Estudo e Implementação do SNOW em C e Python

Jhonatan Cléto[1]

[1]Instituto de Computação – Universidade Estadual de Campinas (UNICAMP)

Abstract. *In this report, we will present a study and implementation of SNOW-V, a new member of the SNOW family of stream ciphers. SNOW-V aims to meet the industrial demand for high-speed encryption in virtualized environments, something that may become relevant in the future 5G mobile network system. The SNOW architecture has been updated making it competitive in a purely software environment, using vector instructions and AES acceleration present in modern CPUs. The result was an algorithm that, even with a software implementation, without vector instructions, has performance comparable to the best AES-128 unaccelerated software implementation, offering a level of security no worse than AES-256.*

Resumo. *Neste relatório, apresentaremos um estudo e implementação do SNOW-V, um novo membro da família SNOW de cifras de fluxo. O SNOW-V objetiva atender a demanda industrial por alta velocidade de encriptação em ambientes virtualizados, algo que pode se tornar relevante no futuro sistema de redes móveis 5G. A arquitetura SNOW foi atualizada tornando-a competitiva em ambiente puramente software, utilizando instruções vetoriais e de aceleração do AES presentes nas CPUs modernas. O resultado foi um algoritmo que mesmo com uma implementação em software, sem instruções vetoriais, apresenta performance comparável a melhor implementação sem aceleração do AES-128 em software, oferecendo um nível de segurança não pior que o AES-256.*

1. Introdução

O novo padrão de tecnologia para redes móveis o 5G, foi aprovado pelo 3GPP (3rd Generation Partnership Project) no final de 2018 e se encontra em processo de implantação em vários países, inclusive no Brasil. Essa nova geração de tecnologias para redes móveis, trouxe uma série de mudanças significativas na arquitetura do sistema e nos requisitos de segurança, que em muitos casos invalidam os algoritmos criptográficos utilizados nas gerações anteriores das tecnologias de redes móveis.

Em todas as gerações de sistemas de telefonia móvel do 3GPP, a figura central na segurança de link é a chave compartilhada entre o dispositivo - o equipamento do usuário, um smartphone por exemplo - e a rede local, o operador da rede móvel do qual o usuário tem um contrato de serviço e do qual ele recebe as suas credenciais. Da chave compartilhada, é criado um conjunto de chaves derivadas, que são utilizadas para proteção de integridade e confidencialidade nos canais de comunicação utilizados pelo usuário.

No 4G, a geração anterior e predominante de tecnologias de redes móveis, são definidos três algoritmos para integridade (128-EIAx) e confidencialidade (128-EEAx), baseados em três diferentes primitivas criptográficas, SNOW 3G [SAG06], AES [Pub01] e ZUC [SAG11]. Todos os algoritmos padronizados no 4G utilizam chaves com 128 bits de tamanho e estão descritos na Tabela 1.

Algoritmo	Encriptação	Integridade
SNOW 3G	EEA1	EIA1
AES	EEA2	EIA2
ZUC	EEA3	EIA3

Tabela 1. Algoritmos utilizados no 4G para integridade e confidencialidade.

Devido às mudanças na arquitetura do sistema do 5G, os algoritmos então utilizados no sistema 4G, enfrentam uma série de desafios. No novo sistema, o 3GPP avalia aumentar o nível de segurança das chaves para 256 bits de comprimento [SA318], mudança que afeta os algoritmos de encriptação pois eles devem ser capazes de acomodar o novo tamanho das chaves e serem fortes o suficiente para fornecer o nível de segurança desejado. Embora, para as primitivas criptográficas utilizadas no 4G, existam versões que utilizam chaves de 256 bits, no sistema 5G muitos dos nós da rede serão virtualizados [3GP]. Com isso, a capacidade de utilizar hardware especializado para as primitivas criptográficas se tornará menos viável.

Atualmente, os processadores modernos tanto na arquitetura x86-64 quanto na arquitetura ARM, incluem instruções para acelerar os cálculos do AES, tornando possível a elaboração de implementações extremamente otimizadas, que alcançam facilmente velocidades entre 20-25 Gbps nos algoritmos EIA2 e EEA2. No entanto, no caso das primitivas SNOW 3G e ZUC, se faz necessário encontrar outras soluções, dado que as melhores implementações puramente em software do SNOW 3G, por exemplo, atinge em torno de 9 Gbps, valor bem distante da velocidade de 20 Gbps desejada para downlink no sistema 5G [ITU17].

Com o objetivo de satisfazer as demandas do sistema de comunicação móvel 5G, como o uso de algoritmos criptográficos com alto desempenho em ambientes virtualizados, nível de segurança das chaves em 256 bits e velocidade de downlink por volta de 20 Gbps, em 2019, pesquisadores da Ericsson em conjunto com pesquisadores da Lund Univesity, desenvolveram um novo membro da família SNOW batizado de SNOW-V (V de Virtualização) [Ekd+19].

O conjunto de algoritmos criptográficos SNOW [EJ00] foi introduzido no projeto Europeu NESSIE, que foi um concurso para novas primitivas criptográficas. Dois ataques foram descobertos [CHJ02; HR02] e o design do cifrador foi atualizado na versão SNOW 2.0 [EJ02]. O grupo de experts em algoritmos de segurança da ETSI, modificou o design do SNOW 2.0, propondo o cifrador SNOW 3G que é utilizado para a proteção da interface aérea das redes de telecomunicação padronizadas pelo 3GPP. Todos os algoritmos dessa família são baseados no método de cifra de fluxo. As Cifras de fluxo são algoritmos de chave simétrica, que encriptam mensagens através combinação dos blocos da mensagem com um fluxo de bits de cifragem derivado da chave através de um gerador de bits pseudo-aleatório. Normalmente, a combinação dos blocos da mensagem com o fluxo de bits é realizada através de uma operação XOR (disjunção exclusiva).

O SNOW-V é uma versão revisada do SNOW 3G, desenvolvido para ser competitivo em implementações puramente em nível de software. Para tanto, ele aproveita de algumas das instruções para aceleração do AES nas arquiteturas modernas, bem como faz uso das instruções SIMD (Single Instruction Multiple Data) que trabalham com gran-

des vetores de inteiros. Além disso, ele também é equipado com um modo de operação para Encriptação Autenticada com Dados Associados (AEAD), de modo a prover ambas confidencialidade e integridade.

Neste relatório, apresentaremos um estudo sobre o design do SNOW-V, assim como uma implementação do mesmo nas linguagens de programação C e Python. O relatório é organizado como segue. Na Seção 2, nós apresentamos o design do SNOW-V, incluindo o pseudo-código. Na Seção 3, descrevemos como utilizar o SNOW-V no modo de operação AEAD. A implementação em software é apresentada no Seção 4. Os resultados obtidos são discutidos na Seção 5. As conclusões do relatório são apresentadas na Seção 6.

2. Design do SNOW-V

Um Overview do design do SNOW-V é apresentada na Figura 1. Em conformidade com os outros algoritmos da família SNOW, o SNOW-V apresenta em seu design duas estruturas principais, o Linear Feedback Shift Register (LFSR) e a Finite State Machine (FSM).

LFSR é um registrador de deslocamento que, quando sincronizado, move o sinal através do registrador de um bit para o próximo bit mais significativo. Geralmente, algumas das saídas são combinadas utilizando a operação XOR, para formar um mecanismo de feedback. O valor inicial de um LFSR é chamado de seed e por realizar uma operação determinística no registrador, o fluxo de valores produzidos pelo registrador é completamente previsível utilizando seu estado atual ou anterior. Além disso, pelo registrador possuir um número finito de possíveis estados, ele eventualmente entrará em um ciclo de repetição. No entanto, para um LFSR com uma função de feedback bem planejada, pode produzir uma sequência de bits que aparenta aleatoriedade e que possui um ciclo de repetição consideravelmente longo.

FSM é um modelo matemático de computação, que é uma abstração de uma máquina que pode estar em um único de um número finito de estados em um dado tempo. Uma FSM pode mudar de um estado a outro em resposta a algumas entradas.

Começando da parte LFSR do SNOW-V, como visto na Figura 1, ela apresenta dois LFSRs nomeados como LFSR-A e LFSR-B, ambos com 16 blocos com 16 bits cada, totalizando, 256 bits por LFSR em uma construção circular. Os 32 blocos são denotados $a15 \ldots a0$ e $b15 \ldots b0$ respectivamente.

Cada bloco representa um elemento em $\mathbf{F}_{2^{16}}$, mas cada LFSR tem um polinômio de geração distinto. Os elementos do LFSR-A são gerados pelo polinômio

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

e os elementos do LFSR-B são gerados pelo polinômio

$$g^B(x) = x^{16} + x^{15} + x^{14} + x^{11} + x^8 + x^6 + x^5 + x + 1 \in \mathbf{F}_2[x]. \tag{2}$$

Quando consideramos os elementos de $\mathbf{F}_{2^{16}}$ como palavras, a posição x^0 é o bit menos significativo da palavra. Seja α um gerador de $g^A(x)$ e β um gerador de $g^B(x)$. No tempo $t \geq 0$, denotamos os estados dos LFSRs como $(a_{15}^{(t)}, a_{14}^{(t)}, \ldots, a_0^{(t)})$, com $a_i^{(t)} \in$

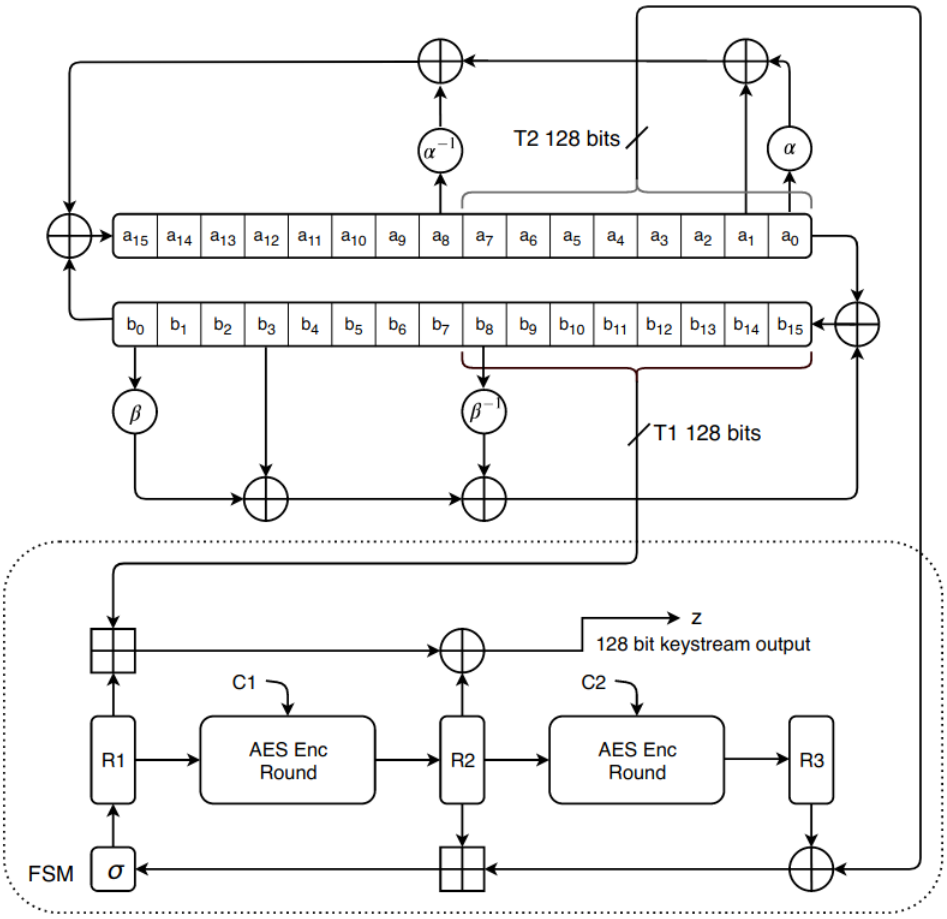

Figura 1. Overview esquemático do SNOW-V.

$\mathbf{F}_{2^{16}}$ e $(b_{15}^{(t)}, b_{14}^{(t)}, \ldots, b_0^{(t)})$, com $b_i^{(t)} \in \mathbf{F}_{2^{16}}$ respectivamente para LFSR-A e LFSR-B. Na Figura 1, os elementos $a_0^{(t)}$ e $b_0^{(t)}$ são as primeiras saídas dos LFSRs. Os LFSRs produzem sequências $a^{(t)}$ e $b^{(t)}$, $t \geq 0$ como é visto nas expressões (3) e (4).

$$a^{(t+16)} = b^{(t)} + \alpha a^{(t)} + a^{(t+1)} + \alpha^{-1} a^{(t+8)} \mod g^A(\alpha) \tag{3}$$

$$b^{(t+16)} = a^{(t)} + \beta b^{(t)} + b^{(t+3)} + \beta^{-1} b^{(t+8)} \mod g^B(\beta) \tag{4}$$

Nas quais, os estados iniciais dos LFSRs são dados por $(a^{(15)}, a^{(14)}, \ldots, a^{(0)})$ e $(b^{(15)}, b^{(14)}, \ldots, b^{(0)})$. A notação α^{-1} e β^{-1}, indicam os inversos multiplicativos de α e β em seus respectivos campos finitos. As equações (3) e (4) podem ser interpretadas como um padrão de bits implícito que preserva a conversão entre os campos.

Toda vez que a parte LFSR é atualizada, são efetuadas 8 operações com as equações $g^A(x)$ e $g^B(x)$, com isso 256 bits de um estado total de 512 bits são atualizados em um único passo. Com isso, duas fitas $T1$ e $T2$ terão novos valores. A fita $T1$ é formada considerando $(b_{15}, b_{14}, \ldots, b_8)$ como uma palavra de 128 bits, na qual b_8 é a parte menos significativa. Similarmente, $T2$ é formado considerando $(a_7, a_6, \ldots, a_0)$ como uma palavra de 128 bits com a_0 sendo a parte menos significativa. O mapeamento das fitas é visto na Figura 2, e as suas expressões são dadas por (5) e (6).

$$T1^{(t)} = (b_{15}^{(8t)}, b_{14}^{(8t)}, \ldots, b_{8}^{(8t)}) \tag{5}$$

$$T2^{(t)} = (a_{7}^{(8t)}, a_{6}^{(8t)}, \ldots, a_{0}^{(8t)}) \tag{6}$$

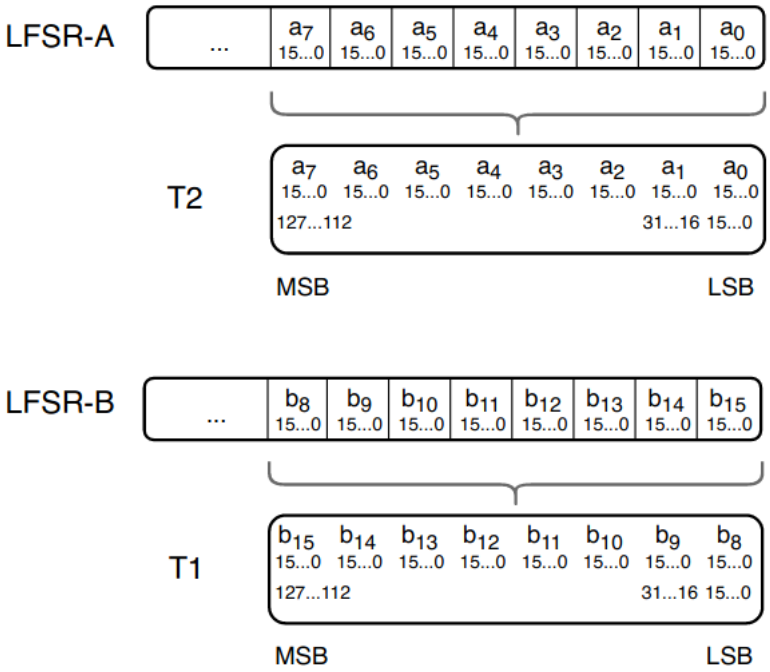

Figura 2. Mapeamento de palavras de 16-bits dos LFSRs em palavras de 128-bits T1 e T2.

Em relação a FSM ela utiliza dois blocos $T1$ e $T2$ da parte LFSR como entradas para produzir um fluxo de 128 bits como saída. Os registradores $R1$, $R2$ e $R3$ têm armazenam 128 bits, $\oplus$ representa a operação XOR bit a bit, e $\boxplus_{32}$ representa a aplicação paralela de 4 adições módulo 2^{32} sobre cada palavra.

A saída, $z^{(t)}$ no tempo $t \geq 0$, é dada pela expressão (7).

$$z^{(t)} = (R1^{(t)} \boxplus_{32} T1^{(t)}) \oplus R2^{(t)}. \tag{7}$$

Os registradores $R2$ e $R3$ são atualizados através de uma rodada completa de encriptação do AES ($AES^R(IN, KEY)$), como é visto na Figura 3. O Mapeamento entre os registradores de 128 bits e a matriz de estados do AES segue a definição de [Pub01] e é visto na Figura 4.

Podemos escrever a expressão de entrada de cada registrador como visto em (8), (9) e (10).

$$R1^{(t+1)} = \sigma(R2^{(t)} \boxplus_3 2(R3^{(t)} \oplus T2^{(t)})), \tag{8}$$

$$R2^{(t+1)} = AES^R(R1^{(t)}, C1), \tag{9}$$

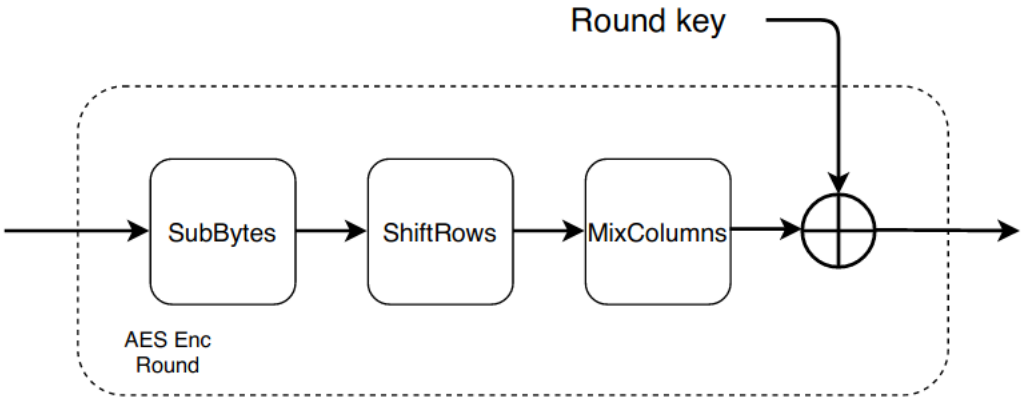

Figura 3. Funções internas da rodada completa de encriptação do AES.

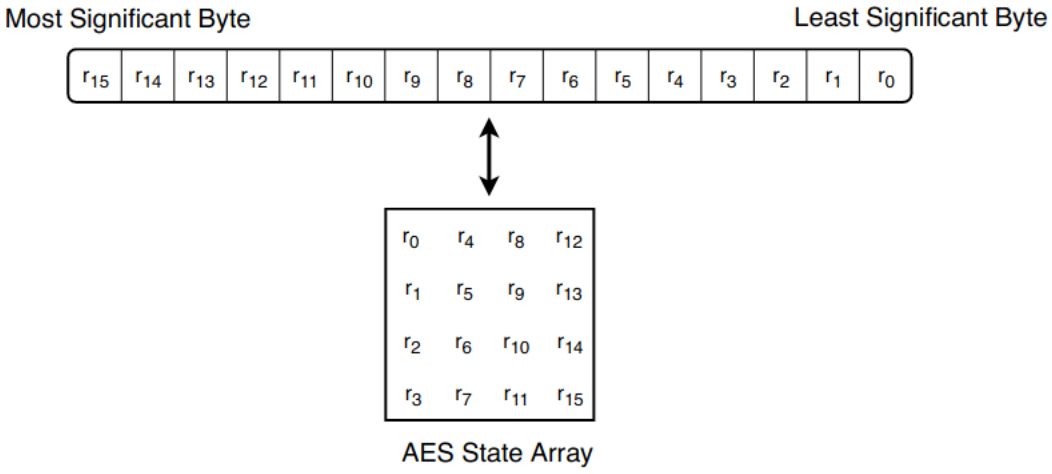

Figura 4. Mapeamento entre um registrador de 128 bits e a matriz de estados do AES.

$$R3^{(t+1)} = AES^R(R2^{(t)}, C1) \tag{10}$$

Os valores $C1$ e $C2$ são chaves constantes definidas como sequências de zeros e σ denota uma permutação de bytes dada por

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

Que é interpretada como o byte 0 é movido para a posição 0, o byte 4 é movido para a posição 1, o byte 8 é movido para a posição 2 e assim por diante. A posição 0 é a menos significativa. A permutação escolhida representa a transposição da matriz de estados do AES.

2.1. Inicialização

Antes de começar a produzir o fluxo de bits é necessário uma fase de inicialização do SNOW-V. Para tanto, o algoritmo tem como entrada uma chave de 256 bits e um vetor

de inicialização de 128 bits. A chave é denotada como $K = (k_{15}, k_{14}, \ldots, k_1, k_0)$, cada k_i, $0 \leq i \leq 15$, é uma palavra de 16 bits. O vetor de inicialização é denotado como $IV = (iv_7, iv_6, \ldots, iv_0)$, cada iv_i, $0 \leq i \leq 7$ é uma palavra de 16 bits.

O primeiro passo da inicialização consistem em carregar a chave K e o vetor de inicialização IV nos LFSRs definindo

$$(a_{15}, a_{14}, \ldots, a_0) = (k_7, k_6, \ldots, k_0, iv_7, iv_6, \ldots, iv_0)$$

e

$$(b_{15}, b_{14}, \ldots, b_0) = (k_{15}, k_{14}, \ldots, k_8, 0, 0, \ldots, 0).$$

A inicialização consistem em 16 rodadas, nas quais o cifrador é atualizado da mesma forma que quando está gerando o fluxo de bits. A única exceção é que os 128 bits de saída são combinados com as posições $(a_{15}, a_{14}, \ldots, a_8)$ da estrutura LFSR através de uma operação xor em cada rodada. Adicionalmente, nas duas ultimas rodadas da inicialização, a chave é combinada com o registrador $R1$, segundo os autores do SNOW-V esse procedimento foi inspirado por [HK18] e objetiva aumentar a segurança da cifra.

Outras decisões de projeto do SNOW-V são que o tamanho máximo de um fluxo de bits gerados por um par (K, IV) é 2^{64} bits e cada chave pode ser utilizada com no máximo 2^{64} vetores de inicialização diferentes. Segundo os autores, parece não existir nenhum caso prático que faça sentido violar essas limitações. O Algoritmo completo do SNOW-V é resumido em pseudocódigo nas Figuras 5 e 6.

Algorithm 1 SNOW-V initialization

procedure INITIALIZATION(K, IV)
  $(a_{15}, a_{14}, \ldots, a_8) \leftarrow (k_7, k_6, \ldots, k_0)$
  $(a_7, a_6, \ldots, a_0) \leftarrow (iv_7, iv_6, \ldots, iv_0)$
  $(b_{15}, b_{14}, \ldots, b_8) \leftarrow (k_{15}, k_{14}, \ldots, k_8)$
  $(b_7, b_6, \ldots, b_0) \leftarrow (0, 0, \ldots, 0)$
  $R1, R2, R3 \leftarrow 0, 0, 0$
  for $t = 1...16$ **do**
    $T1 \leftarrow (b_{15}, b_{14}, \ldots, b_8)$
    $z \leftarrow (R1 \boxplus_{32} T1) \oplus R2$
    $FSMupdate()$
    $LFSRupdate()$
    $(a_{15}, a_{14}, \ldots, a_8) \leftarrow (a_{15}, a_{14}, \ldots, a_8) \oplus z$
    if $t = 15$ **then** $R1 \leftarrow R1 \oplus (k_7, k_6, \ldots, k_0)$
    if $t = 16$ **then** $R1 \leftarrow R1 \oplus (k_{15}, k_{14}, \ldots, k_8)$

Figura 5. Inicialização do SNOW-V

3. Modo de Operação AEAD

No artigo do SNOW-V, os autores sugerem uma forma de utilizar o cifrador para prover Encriptação Autenticada com Dados Associados (AEAD). Para tanto, eles mostram como utilizar o SNOW-V em conjunto com o uma modificação do modo de operação GCM (Galois Counter Mode) [Dwo07]. No GCM, um cifrador de blocos é utilizado em modo

Algorithm 2 SNOW-V algorithm

procedure SNOW-V(K, IV)
  INITIALIZATION(K, IV)
  while more keystream blocks needed **do**
    $T1 \leftarrow (b_{15}, b_{14}, \ldots, b_8)$
    $z \leftarrow (R1 \boxplus_{32} T1) \oplus R2$
    $FSMupdate()$
    $LFSRupdate()$
    Output keystream symbol z

Algorithm 3 LFSR update algorithm

procedure $LFSRupdate()$
  for $i = 0...7$ **do**
    $tmp_a \leftarrow b_0 + \alpha a_0 + a_1 + \alpha^{-1} a_8 \bmod g^A(\alpha)$
    $tmp_b \leftarrow a_0 + \beta b_0 + b_3 + \beta^{-1} b_8 \bmod g^B(\beta)$
    $(a_{15}, a_{14}, \ldots, a_0) \leftarrow (tmp_a, a_{15}, \ldots, a_1)$
    $(b_{15}, b_{14}, \ldots, b_0) \leftarrow (tmp_b, b_{15}, \ldots, b_1)$

Algorithm 4 FSM update algorithm

procedure $FSMupdate()$
  $T2 \leftarrow (a_7, a_6, \ldots, a_0)$
  $tmp \leftarrow R2 \boxplus_{32} (R3 \oplus T2)$
  $R3 \leftarrow AES^R(R2)$ ▷ Note that the round keys for these AES
  $R2 \leftarrow AES^R(R1)$ ▷ encryption rounds are $C1 = C2 = 0$
  $R1 \leftarrow \sigma(tmp)$

Figura 6. Algoritmo do SNOW-V

contador (Counter Mode) para encriptar a mensagem. Além disso, o cifrador de blocos é utilizado para produzir o código de autenticação da mensagem (MAC) e derivar uma chave H utilizada na função $GHASH_H$.

Utilizando o SNOW-V em conjunto com o algoritmo $GHASH_H$, a chave H é definida como o primeiro fluxo de 128 bits gerado pelo cifrador ($z^{(0)}$). O segundo fluxo de 128 bits ($z^{(1)}$) é utilizado como máscara para a construção do MAC. Para encriptar os blocos da mensagem, são utilizados as saídas seguintes da cifra, fornecendo os blocos de texto cifrado como entradas para o $GHASH_H$.

Os autores comentam também que para utilizar o modo AEAD no SNOW-V, é necessário setar a parte menos significativa do LFSR-B para os seguintes valores hexadecimais:

$$(b_7, b_6, \ldots, b_0) = (6D6F, 6854, 676E, 694A, 2064, 6B45, 7865, 6C41) \quad (12)$$

Esses valores hexadecimais são os nomes dos autores codificados em UTF-8.

Uma visão geral de como utilizar o SNOW-V em conjunto com o algoritmo $GHASH_H$ é mostrado na Figura 7. O padding dos dados autenticados adicionais, como concatenar o tamanho desses dados como o tamanho do texto cifrado e todas as outras restrições no tamanho do texto cifrado e mudanças no IV do algoritmo original [Dwo07] são mantidas.

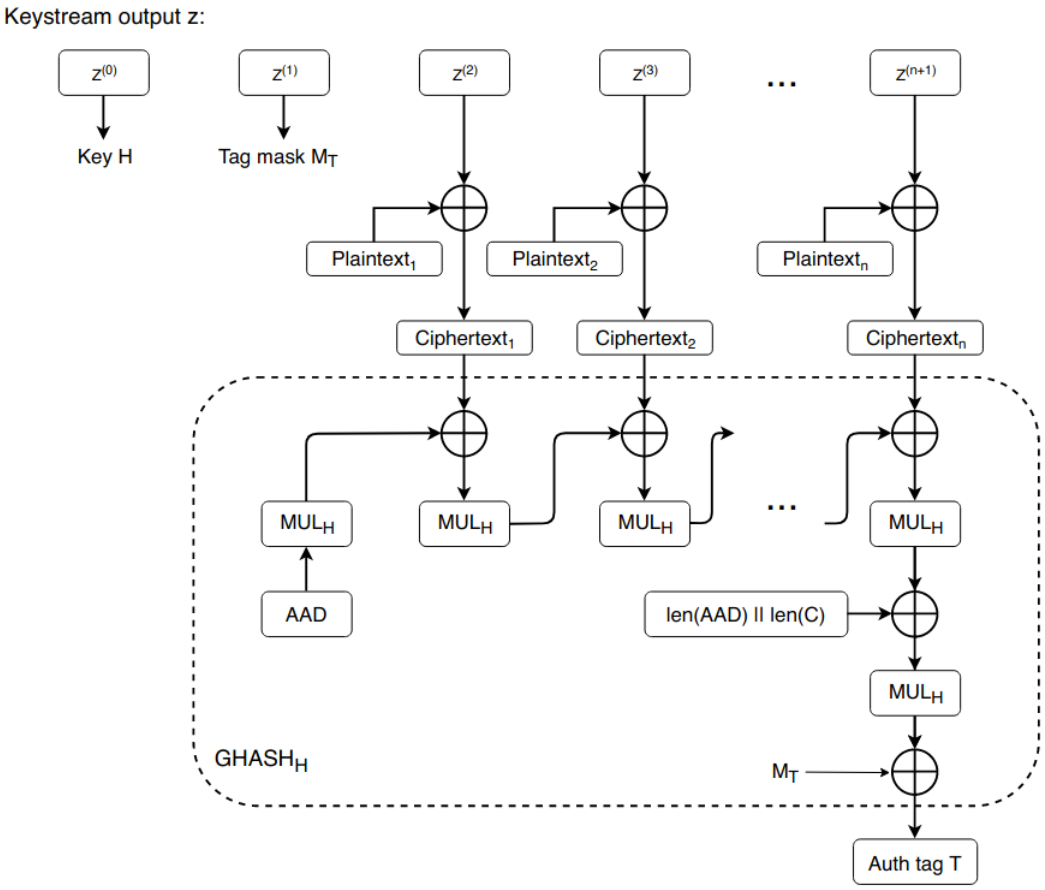

Figura 7. Como o SNOW-V é utilizado com o $GHASH_H$ para permitir AEAD.

4. Implementação em Software do SNOW-V

No artigo original do SNOW-V [Ekd+19] são apresentadas duas implementações em C++ do cifrador como referência, uma versão que não utiliza instruções vetoriais e outra que utiliza instruções vetoriais AVX2. Neste trabalho, o SNOW-V foi implementado utilizando a linguagem de programação C, tomando como base a primeira referência dos autores, que não faz uso de instruções vetoriais. Embora essa implementação tenha menor performance, por meio dela é mais simples entender as estruturas propostas no design do SNOW-V. Portanto, optamos por essa versão, uma vez que o objetivo do trabalho é estudar o cifrador.

Na implementação em C, são utilizadas palavras de 8, 16 e 32 bits para representar os componentes das estruturas do SNOW-V, a Tabela 2 indica qual a estrutura de dados foi utilizada para representar todos os componentes do cifrador.

A escolha em representar a chave (K) e o vetor de inicialização (IV) como vetores bytes, foi tomada com o objetivo de simplificar a transformação de uma cadeia de caracteres ASCII em um vetor de bytes, tornando possível trabalhar com esses componentes como strings. A escolha das representações para os componentes das estruturas LFSR e FSM, foram feitas de modo a alinhar a implementação com o design original visto na Seção 2.

A lista das operações criptográficas implementadas é vista na Tabela 3, como é

Componentes	Representação
K	32×8-bits
IV, z	16×8-bits
LFSRs	16×16-bits
$R1$, $R2$, $R3$	4×32-bits

Tabela 2. Componentes do SNOW-V e suas representações em C.

possível notar o cifrador implementado permite encriptar e decriptar mensagens, além de prover a implementação para o modo AEAD.

Operação	Descrição
snowv-encrypt	Encriptar mensagens utilizando o SNOW-V
snowv-decrypt	Decriptar mensagens utilizando o SNOW-V
snowv-gcm-encrypt	Encriptar mensagens utilizando o Modo AEAD
snowv-gcm-decrypt	Decriptar mensagens utilizando o Modo AEAD

Tabela 3. Operações criptográficas implementadas.

Um exemplo de código da implementação em C é vista no Algoritmo 1, o código completo é encontrado em [Clé22].

```
...
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();
}
...
```

Listing 1: Geração do Fluxo de bytes do SNOW-V.

Visando aumentar o nível de abstração do algoritmo, tornando seu uso mais simplificado, implementamos também uma Wrapper API do SNOW-V, de modo a tornar possível utilizar as operações criptográficas implementadas em aplicações escritas em Python, a implementação foi realizada utilizando a API C do CPython [Pyt].

Na implementação em Python, aplicamos Programação Orientada a Objetos para abstrair a interface do SNOW-V implementada em C. O algoritmo 2, ilustra a implementação da classe em Python que abstrai o cifrador. Na Figura 8, apresentamos um exemplo de uso da nossa implementação do SNOW-V.

Ademais, de modo a verificar a corretude da nossa implementação, executamos a bateria de testes disponibilizada no artigo original do SNOW-V [Ekd+19], observamos que todos os testes obtiveram as saídas esperadas [Clé22] mostrando que a implementação está correta.

```
class SNOWV(object):
    ...
    def encrypt(self, key, iv, plaintxt):
        if type(plaintxt) != bytes:
            plaintxt = bytes(plaintxt.encode('ascii'))
        try:
            return snowv_encrypt(key, iv, plaintxt)
        except ValueError as e:
            raise CMException(e)
    ...
```

Listing 2: Classe Python que Abstrai as funcionalidades do SNOW-V.

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

In [2]: snowv = SNOWV()

In [3]: key, iv = 32*b'j', 16*b'o'

In [4]: key, iv
Out[4]: (b'jjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjj', b'oooooooooooooooo')

In [5]: mensagem = 'Mensagem secreta!'

In [6]: encrypt = snowv.encrypt(key, iv, mensagem)

In [7]: encrypt
Out[7]: b'\xb7q\x1f6z\xdf\x95\xba\xe4\xeb\xbaF\xcb\x9f\xa4n~'

In [8]: snowv.decrypt(key, iv, encrypt)
Out[8]: b'Mensagem secreta!'
```

Figura 8. Exemplo de encriptação e decriptação de uma mensagem utilizando a implementação do SNOW-V utilizando Python.

5. Avaliação da Performance em Software do SNOW-V

No artigo original do SNOW-V, os autores apresentam alguns resultados comparando a implementação do cifrador utilizando instruções vetoriais AVX2, com as primitivas AES, ChaCha20 e SNOW-3G [Ekd+19]. Baseando-se nessa análise, realizamos uma comparação entre a nossa implementação do SNOW-V e os algoritmos padronizados para a segurança da rede no 4G e 5G.

Os algoritmos foram adquiridos em [Mit22], um repositório que contém as implementações de referência das primitivas criptográficas utilizada nas tecnologias de rede móvel. Nesse comparativo, foram utilizados os algoritmos AES-CTR (sem

instruções especializadas, porém otimizado), SNOW-3G e ZUC. As versões utilizadas desses cifradores, são implementações que utilizam chaves de 128 bits, conforme padronizado pelo 3GPP. Além disso, são implementadas em C com APIs em Python, seguindo o mesmo modelo adotado na nossa implementação apresentada na Seção 4.

Todos os testes de performance foram realizados em uma máquina equipada com a CPU Intel Xeon Silver 4108 @1.80GHz com Turbo Boost até @3.00GHz, testamos cada algoritmo em um único processo/thread para vários tamanhos de entradas em texto claro. Ademais, o compilador utilizado foi o GCC na versão 8.3.0 e o interpretador Python na versão 3.9.3.

Cada teste consistiu em avaliar o tempo que cada algoritmo leva para encriptar a entrada. Para cada configuração, foram tomadas 100 amostras e o tempo atribuído a configuração (algoritmo, entrada) foi a média das 100 amostras. Os resultados dos testes são vistos na Tabela 4.

Algoritmo	**Tamanho da entrada do Encriptador em bytes**						
	64	**256**	**1024**	**2048**	**4096**	**8192**	**16384**
AES-CTR	4.0E-05	3.9E-05	4.5E-05	5.2E-05	6.7E-05	9.9E-05	1.6E-04
ZUC	4.2E-06	6.2E-06	1.5E-05	2.7E-05	5.0E-05	9.6E-05	1.9E-04
SNOW3G	3.4E-04	6.7E-04	2.0E-03	3.8E-03	7.3E-03	1.4E-02	2.7E-02
SNOW-V	**6.7E-06**	**9.5E-06**	**1.9E-05**	**3.2E-05**	**5.8E-05**	**1.1E-04**	**2.1E-04**

Tabela 4. Comparação de Performance entre o SNOW-V e os algoritmos utilizados para encriptação no 5G, as medidas indicam tempo de execução em segundos.

Na Figura 9, temos uma visualização dos resultados apresentados na Tabela 4, como é possível verificar, a nossa versão do SNOW-V apresenta um desempenho pareável ao AES-CTR e ao ZUC, mesmo trabalhando com uma chave de tamanho maior e sendo uma versão não otimizada do cifrador. Comparando o SNOW-V com o SNOW-3G, que apresentou os piores resultados nos testes, notamos uma grande evolução na arquitetura SNOW. Enquanto o tempo de execução do SNOW-3G aumenta rapidamente conforme a entrada aumenta, o SNOW-V tem um crescimento menos acentuado nesse intervalo.

Na Figura 10, temos um comparativo da performance relativa dos algoritmos considerando o AES-CTR como referencial, onde notamos novamente o bom desempenho do SNOW-V. Nesse gráfico, observamos ainda uma vantagem para o algoritmo ZUC, que teve o melhor desempenho na maioria dos testes. No entanto, conforme o tamanho da entrada cresce, percebemos redução no desempenho dos cifradores em relação ao AES-CTR, sendo que para mensagens com tamanho maior que 8192 bytes, eles apresentam desempenho inferior. Essa vantagem do AES-CTR em mensagens maiores, pode ser causada pelo fato de ser uma versão otimizada, enquanto os demais cifradores, são baseados nas suas implementações de referência.

6. Conclusões

Neste trabalho, estudamos o SNOW-V um Cifrador de Fluxo de 128 bits. Ele segue os princípios de design das versões anteriores da família SNOW, mas é projetado para aproveitar as instruções vetoriais suportadas pela maioria das CPUs modernas.

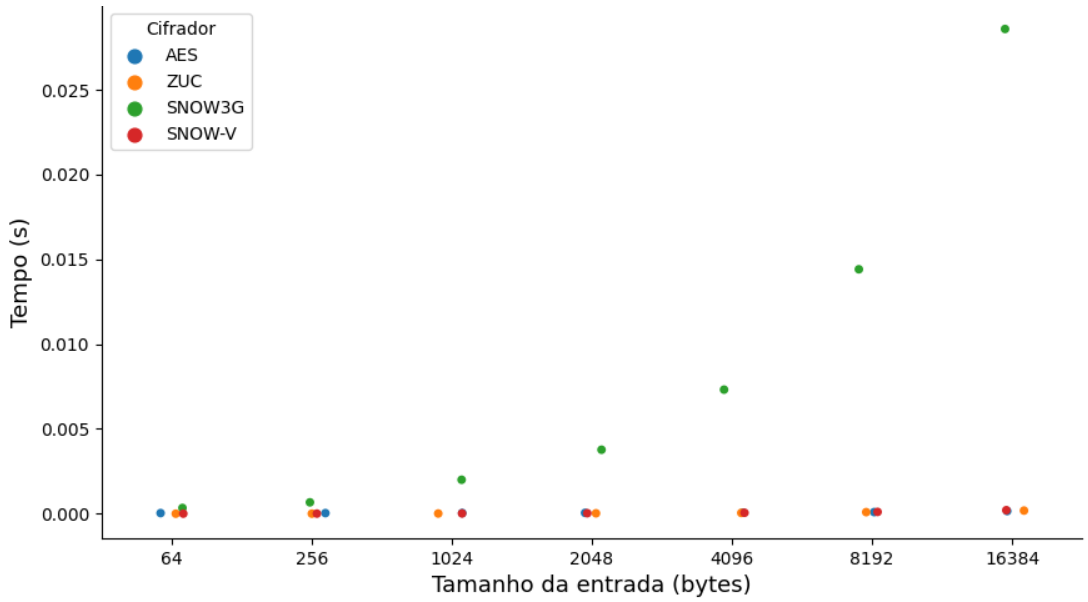

Figura 9. Tempo para encriptar a mensagem em função do tamanho da mensagem para diferentes algoritmos utilizados no 5G.

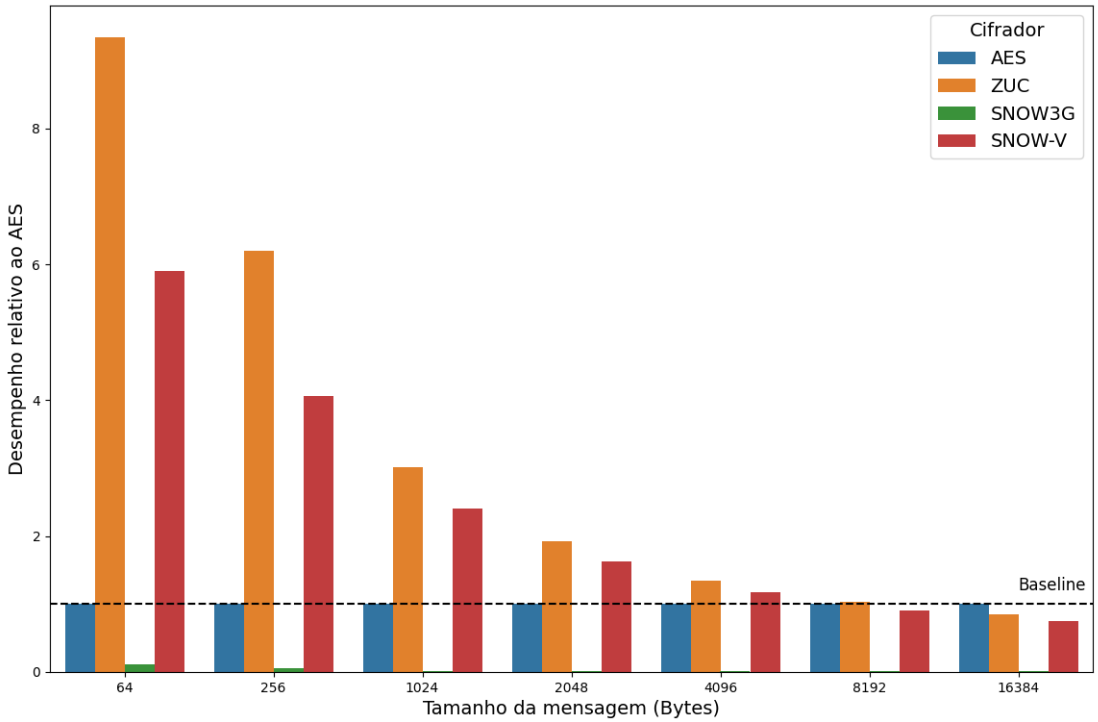

Figura 10. Desempenho dos algoritmos de encriptação do 5G em relação ao AES-CTR

O design do SNOW-V o permite obter grande performance em ambientes virtualizados (puramente software), atendendo as demandas do novo sistema 5G. Apresentamos um modo de operação AEAD baseado no GCM, que fornece ambas confidencialidade e autenticidade sobre os dados.

Mostramos ainda que, mesmo uma implementação não otimizada do SNOW-V consegue atingir uma performance comparável ao AES-128 sem instruções especializadas em tarefas de encriptação para textos com tamanhos acima de 256 bytes, oferecendo um nível de segurança superior.

O SNOW-V se mostrou uma primitiva criptográfica com um alto potencial de padronização tanto para os sistemas 5G quanto para outros contextos de ambientes virtualizados como na computação em nuvem, devido ao seu excelente desempenho.

Por fim, o estudo e implementação do SNOW-V realizados neste trabalho, me permitiu aplicar os conhecimentos adquiridos durante o curso, além de experienciar os desafios enfrentados no processo de implementação de primitivas criptográficas, proporcionando um aprofundamento no meu conhecimento sobre criptografia.

Referências

[EJ00] Patrik Ekdahl e Thomas Johansson. “SNOW-a new stream cipher”. Em: *Proceedings of first open NESSIE workshop, KU-Leuven*. Citeseer. 2000, pp. 167–168.

[Pub01] NIST FIPS Pub. “197: Advanced encryption standard (AES)”. Em: *Federal information processing standards publication* 197.441 (2001), p. 0311.

[CHJ02] Don Coppersmith, Shai Halevi e Charanjit Jutla. “Cryptanalysis of stream ciphers with linear masking”. Em: *Annual International Cryptology Conference*. Springer. 2002, pp. 515–532.

[EJ02] Patrik Ekdahl e Thomas Johansson. “A New Version of the Stream Cipher SNOW”. Em: *Selected Areas in Cryptography*. 2002.

[HR02] Philip Hawkes e Gregory G Rose. “Guess-and-determine attacks on SNOW”. Em: *International Workshop on Selected Areas in Cryptography*. Springer. 2002, pp. 37–46.

[SAG06] SAGE. *Specification of the 3GPP Confidentiality and Integrity Algorithms UEA2 & UIA2*. Versão 1.1. 2006. https://www.gsma.com/aboutus/wp-content/uploads/2014/12/snow3gspec.pdf.

[Dwo07] Morris J Dworkin. *Sp 800-38d. recommendation for block cipher modes of operation: Galois/counter mode (gcm) and gmac*. National Institute of Standards & Technology, 2007.

[SAG11] SAGE. *Specification of the 3GPP Confidentiality and Integrity Algorithms 128-EEA3 & 128-EIA3. Document 2: ZUC Specification*. Versão 1.6. 2011. https://www.gsma.com/security/wp-content/uploads/2019/05/eea3eia3zucv16.pdf.

[ITU17] ITU. *Minimum requirements related to technical performance for IMT-2020 radio interface(s)*. Versão 1.0. 2017. https://www.itu.int/pub/R-REP-M.2410-2017.

[HK18] Matthias Hamann e Matthias Krause. “On stream ciphers with provable beyond-the-birthday-bound security against time-memory-data tradeoff attacks”. Em: *Cryptography and Communications* 10 (2018), pp. 959–1012.

[SA318] 3GPP SA3. *TR 33.841 study on supporting 256-bit algorithms for 5G*. 2018. https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3422.

[Ekd+19] Patrik Ekdahl et al. “A new SNOW stream cipher called SNOW-V”. Em: *IACR Transactions on Symmetric Cryptology* 2019, Issue 3 (2019), pp. 1–42. DOI: 10.13154/tosc.v2019.i3.1-42. https://tosc.iacr.org/index.php/ToSC/article/view/8356.

[Clé22] Jhonatan Cléto. *Projeto Final - MC889*. 2022. https://github.com/Jhon-Cleto/MC889.

[Mit22] Mitshell. *CryptoMobile*. 2022. https://github.com/mitshell/CryptoMobile.

[3GP] 3GPP. *Work item on network functions virtualisation*. https://www.3gpp.org/more/1584-nfv.

[Pyt] Python. *Python/C API Reference Manual*. https://docs.python.org/3/c-api/index.html.