + 18]),
                                MAKEU16(key[4 * j + 17], key[4 * j + 16]));
    }
}
```

Código C

```
// Modo AEAD
void snowv_gcm_encrypt(u8 *A, u8 *ciphertext, u8 *plaintext, \
    u64 plaintext_sz, u8 *aad, u64 aad_sz, u8 *key32, u8 *iv16)
{
    u8 Hkey[16], endPad[16];
    memset(A, 0, 16);
    snowv_initialize(key32, iv16, 1);
    keystream(Hkey);
    keystream(endPad);
    ghash_update(Hkey, A, aad, aad_sz);

    for (u64 i = 0; i < plaintext_sz; i += 16)
    {
        u8 key_stream[16];
        keystream(key_stream);
        For (u8 j = 0; j < min(16, plaintext_sz - i); j++)
            ciphertext[i + j] = key_stream[j] ^ plaintext[i + j];
    }

    ghash_update(Hkey, A, ciphertext, plaintext_sz);
    ghash_final(Hkey, A, aad_sz, plaintext_sz, endPad);
}
```

Código Python

```
// Classe python que abstrai as operações do SNOW-V
class SNOWV(object):
    ...
    def _initializer(self, key, iv):
        try:
            snowv_initializer(key, iv)
        except ValueError as e:
            raise CMException(e)
    ...
    def gcm_encrypt(self, key, iv, plaintext, add):
        if type(plaintext) != bytes:
            plaintext = bytes(plaintext.encode('ascii'))

        if type(add) != bytes:
            add = bytes(add.encode('ascii'))

        try:
            return snowv_gcm_encrypt(key, iv, plaintext, add)
        except ValueError as e:
            raise CMException(e)
    ...
```

Exemplo de uso

```
In [1]: from Cyphers.SNOWV import SNOWV

In [2]: key = bytes.fromhex(32*'ff')

In [3]: iv = bytes.fromhex(16*'ff')

In [4]: aad = 'associado'

In [5]: mensagem = 'Esta Mensagem sera criptografada utilizando o SNOW-V'

In [6]: snowv = SNOWV()

In [7]: enc, mac = snowv.gcm_encrypt(key, iv, mensagem, aad)

In [8]: enc
Out[8]: b'\xd0\xb89@W\xf6\x7fC\x93$J\xd6\xed\xc6\x18\xd2j:\xd8\xe7\xa9m\x96\x0f\x2\xc3q0:_\x9bDt\x9c*\xd1\x80\tE\xbe\x1bb.0\x02i\xf04\xe2\x9e\xa3\xaf'

In [9]: mac
Out[9]: b'\xfdG\xf9)\xae\xa43\x12d.(\xfai(@\xb3'

In [10]: dec = snowv.gcm_decrypt(key, iv, enc, aad, mac)

In [11]: dec
Out[11]: b'Esta Mensagem sera criptografada utilizando o SNOW-V'
```

AVX, SSE e AES-NI

```
inline __m128i keystream(void)
{
    // Extract the tags T1 and T2
    __m128i T1 = _mm256_extracti128_si256(hi, 1);
    __m128i T2 = _mm256_castsi256_si128(lo);

    // LFSR Update
    __m256i mulx = _mm256_xor_si256(_mm256_slli_epi16(lo, 1),
                                     _mm256_and_si256(_snowv_mul, _mm256_srai_epi16(lo, 15)));
    __m256i invx = _mm256_xor_si256(_mm256_srli_epi16(hi, 1),
                                     _mm256_sign_epi16(_snowv_inv, _mm256_slli_epi16(hi, 15)));
    __m256i hi_old = hi;
    hi = _mm256_xor_si256(
        _mm256_xor_si256(
            _mm256_blend_epi32(
                _mm256_alignr_epi8(hi, lo, 1 * 2),
                _mm256_alignr_epi8(hi, lo, 3 * 2), 0xf0),
            _mm256_permute4x64_epi64(lo, 0x4e)),
        _mm256_xor_si256(invx, mulx));
    lo = hi_old;

    // Keystream word
    __m128i z = _mm_xor_si128(R2, _mm_add_epi32(R1, T1));

    // FSM Update
    __m128i R3new = _mm_aesenc_si128(R2, _snowv_zero);
    __m128i R2new = _mm_aesenc_si128(R1, _snowv_zero);
    R1 = _mm_shuffle_epi8(_mm_add_epi32(R2, _mm_xor_si128(R3, T2)), _snowv_sigma);
    R3 = R3new;
    R2 = R2new;
    return z;
}
```


4. RESULTADOS

SNOW-V vetorizado

- Comparativo com AES, ChaCha20 e SNOW 3G
- Configuração do sistema
 - Intel i7-8650U @1.90GHz ~ @4.2GHz (TB)
 - OpenSSL 3.0.0-dev
 - Visual Studio 2017

SNOW-V vetorizado


Table 3: Performance comparison of SNOW-V-(GCM) and best OpenSSL's algorithms. Performance values are given in Gbps.





Encryption only	Size of input plaintext (bytes)						
	16384	8192	4096	2048	1024	256	64
SNOW-3G-128 (C++)	9.22	9.07	8.89	8.50	7.81	5.38	2.37
AES-256-CBC (asm)	8.50	8.50	8.49	8.48	8.42	8.11	7.07
ChaCha20 (asm)	26.53	26.41	26.29	25.86	24.99	11.80	5.61
AES-256-CTR (asm)	35.06	34.82	34.16	32.94	30.95	22.67	11.32
SNOW-V (C++)	58.25	56.98	54.60	50.70	45.28	26.37	9.85
AEAD mode							
ChaCha20-Poly1305 (asm)	18.46	18.24	18.16	17.54	16.99	8.98	4.29
AES-256-GCM (asm)	34.42	33.86	32.74	30.49	27.22	17.32	8.54
SNOW-V-GCM (C++)	38.91	37.66	34.86	30.71	26.16	13.93	5.16









SNOW-V 32 bits






- Comparativo com outros cifradores para 5G
- Configuração do sistema
 - Single Thread
 - Intel Xeon Silver 4108 @1.80GHz ~ @3.0GHz (TB)
 - Python 3.9.3
 - GCC 8.3.0


Fonte dos Algoritmos









 mitshell / **CryptoMobile** Public


 Notifications  Fork 55  Star 72 

 **Code**  Issues 1  Pull requests  Actions  Projects  Wiki  Security  Insights

 master  1 branch  0 tags  Go to file  Code

 mitshell Merge pull request #15 from P1sec/master ... 5bcd996 on Feb 18, 2021 55 commits

 C_alg	C_alg: add header comment with references to algorithms specification...	3 years ago
 C_py	pykeccakp1600: more precise argument error	17 months ago
 CryptoMobile	conv: transfer all conversion functions to conv.py	17 months ago
 _ctypes	new version of the lib, with CPython based wrapper to C files	5 years ago
 test	README: add explanation on ECIES usage	3 years ago
 .gitignore	remove temp files	4 years ago
 README.md	Merge pull request #13 from P1sec/master	2 years ago
 setup.py	extend README and setup to support TUAk	4 years ago


 README.md


CryptoMobile toolkit


Update 2019


About

Cryptography for mobile network - C implementation and Python bindings

 Readme

 72 stars

 11 watching

 55 forks






Releases

No releases published

Packages

No packages published

Contributors 5

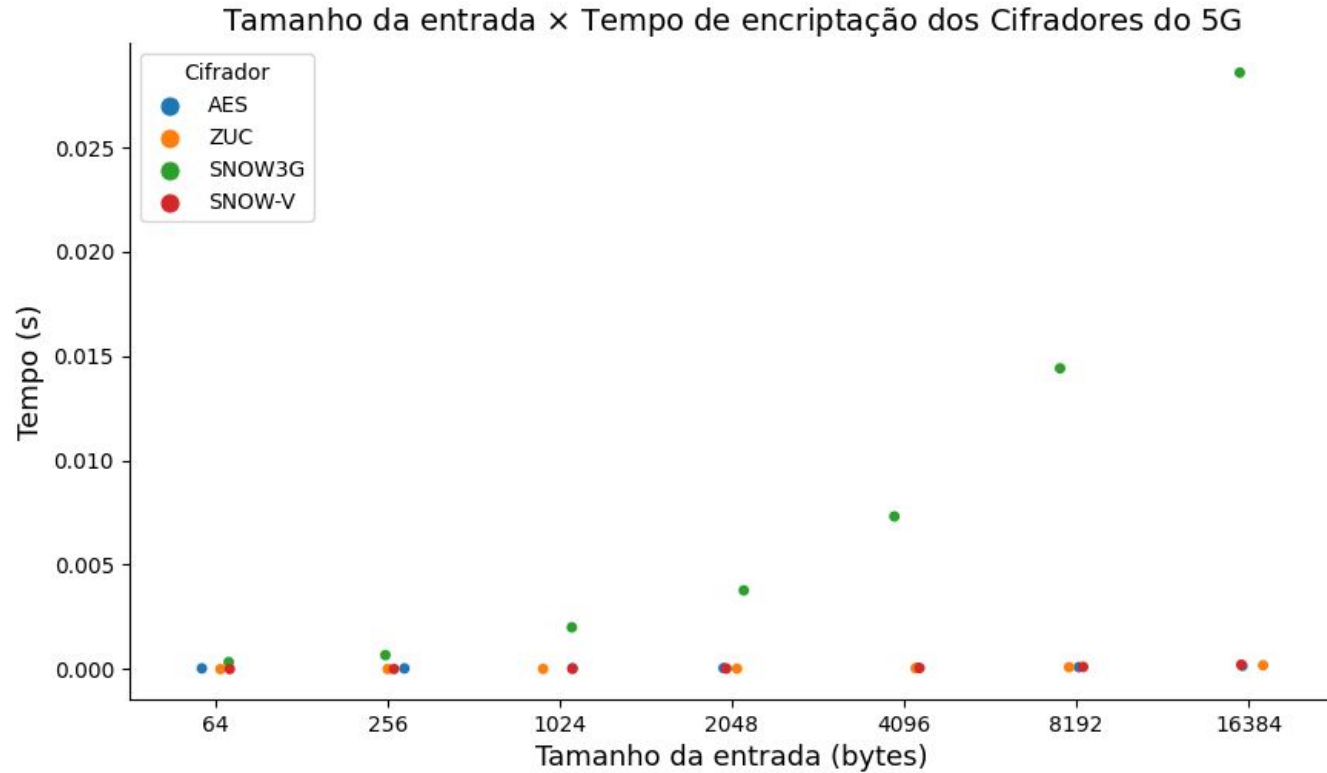
<https://github.com/mitshell/CryptoMobile>

SNOW-V 32 bits

Tabela 4: Comparação de Performance entre os algoritmos utilizados para encriptação no 5G, as medidas indicam tempo de execução em segundos.

Algoritmo	Tamanho da entrada do Encriptador						
	64 bytes	256 bytes	1024 bytes	2048 bytes	4096 bytes	8192 bytes	16384 bytes
AES	3.95E-05	3.85E-05	4.45E-05	5.21E-05	6.72E-05	9.85E-05	1.59E-04
ZUC	4.23E-06	6.21E-06	1.47E-05	2.70E-05	4.97E-05	9.58E-05	1.87E-04
SNOW3G	3.42E-04	6.73E-04	2.00E-03	3.77E-03	7.31E-03	1.44E-02	2.86E-02
SNOW-V	6.70E-06	9.49E-06	1.85E-05	3.20E-05	5.75E-05	1.09E-04	2.12E-04

SNOW-V 32 bits



Referências

- Patrik Ekdahl, Thomas Johansson, Alexander Maximov, & Jing Yang (2019). A new SNOW stream cipher called SNOW-V. *IACR Transactions on Symmetric Cryptology*, 2019, Issue 3, 1-42.
- Aleksandar Kircanski and Amr M Youssef. *On the sliding property of SNOW 3G and SNOW 2.0*. *IET Information Security*, 5(4):199–206, 2011.
- 3GPP. Work item on network functions virtualisation.
<http://www.3gpp.org/more/1584-nfv>.
- 3GPP SA3. TR 33.841 study on supporting 256-bit algorithms for 5G., 2018.
<https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3422>.

Credits

Special thanks to all the people who made and released these awesome resources for free:

- ▷ Presentation template by SlidesCarnival
- ▷ Photographs by Unsplash

Obrigado!

Duvidas?