Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Seguridad en 5G y 6G

Security in 5G and 6G

Gabriel A. Luis Freitas

La Laguna, 26 de mayo de 2023

Dña. **Pino Caballero Gil**, con N.I.F. 45.534.310-Z Catedrática de Universidad adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutora.

Dña. **Jezabel Molina Gil**, con N.I.F. 78.507.682-B Profesora Ayudante Doctora adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutora

C E R T I F I C A N

Que la presente memoria titulada:

”*Seguridad en 5G y 6G*”

ha sido realizada bajo su dirección por D. **Gabriel A. Luis Freitas**, con N.I.F. 43.384.512-A.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos, firman la presente en La Laguna a 26 de mayo de 2023

Agradecimientos

A mi familia y amigos, por haber estado siempre a mi lado y apoyarme en todo momento de manera incondicional.

A mis tutoras Pino Caballero Gil y Jezabel Molina Gil, por ayudarme, aconsejarme y guiarme a lo largo del desarrollo del presente trabajo, y por orientarme sobre mi futuro desarrollo como profesional.

Licencia

Resumen

En este Trabajo de Fin de Grado se ha realizado un análisis de varias propuestas presentadas a estándar como cifrado para proteger la confidencialidad e integridad en el nuevo sistema de telefonía móvil 5G y el futuro paradigma 6G. Para llevarlo a cabo se realizó la implementación originalmente propuesta para los diferentes cifrados, y se adaptó a las diferentes plataformas disponibles en el mercado actual. Además se realizó una investigación exhaustiva del cifrado en flujo SNOW-Vi y se desarrolló una propuesta de mejora de su implementación. Esta propuesta está basada en el uso de la técnica de ventanas deslizantes de los vectores que componen una de las estructuras del cifrado, y ofrece una mejora de rendimiento entre el 14,6 % y el 34,6 %, dependiendo de la plataforma sobre la que se ejecute. De esta forma, en este trabajo se concluye que el cifrado SNOW-Vi se sitúa en una situación privilegiada dentro del marco de la telefonía móvil 6G.

Palabras clave: Confidencialidad, Integridad, Cifrado en flujo, SNOW-Vi, 5G, 6G, Rendimiento.

Abstract

In this Final Degree Project, an analysis has been carried out of several proposals presented to the standard to protect confidentiality and integrity in the new 5G mobile telephony system and the future 6G paradigm. In order to carry it out, the implementation originally proposed for the different ciphers was carried out, and adapted to the different platforms available in the current market. In addition, a deep research of the SNOW-Vi stream cipher was developed and a proposal for improving its implementation was made. This proposal is based on sliding windows of the vectors that make up one of the structures of the cipher, and offers a performance improvement between 14.6 % and 34.6 %, depending on the platform on which it is executed. Thus, this work concludes that SNOW-Vi encryption is in a privileged situation within the framework of 6G mobile telephony.

Keywords: Confidentiality, Integrity, Stream Cipher, SNOW-Vi, 5G, 6G, Performance.

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

La quinta generación de los sistemas de comunicación de telefonía móvil se encuentra desplegada en la mayoría de países desde el año 2020. Esto ha impulsado la velocidad de transmisión de estos sistemas hasta velocidades sustancialmente superiores a las que se habían utilizado hasta el momento, superando los 20 Gbps en el caso de la quinta generación. Por otra parte, como se puede leer en la publicación (1) publicada en el 2019 por el *6 Genesis Project*, la sexta generación alcanzará velocidades superiores a los 100 Gbps (ver Fig. 1.1). Esto conlleva un cambio sustancial en la mayoría de sectores relacionados con la telefonía móvil. Entre ellos destaca el ámbito de la ciberseguridad y la criptografía ya que los cifrados que se utilizan en la actualidad no fueron ideados teniendo en cuenta estas velocidades.

Figura 1.1: Últimas generaciones de telefonía móvil

1.1. Objetivo

Para la quinta generación de telefonía móvil, los cifrados propuestos a estándar de privacidad e integridad son variaciones de los cifrados de los que se disponía en generaciones anteriores. Entre estos cifrados se encuentran AES (2) en su versión de 256 bits, ZUC (3), que ha pasado por un proceso de rediseño para alcanzar el tamaño de clave requerido, y SNOW (4) (ver Fig. 1.2), que también ha recibido una actualización sustancial para poder adaptarse a los nuevos requerimientos. A estos cifrados se les suma un nuevo cifrado presentado en el año 2021, ROCCA (5), basado en AES y del que se afirma que cumple con los estándares de seguridad actuales y que alcanza velocidades de hasta 160 Gbps, convirtiéndose así en el algoritmo más veloz de los mencionados. Sin embargo, este cifrado cuenta con muy poco recorrido y por tanto hasta el momento se encuentra poca investigación sobre él.

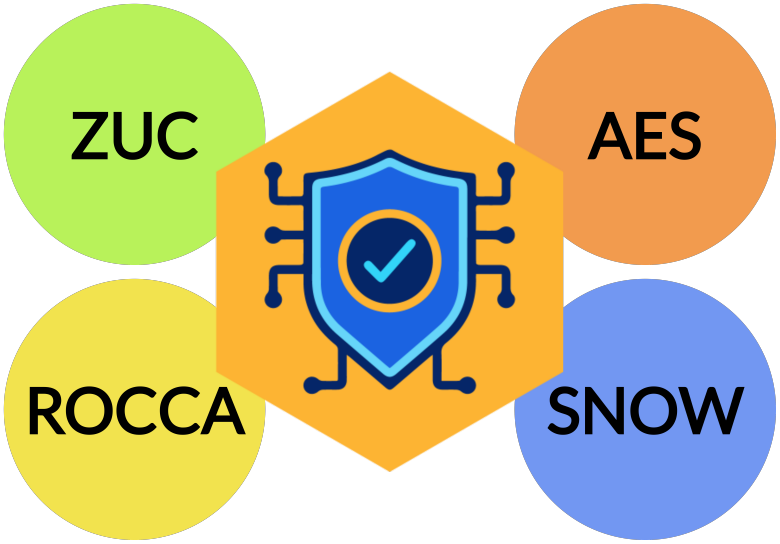

Figura 1.2: Cifrados estudiados

1.2. Estructura

A lo largo de este Trabajo de Fin de Grado se ha realizado un estudio de los principales cifrados propuestos a estándar para el paradigma actual de la telefonía Móvil. Se comenzó con un estudio teórico de los algoritmos y de la implementación propuesta para cada uno de ellos en las diferentes plataformas disponibles dentro del mercado actual. Posteriormente, se estudiaron esas implementaciones de los algoritmos de cifrado seleccionados. De esas investigaciones se derivó una posible mejora en la implementación del cifrado SNOW-Vi (6), con objeto de optimizar su velocidad, lo que lo convertiría en serio candidato para el paradigma 6G. Dicha propuesta de mejora se explica detalladamente en el capítulo 4, donde también se incluye la nueva implementación y los resultados obtenidos, comparándose con las velocidades originales que ofrecía ese cifrado. Finalmente, tras el presupuesto, en sucesivos apéndices se incluyen, respectivamente, la referencia de un artículo producto de este trabajo que ha sido aceptado en un congreso nacional y que será presentado en las próximas semanas, varias tablas de resultados, el código de la mejora propuesta, y el repositorio de las implementaciones realizadas.

Capítulo 2

Estado del arte

Los cifrados en flujo han sido un elemento clave en la seguridad de las transmisiones de datos en múltiples generaciones de sistemas de telefonía, constituyendo un pilar para garantizar la confidencialidad, integridad y autenticidad de la información enviada.

En la actualidad existe una amplia variedad de cifrados, tanto en flujo como en bloque, con múltiples características dependiendo del uso y el entorno. Los cifrados ZUC, AES o la familia SNOW son algunos de los mejor posicionados para convertirse en la base sobre la que se sustente la seguridad en las redes móviles.

2.1. Estándares 5G y 6G

Para los sistemas de telefonía móvil 4G, los cifrados que conformaban los sistemas de confidencialidad e integridad eran SNOW 3G (7), AES y ZUC-128, especificados en los documentos 128-EEA1, 128-EEA2 (8), 128-EEA3 (9), respectivamente. Estos mismos algoritmos también han sido seleccionados como sistemas criptográficos para el nuevo paradigma 5G. Sin embargo, para esta nueva generación, el proyecto 3GPP (*3rd Generation Partnership Project* (10) ha cambiado los requisitos exigidos, tanto en el ámbito de la seguridad como en el de la velocidad.

Uno de estos requisitos ha sido el incremento del nivel de seguridad de las claves utilizadas, requiriéndose claves de 256 bits de longitud para evitar ataques contra claves de menor tamaño. Este cambio se puede alcanzar de una forma sencilla en AES, ya que su versión de 256 bits lleva tiempo siendo utilizada. Sin embargo, los cifrados ZUC y SNOW 3G no fueron diseñados originalmente para trabajar con esta longitud de clave, por lo que se hace necesaria su adaptación. En el caso de ZUC se desarrolló una versión de 256 bits que se dio a conocer en 2018. Por otra parte, la adaptación de SNOW 3G recibió una actualización denominada SNOW-V (11).

Además, en el contexto de 5G, se ha establecido el requisito de alcanzar cuotas de velocidad específicas en entornos software de, como mínimo, 20 Gbps. Este mínimo de velocidad se establece para evitar que se ralenticen las comunicaciones o los dispositivos debido al uso de estos cifrados. Por otro lado, también se ha establecido la necesidad de que estos algoritmos puedan ser ejecutados en entornos totalmente virtualizados, debido a que estos cifrados deben funcionar en servidores y dispositivos móviles. En estos casos no es viable utilizar hardware especializado para mejorar el rendimiento, y, por tanto, la capacidad de funcionar con una velocidad determinada en entornos de software es un factor clave a considerar en el desarrollo de estos cifrados. Para el cifrado AES, este requisito no ha sido un impedimento, ya que la mayoría de CPUs disponibles

en la actualidad (prácticamente todos los procesadores Intel, AMD o ARM, exceptuando algunos casos en donde, por presupuesto, se recortan los módulos de estos procesadores) cuentan con un módulo específico de criptografía que incluye una ronda de AES como parte fundamental, debido a la necesidad de aceleración de este cifrado. Por otro lado, ni ZUC ni SNOW 3G cuentan con una característica similar que les permita alcanzar la cuota mínima de velocidad establecida en 20 Gbps para el cifrado en entornos de software. Debido a esto, es poco probable que puedan cumplir con este requisito en su forma actual. Por tanto, se hace necesario buscar alternativas o realizar mejoras en estos cifrados si se desea utilizar en este tipo de entornos.

En el caso de la familia SNOW, se aprovechó la incorporación de las instrucciones de procesador específicas para AES y de sets de instrucciones SIMD (Single Instruction Multiple Data), como AVX2, para acelerar el cifrado y así intentar alcanzar la cuota objetivo.

Por otro lado, teniendo en cuenta los requerimientos que traerá el futuro sistema de telefonía de sexta generación, muchos de estos cifrados buscan alcanzar velocidades cercanas a los 100 Gbps para poder permanecer dentro de este paradigma futuro.

Teniendo en cuenta la carrera de velocidades en la que se encuentran los cifrados que buscan convertirse en estándar de las telefonías móviles, cualquier cambio que ofrezca una mejora en velocidad sin comprometer el nivel de seguridad podría llegar a ser una pieza clave en los sistemas de redes de telefonías actuales.

2.2. Cifrados actuales y vulnerabilidades conocidas

2.2.1. AES

AES (*Advanced Encryption Standard*) (2) es un cifrado en bloque desarrollado por el NIST (*National Institute of Standards and Technology*) publicado en el año 2001 para convertirse en estándar en el año 2002. Este cifrado se basa en una red de sustitución-permutación de bloques y su uso es tan extenso que su iteración de cifrado viene implementada en la mayoría de CPUs de la actualidad.

En la actualidad no se ha publicado ningún ataque efectivo contra este cifrado, lo que ha llevado a que este cifrado sea utilizado por agencias como la NSA (*National Security Agency*) en el cifrado de sus documentos clasificados, o en los cifrados de extremo a extremo que se incluyen en las aplicaciones de telefonía actuales como Telegram, Signal o WhatsApp.

Gracias a la amplia implementación de AES en múltiples ámbitos, la mayoría de fabricantes de microprocesadores incluyen la iteración de AES como una de sus instrucciones básicas. Esto ha provocado que su velocidad haya aumentado hasta los 35 Gbps. Además, hasta la actualidad, se sigue considerando AES un cifrado totalmente seguro.

2.2.2. ZUC

ZUC (3) es un sistema de cifrado en flujo basado en un LFSR (*Linear Feedback Shift Register*) como generador pseudoaleatorio y un conjunto de operaciones no lineales para aumentar su resistencia. Este cifrado forma parte del conjunto de algoritmos para la protección de la confidencialidad e integridad en la tecnología LTE (*Long Term Evolution*) propuestos por *3GPP* y denotados 128-EEA3 y 128-EIA3 (9).

ZUC ha sido ampliamente utilizado en la telefonía móvil del pasado reciente en su versión de 128 bits. Además se ha propuesto como una posibilidad de estándar para las nuevas redes de telefonía 5G, modificando su tamaño de clave para alcanzar los 256 bits. Esta adaptación es relativamente sencilla, ya que solo difiere de su versión de 128 bits en la fase de inicialización y en la generación de los códigos de autenticación, mientras se mantiene el resto de aspectos de su versión anterior.

La versión de 256 bits ha demostrado ser segura ya que hasta el momento solo se ha podido atacar a una versión con un número de iteraciones de inicialización menor que el indicado por los proponentes.

Sin embargo, aunque este cifrado alcanza cuotas de velocidad suficientemente altas en entornos hardware, en entornos software no alcanza el objetivo, quedándose rezagado frente a sus competidores con cuotas de entre 1,5 Gbps y 3 Gbps, lo que le impediría cumplir los exigentes requisitos del estándar 5G.

2.2.3. Rocca

Rocca (5) es un cifrado bastante nuevo, presentado en el año 2021. Se basa en el uso de las instrucciones AES incluidas en la mayoría de procesadores y las instrucciones SIMD (*Single Instruction Multiple Data*) para acelerar el algoritmo y alcanzar los 160 Gbps, superando los requerimientos para los sistemas de transmisión 6G. La forma en que han logrado acelerar este algoritmo se basa en usar iteraciones de AES y operaciones XOR en cascada, mientras se realizan permutaciones entre las palabras de 128 bits que conforman el estado.

Sin embargo, a pesar de que este cifrado es uno de los más interesantes debido a la alta velocidad que ofrece, en el año 2022 se presentó una publicación (12) que afirma que es susceptible a ataques por fallo diferencial. Este ataque dio como resultado una recuperación total del estado interno. Concretamente, debido a que la función de actualización de Rocca es reversible, fue posible recuperar la clave original. Ese mismo año también se publicó un criptoanálisis del cifrado (13), que mostraba múltiples vulnerabilidades, principalmente a nivel de autenticación. Por el lado positivo, en ese trabajo se demostró un fuerte nivel de seguridad en cuanto a privacidad. Esta fortaleza le permitiría ser utilizado en esquemas en los que la autenticación no sea primordial, por ejemplo, en las aplicaciones de IoT (*Internet of Things*).

2.2.4. SNOW

Los cifrados SNOW (4) son un conjunto de cifrados en flujo basados en el uso de un LFSR y una FSM (*Finite State Machine*) para garantizar un nivel de seguridad e integridad elevado, pudiendo además ofrecer una gran velocidad de cifrado.

La primera versión de la familia SNOW es SNOW 1.0 (4). Este cifrado fue presentado al proyecto NESSIE en el año 2000, pero ofrecía múltiples vulnerabilidades y por ello recibió varios ataques exitosos (14), principalmente ataques por diferenciación (15). La mayoría de ataques solo necesitaron una salida de 2^{95} de longitud para poder determinar la clave utilizada. Estas vulnerabilidades forzaron la necesidad de actualizar el generador, lo que dio como fruto en el año 2002 a la segunda versión del cifrado, denominada SNOW 2.0 (16). Este cifrado proponía la modificación de las operaciones que se realizaban dentro de ambas estructuras. Dichos cambios aumentaron su nivel de seguridad ante los ataques que afectaban a la primera versión, lo que le valió para convertirse en uno de

los cifrados en flujo elegidos para el estándar ISO/IEC 18033-4. Sin embargo, durante la evaluación por parte del ETSI (*European Telecommunications Standards Institute*), se tomó la decisión de modificarlo para aumentar su capacidad de resistir ante ataques algebraicos. A esta modificación se le dio el nombre de SNOW 3G (7).

La versión 3G de este cifrado pertenece al conjunto UEA2 de cifrados en flujo de *3GPP* para la protección de la confidencialidad, como reemplazo del algoritmo KASUMI. Este algoritmo, al igual que sus predecesores, partía de una estructura basada en un LFSR y una FSM, como se puede apreciar en la Fig. 2.1.

SNOW 3G fue utilizada en la tecnología móvil para las tecnologías UMTS (3G) y LTE (4G), y aunque este algoritmo parte de una clave de 128 bits, se planteó su modificación para que admitiera claves de 256 bits con el objetivo de mantenerlo como candidato para la nueva generación de redes 5G.

Sin embargo, sus autores han decidido partir de su desarrollo e incluir la instrucción AES-enc (con una única iteración del cifrado AES) para definir una nueva versión más rápida, que soporte claves de 256 bits.

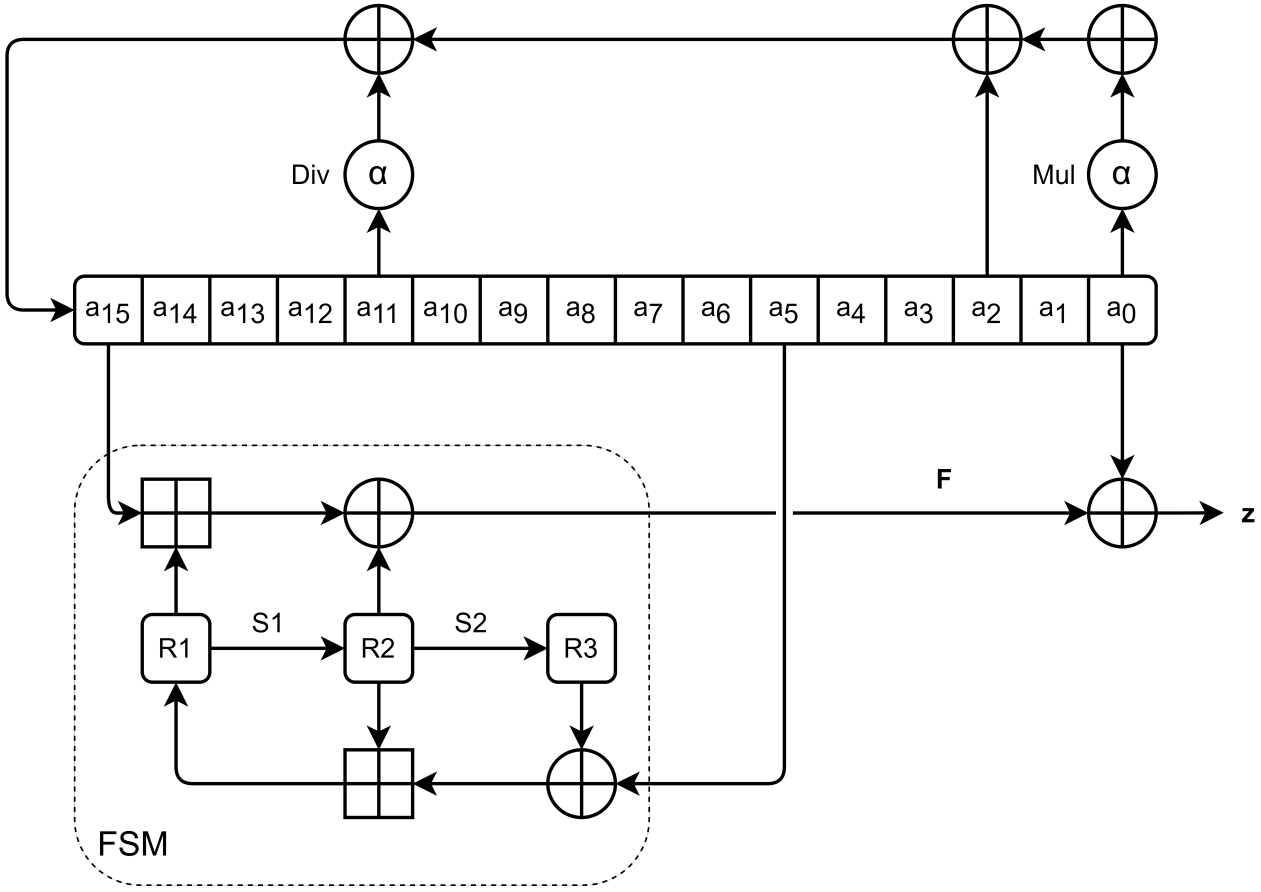

Figura 2.1: Esquema de SNOW 3G

Esta nueva versión, publicada en el año 2018 como propuesta para el nuevo estándar 5G, fue denominada SNOW-V debido a la necesidad de que el cifrado fuera implementado de forma totalmente 'virtualizada' (11). Este cifrado introducía múltiples cambios en el funcionamiento del LFSR y la FSM que lograban aumentar su seguridad al implementar la clave de 256 bits y usar la instrucción AES-enc como parte de su FSM. Esto aumentaba significativamente la velocidad, logrando alcanzar cuotas de cifrado de 58 Gbps, y superando así fácilmente la barrera de 20 Gbps impuesta por el 3GPP, pero quedándose

lejos de los 100 Gbps que se proponen para el 6G.

Uno de los puntos fuertes de este cifrado viene dado debido a la gran cantidad de ataques que se han intentado realizar a sus predecesores y sobre los que se basaron a la hora de desarrollar esta nueva versión. Hasta la actualidad, hay pocos trabajos que hayan intentado realizar un ataque con éxito contra este cifrado. Uno de ellos (17), publicado en el año 2022, demuestra un posible ataque ante una versión de este cifrado y su sucesor, consiguiendo ataques diferenciales de forma exitosa al reducir el número de rondas a 4. En la publicación original (11), se toma en cuenta esta posibilidad y se demuestra cómo aumentando el número de rondas de inicialización a 16 se eliminan las posibilidades de ataque por esta vía. Además, en ese documento también se analiza la seguridad con respecto a otros ataques de inicialización, ataques de detección de tiempo/memoria/datos, de distinción lineal, de correlación, algebraicos, y del tipo *'Guess and Determine attack'*.

Sin embargo, debido a que los dispositivos sobre los que se implementaría este cifrado en el caso de seleccionarse como estándar es realmente amplio, este cifrado acabaría llegando también a dispositivos con bajos recursos, lo que provocaría que la ejecución se ralentizase. Por ello, sus proponentes publicaron en el año 2021 una nueva versión de este cifrado, que mantenía los niveles de seguridad y se centraba en aumentar las cuotas de velocidad del cifrado.

Esa versión, a pesar de ser una simplificación de la versión anterior, logra mantener los niveles de seguridad gracias a que mantienen el mismo ciclo de generación de $2^{512} - 1$ en el LFSR. En particular, se elimina el caso de entrada vacía, ya que no generaría ningún ciclo. Esto también implica que la mayoría de intentos de ataques realizados a SNOW-V son fácilmente aplicables a su nueva versión.

Debido a la alta velocidad que alcanza este cifrado, se postula como un claro contendiente a estándar para las nuevas versiones de telefonía, tanto por nivel de seguridad, como por velocidad.

Capítulo 3

Cifrados estudiados

En este capítulo se incluyen las implementaciones de los cifrados propuestos a estándar para las nuevas generaciones de telefonía móvil, y se explican las diferentes estructuras y operaciones que los componen.

Para poder estudiar estos cifrados, teniendo en cuenta los entornos sobre los que se van a ejecutar, se ha realizado la implementación en plataformas ARM y x86. De esta forma se logra abarcar la mayoría de dispositivos que se encuentran en el mercado actual, obteniéndose una mayor comprensión del funcionamiento de dichos cifrados. Otro aspecto a detallar es que de cada familia de cifrados se ha llevado a cabo la implementación de la última versión, asegurando así que se implementa la versión más segura y veloz de cada uno de los cifrados contemplados. Estas implementaciones se pueden encontrar en el repositorio adjunto a esta memoria como Apéndice A.4.

3.1. AES

El algoritmo AES, en todas sus versiones, es un esquema de cifrado en bloque basado en una red de sustitución-permutación. En su versión de 256 bits se compone de un bloque fijo de este tamaño, que es operado en forma de matriz 4x4 llamada estado (*state*).

Esta red de sustitución-permutación de bloques se basa en cuatro operaciones que se ejecutan a lo largo de una serie de rondas para obtener la salida deseada sobre el estado original. Estas operaciones son:

- *SubBytes*: Sustitución no lineal donde cada byte se reemplaza siguiendo una tabla predefinida.

- *ShiftRows*: Se transpone cada fila de manera cíclica de la siguiente manera. La primera fila no se modifica. La segunda fila traslada cada celda una posición a la izquierda. La tercera fila se traslada dos celdas y finalmente, la tercera se traslada tres celdas en esta dirección. Estos traslados se realizan de forma circular, es decir, si la primera celda de una fila se desplaza una posición, pasa a situarse en la última celda de dicha fila.

- *MixColumns*: Mezcla de columnas en donde los cuatro registros de cada columna se combinan siguiendo una transformación lineal, definida como la multiplicación que se muestra en la Fig. 3.1.

- *AddRoundKey*: Cada registro del estado se opera con la subclave correspondiente a dicha ronda, mediante la puerta lógica XOR, mezclando así el estado con la subclave.

Las subclaves son generadas previamente, y se generan tantas como rondas se vayan a ejecutar. Estas subclaves se generan siguiendo el esquema de generación de subclaves que se puede observar en el documento en el que se detallan todos los aspectos de este

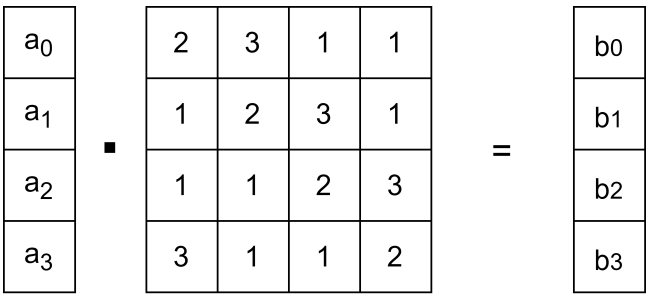

Figura 3.1: Esquema de MixColumns

cifrado (2).

En cuanto a la implementación, existe un documento publicado por la compañía INTEL, en el que se detalla una implementación aprovechando los sets de instrucciones incluidos en los procesadores x86 (18).

Por otra parte, para la implementación en las plataformas ARM no existe ninguna documentación oficial que seguir para una implementación real. Sin embargo, todas las operaciones necesarias para la implementación vienen incluidas dentro del set de instrucciones de ARM, y, por tanto, la implementación es relativamente directa siguiendo la publicación de AES y trasladando las operaciones referidas por las operaciones incluidas en el set.

En la actualidad, y gracias a la amplia implementación en hardware que se ha realizado de este cifrado, es posible alcanzar tasas de velocidad de 35 Gbps. Además, este cifrado, que está implementado en una gran cantidad de sistemas y tecnologías, a día de hoy se sigue considerando totalmente seguro.

3.2. ZUC

El algoritmo ZUC es un cifrado en flujo compuesto por un LFSR de 496 bits como generador principal, una capa de reorganización de bits (BR, *'Bit Reorganization'*) y una máquina de estado finito o FSM.

El LFSR se comporta de forma particular, ya que, a diferencia de los demás cifrados que solemos encontrar con un LFSR como base, este LFSR se compone de un estado con 16 etapas, cada una conteniendo 32 bits. Por su parte, la capa de reorganización toma en cada ronda de cifrado el contenido de este LFSR y conforma cuatro palabras de 32 bits denominadas X_0, X_1, X_2 y X_3. Estos registros son sobre los que se opera en la máquina de estado finito junto con las palabras almacenadas en R1 y R2, dos registros de 32 bits que se mantienen dentro de la FSM y que se generan en la ronda anterior. De esta forma, la salida que ofrece la FSM es la secuencia pseudoaleatoria de bits con la que se cifra el texto original. Un esquema de este flujo se puede observar en la Fig. 3.2.

El cifrado ZUC opera sobre sí mismo, realizando una suma entre la salida del módulo y la salida del cifrado, durante las rondas de inicialización, para poder garantizar una correcta inicialización y un nivel suficiente de seguridad.

En el caso de la implementación de este cifrado aún no ha habido una propuesta oficial por parte de los investigadores en la que se presente una implementación haciendo uso de las diferentes instrucciones SIMD o de la función AES-enc. Por ello se espera que este cifrado pueda ser mucho más veloz y seguro si se le realiza una revisión en los próximos años. La implementación realizada en este trabajo ha sido posible siguiendo

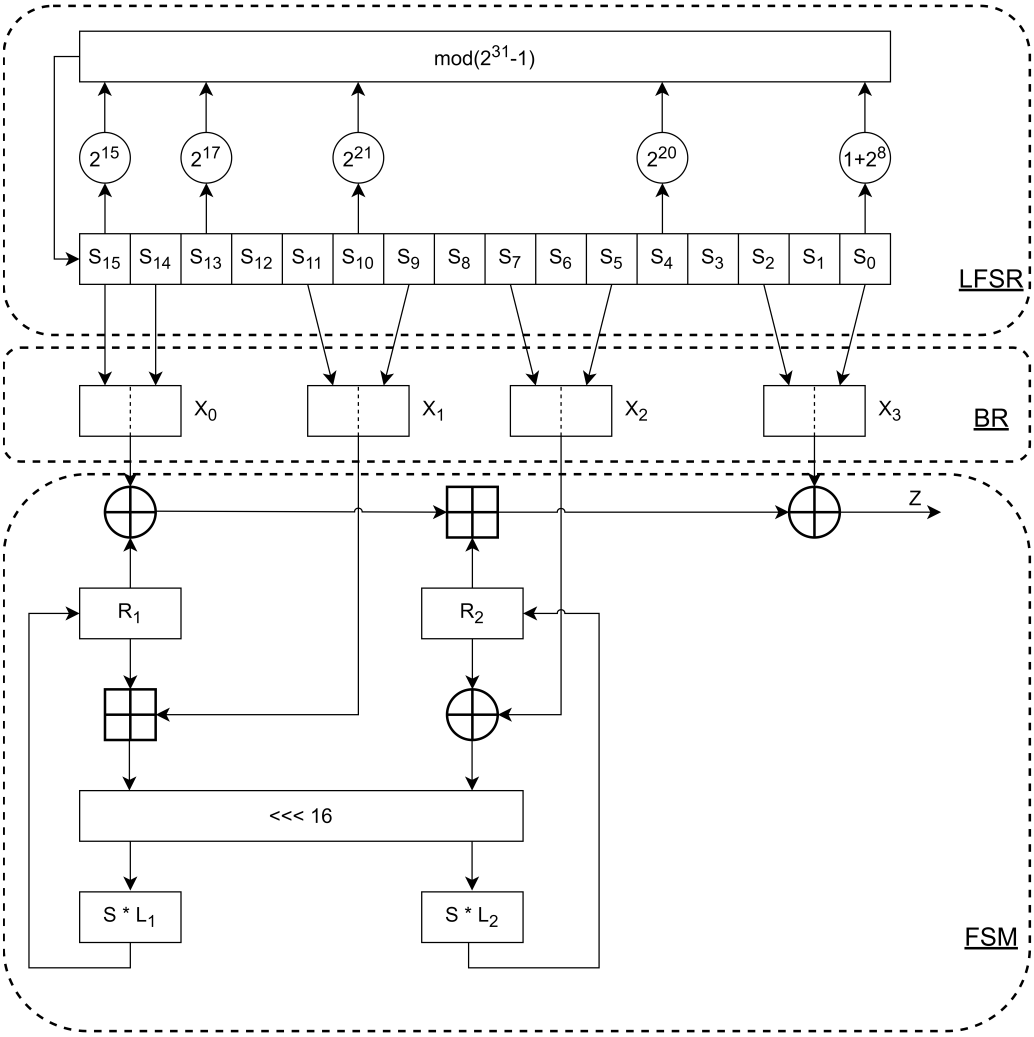

Figura 3.2: Esquema de ZUC

las implementaciones realizadas en múltiples librerías así como en las especificaciones realizadas por el organismo 3GPP (3).

3.3. Rocca

Rocca es un algoritmo de cifrado basado en el esquema AES, que hace uso de funciones SIMD y del set de instrucciones AES-NI (incluyendo AES-enc), para acelerar la ejecución y aumentar el nivel de seguridad que ofrece.

Para minimizar el impacto de estas funciones dentro del algoritmo, el estado interno se actualiza haciendo uso de dos instrucciones, AES-enc o XOR, dependiendo del registro sobre el que se opere.

Este estado interno se actualiza siguiendo el esquema que se puede observar en la expresión 3.1, donde se puede apreciar qué registros operan sobre una función AES-enc,

y cuáles sobre una función XOR.

$$\begin{aligned} S_0^{new} &= S_7 \oplus X_0 \\ S_1^{new} &= AES(S_0, S_7) \\ S_2^{new} &= S_1 \oplus S_6 \\ S_3^{new} &= AES(S_2, S_1) \\ S_4^{new} &= S_3 \oplus X_1 \\ S_5^{new} &= AES(S_4, S_3) \\ S_6^{new} &= AES(S_5, S_4) \\ S_7^{new} &= S_0 \oplus S_6 \end{aligned} \tag{3.1}$$

Las operaciones indicadas en la expresión 3.1 permiten refrescar el estado en cada ronda. Sin embargo, a la hora de realizar el cifrado, al texto original se le aplica de nuevo un conjunto de operaciones AES y XOR en conjunto a los estados que terminan generando la salida cifrada. Esta función se puede definir siguiendo el Algoritmo 1.

Algoritmo 1 Rocca - Función de cifrado

for i = 0; i <m **do**
  C_i^0 = AES(S_1, S_5) $\oplus$M$_i^0$
  C_i^1 = AES(S_0 $\oplus$S$_4$, S_2) $\oplus$M$_i^1$
  $R(S, M_i^0, M_i^1)$
end for

En el Algoritmo 1, C_Y^X equivale a la salida del texto cifrado en la iteración X, tomando la parte Y, pudiendo ser Y la parte alta o baja del registro. AES hace referencia a la operación AES-enc. S hace referencia al estado interno del cifrado y M hace referencia al texto original de entrada. Finalmente, la función de la línea 4, R, referencia a la función de actualización del estado interno del cifrado.

Este cifrado tomó como punto de referencia el cifrado que comentaremos posteriormente, SNOW-V. Sin embargo, para poder alcanzar el objetivo de 100 Gbps que pretendía conseguirse, habría que conseguir multiplicar por al menos tres veces la velocidad de este cifrado. Por eso se decidió seguir otra estrategia basada en el uso de funciones AES-enc, y no en el uso de un LFSR o una FSM. Sin embargo, a la finalización del presente trabajo se ha podido demostrar que partiendo de un cifrado como SNOW-Vi, que no deja de ser una versión mejorada de la versión V, y aplicando una implementación lo suficientemente veloz, se ha podido alcanzar el objetivo que se había planteado con la propuesta de Rocca.

La velocidad que se logra con este cifrado viene en gran parte dada gracias a la posibilidad de paralelizar el uso de funciones de AES-enc. Por aclarar mejor este punto, en la arquitectura de Intel Ice-Lake, la latencia de la función AES-enc es de tres ciclos, mientras que la plataforma permitía llamar a dos de estas funciones en cada ciclo. De esta forma, planteando de forma correcta la actualización dentro del estado interno, se pueden paralelizar las funciones AES-enc que se ejecutan internamente, y, por tanto, acelerar sustancialmente la velocidad alcanzada por el algoritmo.

3.4. SNOW

3.4.1. SNOW-V

SNOW-V es la versión propuesta para el estándar 5G por los mismos investigadores del resto de la familia SNOW. Para poder cumplir con los requisitos impuestos por el 3GPP, se han utilizado claves de 256 bits, y se ha propuesto la implementación de forma totalmente virtualizada, para poder incluir este cifrado en los diferentes servidores y dispositivos que lo requieran, sin necesidad de incluir ningún elemento hardware extra. Además, en esta implementación se logran alcanzar tasas de hasta 58 Gbps, lo que supera con soltura los 20 Gbps que se piden para este nuevo estándar.

En la versión V, el LFSR y la FSM fueron remodelados totalmente para cumplir con estos propósitos de velocidad y seguridad. La nueva versión se describe en la Fig. 3.3.

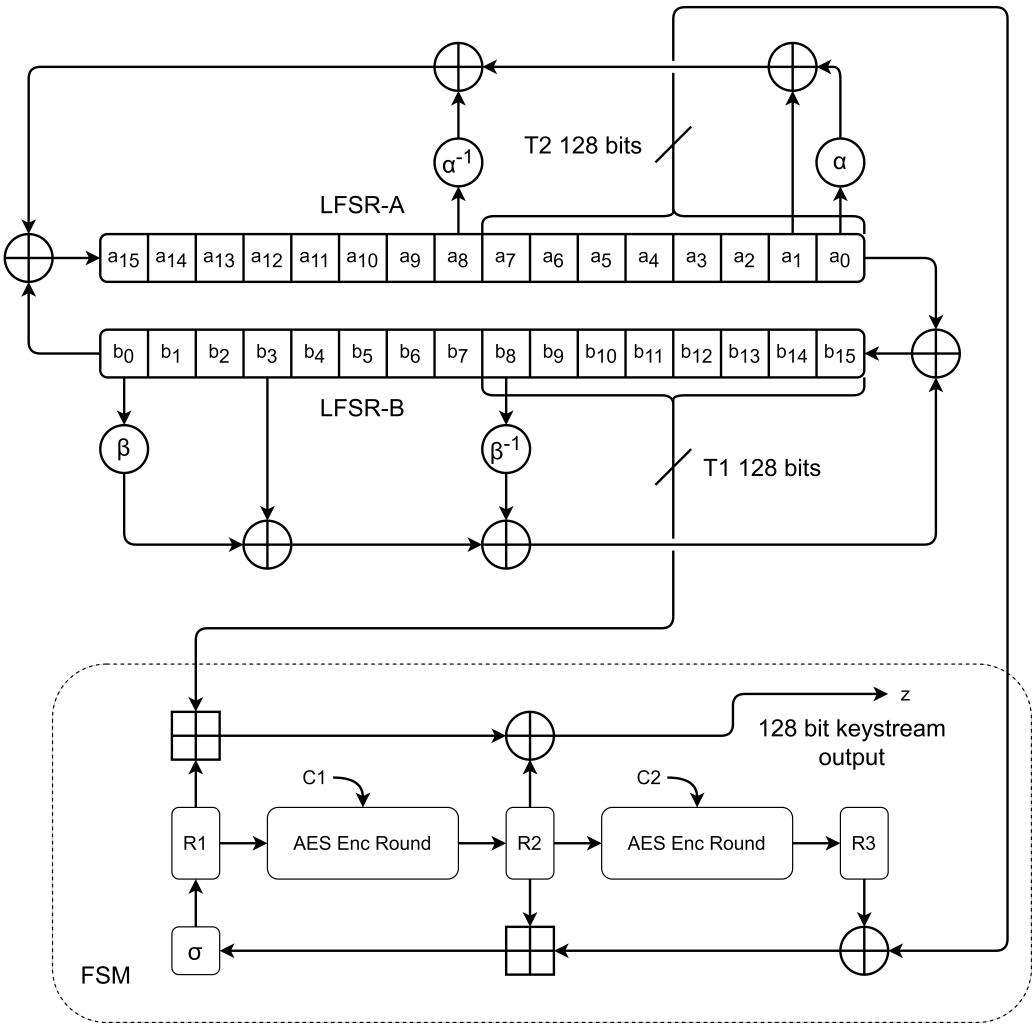

Figura 3.3: Esquema de SNOW-V

El LFSR en SNOW-V está formado por dos LFSR de 256 bits, denotados LFSR-A y LFSR-B, que se alimentan mutuamente. Cada uno de estos registros está constituido por 16 celdas de 16 bits. Estos elementos en ambos LFSR son generados de acuerdo a los polinomios definidos en la *Ec. (3.2)*:

$$\begin{aligned} g^A(x) &= x^{16} + x^{15} + x^{12} + x^{11} + x^8 + x^3 + x^2 + x + 1 \\ g^B(x) &= x^{16} + x^{15} + x^{14} + x^{11} + x^8 + x^6 + x^5 + x + 1 \end{aligned} \tag{3.2}$$

Los estados de estos LFSR se desplazan en cada golpe de reloj donde en $a_x^{(t)}$, x denota la posición y t el momento de reloj en el que se encuentra. Los estados en la posición 15 son los que se generan en cada golpe de reloj, y con el desplazamiento, los que se encuentran en la posición 0 denotan la salida de los registros.

La expresión que describe la actualización de los registros LFSR-A y LFSR-B tras 16 golpes de reloj (lo necesario para refrescar T1 y T2), es la que se encuentra en la *Ec. (3.3)*:

$$\begin{aligned} a^{(t+16)} &= b^{(t)} + \alpha a^{(t)} + a^{(t+1)} + \alpha^{-1} a^{(t+8)} \mod g^A(\alpha) \\ b^{(t+16)} &= a^{(t)} + \beta b^{(t)} + b^{(t+3)} + \beta^{-1} b^{(t+8)} \mod g^B(\beta) \end{aligned} \tag{3.3}$$

donde α y β son las raíces de $g^A(x)$ y $g^B(x)$ respectivamente, y α^{-1} y β^{-1} son sus respectivas inversas.

Gracias a estos cálculos, podemos realizar 16 golpes de reloj de forma simultánea, gracias a las instrucciones de los sets que ofrecen los diferentes procesadores.

Con cada actualización del LFSR se dan 16 golpes de reloj con el fin de actualizar completamente T1 y T2, conformados por los registros (b_{15}, ..., b_8) y (a_7, ..., a_0) respectivamente, formando así dos palabras de 128 bits que servirán de entrada para la FSM.

Por su parte, la FSM genera una salida de 128 bits y está compuesta por tres registros de 128 bits, R1, R2 y R3. Estos se combinan de la forma descrita en la *Ec. (3.4)* para generar la salida pertinente:

$$z^{(t)} = (R1^{(t)} \boxplus_{32} T1^{(t)}) \oplus R2^{(t)} \tag{3.4}$$

mientras que los registros que componen la FSM se actualizan según la *Ec. (3.5)*:

$$\begin{aligned} R1^{(t+1)} &= \sigma(R2^{(t)} \boxplus_{32} (R3^{(t)} \oplus T2^{(t)})) \\ R2^{(t+1)} &= AES(R1^{(t)}, C1) \\ R3^{(t+1)} &= AES(R2^{(t)}, C2) \end{aligned} \tag{3.5}$$

donde $AES()$ es la instrucción que define una ronda de AES con la clave establecida en cero (C1 = C2 = 0). La operación $\boxplus_{32}$ son 4 sumas aritméticas paralelas de 32 bits. σ describe una permutación a escala de bit donde se sigue la matriz de estado de AES 4x4, tal como se muestra en la *Ec. (3.6)*:

$$\sigma = [0, 4, 8, 12, 1, 5, 9, 13, 2, 6, 10, 14, 3, 7, 11, 15] \tag{3.6}$$

Estas operaciones describen la ejecución del cifrado SNOW-V. Sin embargo, en dispositivos con bajos recursos, estas operaciones no ofrecen el rendimiento esperado. Esto se solventa en la última versión publicada de este cifrado, SNOW-Vi, gracias a un rediseño del LFSR.

3.4.2. SNOW-Vi

El diseño de SNOW-Vi busca modificar el LFSR de su antecesor con el fin de acelerar drásticamente el rendimiento del cifrado. La modificación se basa en una simplificación de las operaciones que producen la entrada del LFSR y en el uso de los 8 registros más

altos del LFSR-A en lugar de los 8 más bajos en cuanto a la salida que producen. Por su parte, la FSM se mantiene exactamente igual que en la versión anterior. Este nuevo esquema se puede observar en la Fig. 3.4.

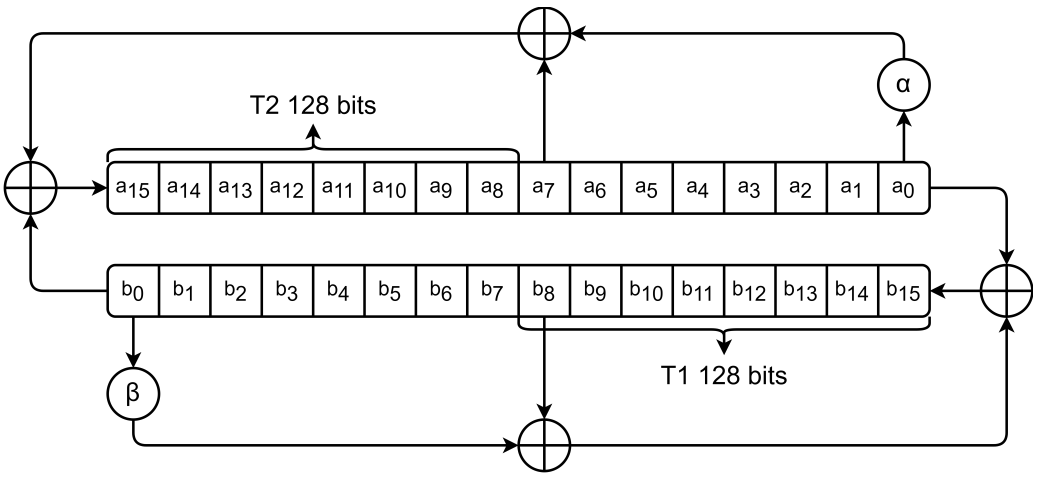

Figura 3.4: Esquema del LFSR de SNOW-Vi

Con este cambio, los polinomios que definen α y β se definen en la *Ec. (3.7)*:

$$\begin{aligned} g^A(x) &= x^{16} + x^{14} + x^{11} + x^9 + x^6 + x^5 + x^3 + x^2 + 1 \\ g^B(x) &= x^{16} + x^{15} + x^{14} + x^{11} + x^{10} + x^7 + x^2 + x + 1 \end{aligned} \tag{3.7}$$

Por otra parte, los nuevos LFSR mostrados en la Fig. 3.4 se actualizan según la *Ec. (3.8)*:

$$\begin{aligned} a^{(t+16)} &= b^{(t)} + \alpha a^{(t)} + a^{(t+7)} \mod g^A(\alpha) \\ b^{(t+16)} &= a^{(t)} + \beta b^{(t)} + b^{(t+8)} \mod g^B(\beta) \end{aligned} \tag{3.8}$$

Este nuevo diseño logra su cometido y obtiene una mejora en rendimiento de hasta el 50 % sin comprometer la longitud del ciclo que se alcanzaba en la versión anterior de $2^{512} - 1$ (excluyendo el estado cero) o el nivel de seguridad, como demuestran en la publicación original.

Capítulo 4

Investigación sobre el cifrado SNOW-Vi

Tras implementar todos los cifrados descritos en el capítulo anterior, se propuso investigar a fondo uno de ellos. El cifrado escogido es SNOW-Vi, la última propuesta de la familia SNOW. Este cifrado ofrece una tasa de velocidad realmente alta, de 92 Gbps, que supera a todos los demás cifrados salvo Rocca, que alcanza los 160 Gbps. Además, este cifrado ofrece unos niveles de seguridad realmente prometedores, junto a una mayor cantidad de referencias de investigación que avalan su nivel de seguridad. Todo esto lo avala como gran candidato a establecerse como estándar en los próximos años para los nuevos sistemas de telefonía móvil de quinta generación.

4.1. Propuesta de mejora

El cifrado SNOW-Vi, a pesar de que cumple los requisitos para la tecnología 5G, no cumple los requisitos para el futuro estándar 6G debido principalmente a la restricción en velocidad, que, aunque supera con amplia soltura los 20 Gbps establecidos en el 5G, no logra alcanzar los 100 Gbps que se esperan para la sexta generación. En el presente capítulo se realiza una propuesta de mejora en la implementación que ofrece unas tasas de mejora en velocidad entre el 14,6 % y el 34,6 %, dependiendo de la plataforma sobre la que se implemente. Esta mejora llevaría la tasa de velocidad del cifrado hasta los ∼112,5 Gbps, lo que avalaría a la nueva versión como alternativa prometedora de cara al presente y futuro de las redes de telefonía móvil 6G.

La idea de la propuesta que se presenta a continuación radica en el uso del concepto de ventanas deslizantes (19) de los vectores que componen el LFSR original. Esto permite reducir sustancialmente los tiempos de CPU que consume esta estructura, lo que deriva en una reducción del número de instrucciones general y, por consiguiente, en un aumento de la velocidad del algoritmo generador.

La técnica de la ventana deslizante se basa en la utilización de una subcadena, conocida como ventana, dentro de una cadena principal. Esa ventana se desliza de manera iterativa a través de los registros de la cadena principal. Esta dinámica se puede visualizar fácilmente en la Fig. 4.1. En el contexto de la propuesta en cuestión, la cadena principal está compuesta por tres vectores de 128 bits cada uno. La ventana, por otro lado, está conformada por los dos vectores que componen el LFSR. Con esta técnica se logra evitar el uso de copias temporales para la generación del LFSR, lo que se traduce en una mejora significativa del rendimiento global. Al deslizar la ventana sobre los registros de la cadena principal, se evita la necesidad de almacenar copias temporales de los datos generados por el LFSR. En cambio, se aprovecha el desplazamiento de la ventana para generar

directamente los nuevos valores requeridos en cada iteración. Esta estrategia reduce la carga de memoria y disminuye los tiempos de acceso y escritura, mejorando así la eficiencia del algoritmo.

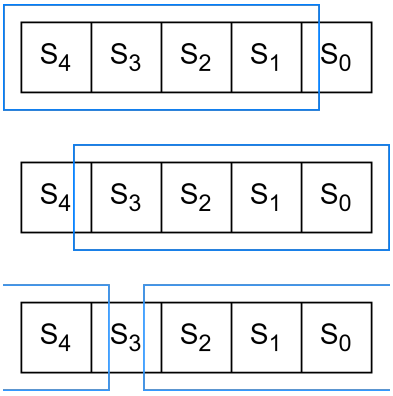

Figura 4.1: Funcionamiento de ventanas deslizantes

En la implementación software que se explicita en la publicación original de este algoritmo, se hace uso de dos registros auxiliares, T1 y T2, para almacenar los valores de B1 y A1 (los 8 registros más altos de ambos LFSR), para recuperarlos posteriormente tras operar con los vectores.

El proceso original se puede resumir como sigue. Se almacenan los valores de A1 y B1 en T2 y T1 respectivamente. Posteriormente se opera con los vectores para generar los 8 registros siguientes en A1 y B1. Finalmente, se establecen A0 y B0 a los valores de T2 y T1.

Esa forma de operar, haciendo uso de variables auxiliares, provoca cuatro copias de 128 bits por cada ronda de cifrado que se pueden eludir. La forma propuesta en este trabajo para intentar evitar esas copias temporales y acelerar drásticamente el algoritmo es hacer uso de tres vectores (ventanas) de 128 bits. Con estas ventanas se pueden utilizar dos de estos vectores para conformar el LFSR, y utilizar el vector sobrante para generar la salida que se necesita en la siguiente ronda. Luego, se utiliza el vector generado en la ronda anterior como parte alta del LFSR y el vector que se sitúa en esta posición pasa a la parte baja. Esto deja el vector que ocupaba la parte baja del vector sin usar para poder ser utilizado como salida para la siguiente ronda. Estos cambios se hacen de forma virtual, a nivel meramente conceptual, ya que en el ámbito del código, simplemente se cambia el orden de las variables, eliminando así estas copias sobre un registro temporal y la estructura LFSR, lo que conlleva una aceleración general del cifrado.

Como se puede ver en la Fig. 4.2, la iteración 0 usa los vectores A0 y A1 para generar la salida sobre A2. En la siguiente iteración, el vector A1 pasa a ocupar la primera parte del LFSR, A2 la segunda parte, y A0 se usa para almacenar la actualización del LFSR que se requiere para la siguiente iteración.

Este mismo proceso que se aplica con el LFSR-A se aplica sobre el LFSR-B, generando el proceso de ventanas deslizantes en los vectores que se usan en cada iteración y consiguiendo así una mejora sustancial en cuanto a la velocidad de ejecución del algoritmo.

Sin embargo, esos cambios producen un problema. En cada iteración del algoritmo, se depende de la anterior iteración para conocer qué vectores se han de colocar en cada posición, pero teniendo en cuenta que este conjunto de iteraciones se repite en ciclos de tres, es posible usar funciones que ejecuten tres rondas de forma consecutiva y solo

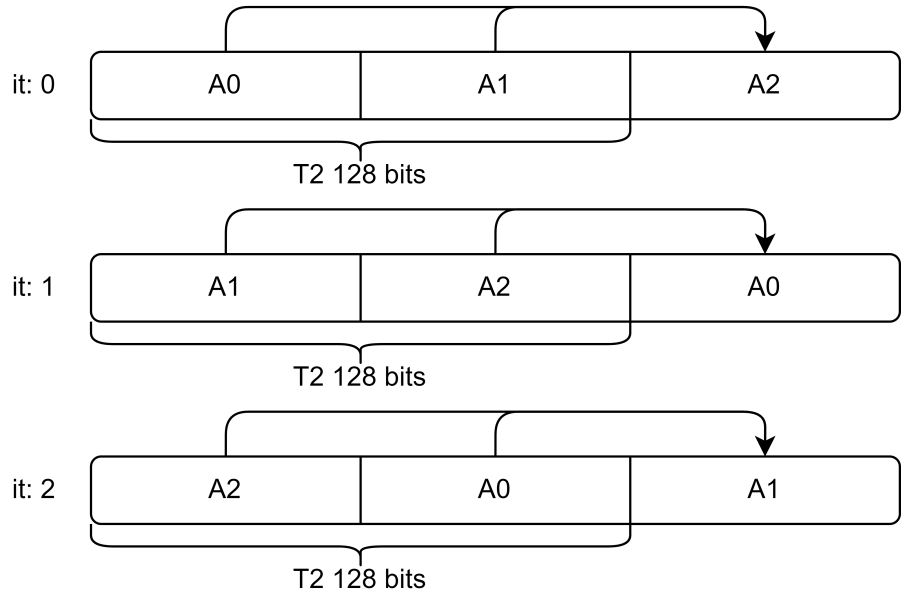

Figura 4.2: Esquema de ventanas en LFSR

ejecutar un control de estas iteraciones en la etapa final del cifrado. En la finalización del cifrado, podemos aplicar tres rondas si la longitud de la entrada es divisible por 48. En caso de que no lo sea, se comprueba la longitud restante por cifrar al finalizar las rondas triples. Si esta longitud es divisible por 32, se deben aplicar solo dos rondas en lugar de tres para finalizar, y si es solo divisible por 16, se ha de realizar una única ronda individual al final del cifrado.

Otro aspecto que cambia dentro de la implementación es la inicialización. Según la implementación original, se ejecutan 16 rondas para inicializar el proceso, y en las últimas dos rondas se aplica una operación XOR extra en el registro R1 del FSM.

La forma de solucionar esto en la nueva implementación es creando funciones que ejecuten rondas específicas para esta inicialización. En estas funciones se ejecutan de forma explícita las operaciones pertinentes para evitar el uso de instrucciones de control que ralenticen la ejecución.

En el Algoritmo 2 se muestra el flujo de ejecución de la propuesta, aplicando las funciones creadas para poder aplicar esta nueva implementación.

En la carga de claves de las líneas 3 y 4, los símbolos $\lfloor key \rfloor$ y $\lceil key \rceil$ hacen referencia a la parte baja o alta de la clave, respectivamente.

4.2. Implementaciones

En esta sección se presenta el código implementado junto con una descripción detallada de cada parte en particular. La implementación es analizada y explicada de forma exhaustiva, identificando las diferencias con respecto a la versión original. Además, se explican los cambios implementados y se justifica la necesidad de realizar dichas modificaciones.

En el *Listado A.1* se muestran las instrucciones SIMD que se utilizarán para la implementación del código. En caso de querer llevar este código a un sistema de la familia NEON-ARM, en lugar de importar la librería de intrínsecos de x86, se importaría la librería $< arm\_neon.h >$ y se buscaría una transformación directa de las instrucciones del set x86 en el set de instrucciones NEON. Otra opción podría ser el uso de la librería

Algoritmo 2 SNOW-Vi - Propuesta de implementación

$B0 \leftarrow R1 \leftarrow R2 \leftarrow 0$
$A0 \leftarrow iv$
$R3 \leftarrow A1 \leftarrow \lfloor key \rfloor$
$B1 \leftarrow \lceil key \rceil$
16 Rounds init
$i = 0$
for $i <= length - 48; i+ = 48$ **do**
    3 Rounds
end for
if $i <= length - 32$ **then**
    2 Rounds
end if
if $i <= length - 16$ **then**
    1 Round
end if

$< sse2neon.h >$, que incluye una serie de traducciones directas de las instrucciones sse a las instrucciones $neon$. Sin embargo, en este trabajo se ha optado por la primera opción, ya que permite evitar ciertas conversiones que realiza la librería debido a una cuestión de compatibilidad. Estas traducciones se pueden encontrar en el *Listado A.2*

Entrando en detalle en la implementación del algoritmo, en el *Listado A.3* se muestran las declaraciones necesarias para ejecutar rondas de tres en tres en la inicialización, y para ejecutar las últimas 4 rondas finales de la inicialización. En las rondas triples se puede observar que realizan de la forma en que se muestra en la Fig. 4.2, comenzando en la primera ronda por la salida que producen A0, A1, B0 y B1, y cargando la generación en A2 y B2. La segunda ronda carga la generación en A0 y B0, mientras que la tercera lo hace en A1 y B1. De esta forma podemos repetir este ciclo manteniendo el mismo orden, pero como en la inicialización se producen 16 rondas (un número de rondas no múltiplo de 3) y en las dos últimas rondas se ejecuta una instrucción extra, se tomó la decisión de crear otra declaración que abarcara las últimas cuatro rondas de la inicialización en su conjunto de forma explícita.

Estas cuatro rondas serán idénticas a las anteriores, con la salvedad de la instrucción extra en las dos últimas rondas y que al término de esta inicialización, el orden de A y B no es el mismo que al principio, lo que provoca que, para el bucle de cifrado, tengamos que declarar otras tres rondas con un orden diferente a las que declaramos previamente.

Las instrucciones extra que se nombraron en el párrafo anterior constituyen, simplemente, la carga de la clave en el registro R1 mediante una operación XOR.

En cuanto a las rondas de cifrado, nos encontraremos con tres tipos.

El primer tipo de ronda (*Listado 4.1*) es la que ejecuta el cifrado en rondas triples, partiendo en la ronda siguiente a la inicialización. Estas rondas, al ser ejecutadas de tres en tres, han que tener un offset que se pasa por parámetro, y en cada ronda de la declaración, a este offset se le añaden 16 bytes. De esta forma nos aseguramos que cada ronda se ejecuta sobre el conjunto de bytes adecuado.

El segundo tipo de rondas (*Listado 4.1*) refiere a la finalización del cifrado en el caso en que la longitud del texto original no sea múltiplo de 48, es decir, si no es posible finalizar el cifrado con rondas triples, sino que sobran 32 bytes al final de las rondas triples que

ejecutamos previamente. En este caso, solo se ejecutarán dos rondas. Como aquí se puede determinar cuál es la última ronda que se va a ejecutar, es posible obviar las instrucciones que establecen el estado para la siguiente ronda, ya que esta no existirá. De esta forma, evitamos ejecutar instrucciones innecesarias.

El tercer tipo de ronda (*Listado 4.2*) se refiere a un caso similar al anterior, pero en el que el tamaño por cifrar es, únicamente, 16 bytes en lugar de 32, y en este caso solo es necesario ejecutar la operación *'STORE'*, debido a que los parámetros que utiliza no se actualizan en la misma ronda y no hace falta preparar los registros para la ronda siguiente.

```
#define SNOWVi_3ROUNDS(offset)                    \
A0 = XOR(XOR(XOR(TAP7(A2, A1), B1),               \
     AND(SRA(A1),SET(0x4a6d))), SLL(A1));         \
B0 = XOR(XOR(B2, AND(SRA(B1),                     \
     SET(0xcc87))), XOR(A1, SLL(B1)));            \
STORE(out + offset, XOR(ADD(B2, R1),              \
     XOR(LOAD(in + offset), R2)));                \
A1 = ADD(R2, R3);                                 \
R3 = AESR(R2, A0);                                \
R2 = AESR(R1, ZERO());                            \
R1 = SIGMA(A1);                                   \
A1 = XOR(XOR(XOR(TAP7(A0, A2), B2),               \
     AND(SRA(A2),SET(0x4a6d))), SLL(A2));         \
B1 = XOR(XOR(B0, AND(SRA(B2),                     \
     SET(0xcc87))), XOR(A2, SLL(B2)));            \
STORE(out + offset + 16, XOR(ADD(B0, R1),         \
     XOR(LOAD(in + offset + 16), R2)));           \
A2 = ADD(R2, R3);                                 \
R3 = AESR(R2, A1);                                \
R2 = AESR(R1, ZERO());                            \
R1 = SIGMA(A2);                                   \
A2 = XOR(XOR(XOR(TAP7(A1, A0), B0),               \
     AND(SRA(A0),SET(0x4a6d))), SLL(A0));         \
B2 = XOR(XOR(B1, AND(SRA(B0),                     \
     SET(0xcc87))), XOR(A0, SLL(B0)));            \
STORE(out + offset + 32, XOR(ADD(B1, R1),         \
     XOR(LOAD(in + offset + 32), R2)));           \
A0 = ADD(R2, R3);                                 \
R3 = AESR(R2, A2);                                \
R2 = AESR(R1, ZERO());                            \
R1 = SIGMA(A0);                                   \

#define SNOWVi_2ROUNDS(offset)                    \
A0 = XOR(XOR(XOR(TAP7(A2, A1), B1),               \
     AND(SRA(A1),SET(0x4a6d))), SLL(A1));         \
B0 = XOR(XOR(B2, AND(SRA(B1),                     \
     SET(0xcc87))), XOR(A1, SLL(B1)));            \
STORE(out + offset, XOR(ADD(B2, R1),              \
     XOR(LOAD(in + offset), R2)));                \
A1 = ADD(R2, R3);                                 \
R2 = AESR(R1, ZERO());                            \
R1 = SIGMA(A1);                                   \
STORE(out + offset + 16, XOR(ADD(B0, R1),         \
     XOR(LOAD(in + offset + 16), R2)));           \
```

Listado 4.1: Rondas de cifrado

Por último, en el *Listado 4.2* se muestra la función principal de cifrado, donde se ejecutarán tanto las operaciones de inicialización como las rondas necesarias para el cifrado del texto original que se han ido detallando previamente. Esta función recibe como parámetros la longitud total del texto original a cifrar, un registro donde almacenar la salida, un registro que contiene la entrada, un registro que contiene la clave y un último registro que contiene el vector de inicialización 'iv'.

```
static inline void SNOWVi_improved
  (int length, u8 * out, u8 * in, u8 * key, u8 * iv)
{ __m128i A0, A1, A2, B0, B1, B2, R1, R2, R3;
  B0 = R1 = R2 = ZERO();
  A0 = LOAD(iv);
  R3 = A1 = LOAD(key);
  B1 = LOAD(key + 16);

  // Initial rounds
  SNOWVi_3ROUNDS_INIT;
  SNOWVi_3ROUNDS_INIT;
  SNOWVi_3ROUNDS_INIT;
  SNOWVi_3ROUNDS_INIT;
  SNOWVi_4ROUNDS_INIT;

  // Main loop
  int i = 0;
  for (; i <= length - 48; i += 48)
  {
    SNOWVi_3ROUNDS(i);
  }
  if (i <= length - 32)
  {
    SNOWVi_2ROUNDS(i);
    return;
  }
  if (i <= length - 16)
  {
    STORE(out + i, XOR(ADD(B2, R1),              \
        XOR(LOAD(in + i), R2)));
  }
}
```

Listado 4.2: Función principal de encriptado

Para finalizar, destacar que este tipo de rondas tan explícitas, en el cifrado original, no existen, ya que únicamente existe un tipo de ronda que posee diferentes instrucciones de control que determinan el tipo de ronda o el caso en el que se encuentra. Gracias a estas rondas, que se realizan de forma explícita, se puede reducir aún más el número de instrucciones a procesar y, por tanto, se reduce el tiempo de ejecución del algoritmo

4.3. Análisis de resultados

Se presentan a continuación las diferentes pruebas que se llevaron a cabo en diversos entornos para evaluar tanto la versión original como la propuesta. A través de estas pruebas se ha podido observar la discrepancia existente entre ambas implementaciones,

lo que permite evaluar su rendimiento en distintas condiciones para concluir en qué casos presentan alguna ventaja.

4.3.1. Entorno de pruebas

Para la medición se ha hecho uso de la función $clock()$ de $< time.h >$ que permite medir el número de instrucciones de CPU utilizado en cada implementación. Cabe destacar que, dado que se ha utilizado un entorno diferente al utilizado en la publicación original de SNOW-Vi, las diferencias se calculan dividiendo el número de instrucciones de la implementación propuesta entre el número de instrucciones de la implementación original para mostrar la diferencia porcentual.

En cuanto a la longitud del texto original utilizado, se emplean tanto los tamaños presentados en la implementación original, como tamaños más grandes para comprobar el funcionamiento con entradas largas. Los tamaños utilizados en la publicación original de SNOW-Vi son 64B, 256B, 1KB, 4KB, 16KB. Con objeto de ampliar los casos de utilización, se han añadido los tamaños 64KB y 256KB.

Para intentar reducir las diferencias que puede haber de un momento a otro en un dispositivo a la hora de ejecutar los cifrados, y para reducir la variación que se puede producir con las medidas de medición de tiempo, se ejecutarán dentro de bucles 'for' de tamaño 1000000.

4.3.2. Plataformas de pruebas

Uno de los objetivos al seleccionar los entornos de pruebas para ejecutar las implementaciones es lograr una cobertura amplia de casos, por lo que se buscan entornos que permitan abarcar el mayor número de configuraciones posibles. Entre los que se han utilizado para este trabajo, se incluyen procesadores de sobremesa como Intel y AMD para la arquitectura x86 o Apple para ARM, así como procesadores ARM más parecidos a los que se encuentran en dispositivos móviles, como la familia Rockchip.

En la Tabla 4.1 se especifican las diferentes CPUs utilizadas, detallando la familia a la que pertenecen, el modelo, la frecuencia de reloj y el número de núcleos sobre el que se ejecutan las pruebas. Nótese que aunque las plataformas 6 y 7 parecen exactamente idénticas en la Tabla 4.1, se trata del mismo procesador implementado en dos dispositivos diferentes. La plataforma 6 es un dispositivo de sobremesa, mientras que la plataforma 7 es un dispositivo portátil. Esto se ha realizado así para poder obtener una base más extensa, conociendo si las capacidades de refrigeración, y, por tanto, las frecuencias de reloj, afectaban, y en qué medida, a estas ejecuciones.

El objetivo principal de usar estos entornos de prueba es poder comprobar la diferencia de rendimiento que existe entre ambas implementaciones, independientemente de la familia, velocidad del procesador o número de núcleos. Como se puede observar, se cuenta con procesadores con muchos y pocos núcleos, dispositivos de mayor y menor frecuencia de reloj y familias orientadas a dispositivos sobremesa, así como orientadas a dispositivos móviles.

En los casos en los que el número de núcleos viene especificado como 'X + Y', esto significa que no todos los núcleos de dichas plataformas son idénticos, ya que en los últimos años se ha seguido una tendencia de utilizar un número de núcleos de alto rendimiento, en conjunto con un número de núcleos de alta eficiencia. Esto se ha llevado a cabo principalmente en dispositivos que poseen unas capacidades de refrigeración

Tabla 4.1: Plataformas de ejecución

N.º	Familia	Modelo	Núcleos	Freq. GHz
1	ARM	RK3588s	4 + 4	2.4
2	ARM	RK3399	2 + 4	1.8
3	x86	Ryzen 7 1700	8	3
4	x86	i7-1185G7	4	3
5	x86	i9-139000K	8 + 16	3
6	ARM	M1 Pro	8	3.22
7	ARM	M1 Pro	8	3.22

limitadas, o en aquellos que dependen de una batería para su alimentación. En la Tabla 4.1, el primer valor hace referencia al número de núcleos de alto rendimiento, mientras que el segundo refiere al número de núcleos de alta eficiencia.

Entre estas plataformas, es importante destacar algunas de ellas por encima de las otras. En primer lugar, sobresalen las plataformas 1 y 2. Estas plataformas conforman placas SBC (Single Board Computer) y guardan una gran relación con la mayoría de dispositivos móviles del mercado, ya que cuentan con una refrigeración y una alimentación limitada, una arquitectura ARM, y un tamaño muy similar a las placas base encontradas en smartphones o tablets.

Otras plataformas a destacar son las plataformas 6 y 7. Estas plataformas se refieren a procesadores ARM fabricados por Apple. La importancia de estos procesadores radica en que su uso se está llevando desde dispositivos portátiles, sobremesa, tablets y smartphones, con lo que abarcan una gran cantidad de dispositivos y finalidades.

Por último, destacar la plataforma 5, que, aunque hace referencia a una arquitectura más clásica, basada en una plataforma x64 montada en un ordenador de sobremesa, ofrece una relevancia clara debido a la alta cantidad de núcleos que ofrece. Los servidores que soportarán las redes de telefonía poseen procesadores con grandes cantidades de núcleos, y esta plataforma es la más similar frente al resto de plataformas evaluadas.

4.3.3. Resultados obtenidos

A continuación, en la Fig. 4.3 se muestra la media de mejoras obtenidas en cada uno de los tamaños de texto original. Por otro lado, en la Fig. 4.4 se muestra la media de mejora en cada una de las plataformas evaluadas según la Tabla 4.1.

Los resultados obtenidos permiten confirmar que este cifrado mejora sustancialmente la velocidad, manteniendo las mismas estructuras internas. Además, el nivel de seguridad no se ve comprometido, mientras que su rendimiento sí se ve impulsado notablemente.

Como se puede observar, con la nueva implementación se logra una reducción del número de instrucciones clara, lo que conlleva una mejora del ∼22,41 %, visible en la Tabla 4.2, para un gran número de plataformas y entornos distintos. Mirando la cuota de cifrado alcanzada por SNOW-Vi en su implementación original, se puede observar que alcanza los 92 Gbps. Con los datos obtenidos se llega a la conclusión de que la cuota de cifrado de esta implementación se sitúa en los ∼112,5 Gbps, superando así la barrera de los 100 Gbps, y posicionándose como propuesta seria para el paradigma 6G.

Este cambio de paradigma supone una clara ventaja de SNOW-Vi frente al resto de

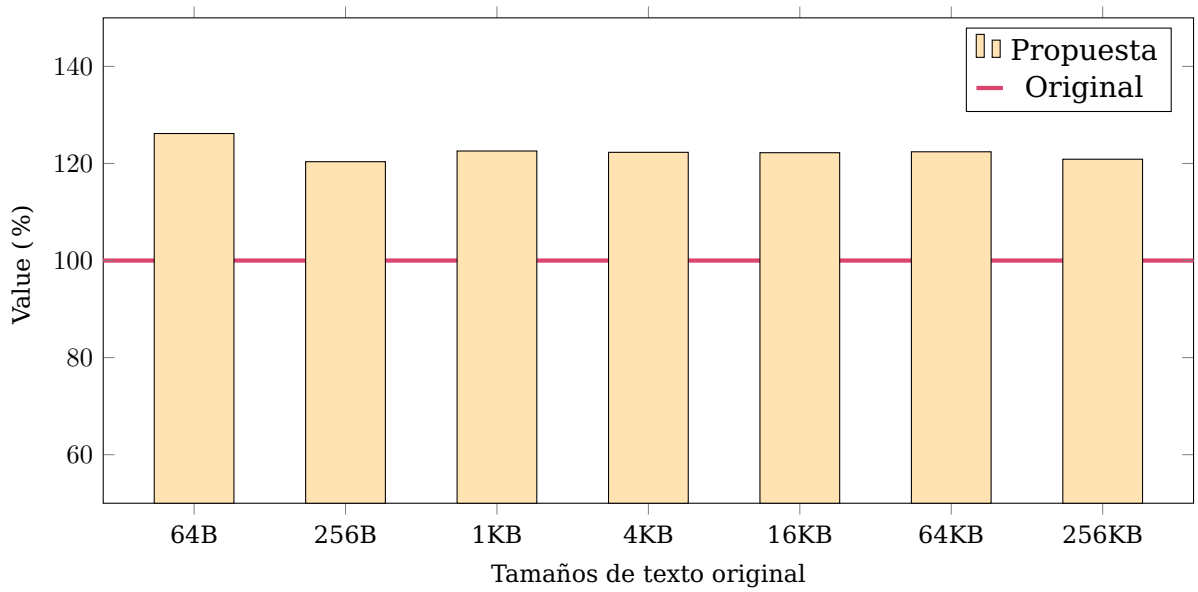

Figura 4.3: Diferencia de rendimiento por tamaños

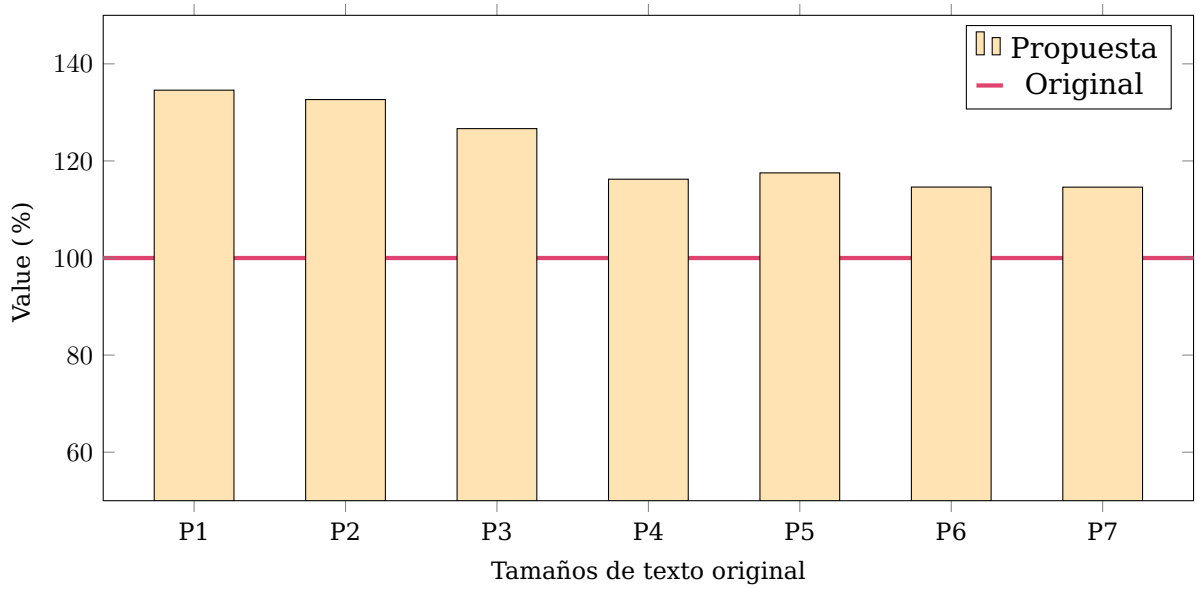

Figura 4.4: Diferencia de rendimiento por entornos

Tabla 4.2: Resultados obtenidos

P.	64B	256B	1KB	4KB	16KB	64KB	256KB
1	+58,56 %	+35,49 %	+31,53 %	+29,63 %	+29,54 %	+28,91 %	+28,43 %
2	+55,1 %	+34,39 %	+31,22 %	+27,2 %	+24,68 %	+27,92 %	+28,02 %
3	+13,75 %	+18,03 %	+29,98 %	+32,55 %	+33,84 %	+34,24 %	+24,22 %
4	+9,13 %	+10,69 %	+19,63 %	+19,1 %	+19,22 %	+18,26 %	+17,63 %
5	+7,13 %	+12,15 %	+17,45 %	+20,57 %	+21,99 %	+21,54 %	+21,94 %
6	+19,83 %	+15,9 %	+14,1 %	+13,51 %	+13,07 %	+12,98 %	+12,95 %
7	+19,67 %	+15,91 %	+14,04 %	+13,5 %	+13,17 %	+12,99 %	+12,93 %

cifrados. Por un lado, este cifrado cuenta con un gran nivel de seguridad y una amplia investigación que lo respalda. Por otro lado, gracias a este trabajo se ha confirmado que

dicho cifrado permite alcanzar cuotas de velocidad que permitirán su uso en una gran cantidad de ámbitos y dispositivos. De hecho, es posible alcanzar sus cuotas más altas en dispositivos de altas prestaciones, evitando así que los dispositivos que no dispongan de esas capacidades se vean ralentizados por el cifrado.

Independientemente de que su uso se lleve a las velocidades establecidas para el paradigma 6G, no cabe duda de que esta mejora provocará un mejor rendimiento en la mayoría de dispositivos en donde se aplique, y en el caso de los dispositivos con menos recursos, permitirá alcanzar cuotas suficientemente altas para el paradigma actual.

Destaca la gran velocidad obtenida en las plataformas 1 y 2, que poseen un procesador con una arquitectura similar a la de los dispositivos móviles. Teniendo en cuenta que este tipo de dispositivos, junto a los servidores, son las plataformas principales sobre las que se van a implementar este tipo de cifrados para aprovechar las velocidades del paradigma de las redes 5G y 6G. Esa mejora resulta de gran relevancia ya que permitirá a dispositivos de las gamas inferiores, alcanzar las cuotas necesarias para funcionar sin problemas en las redes más rápidas.

Por otro lado, entre los entornos que ofrecen una mejora menos sustancial, se sigue alcanzando una mejora de un 14,6 % de media, lo que sigue siendo un aumento en la velocidad realmente notable.

Todas las tablas de resultados obtenidas para cada una de las plataformas se pueden encontrar en el Apéndice A.2.

Si se observa atentamente la Tabla 4.2, se ve que a medida que los tamaños de texto original aumentan, cada arquitectura tiende hacia un porcentaje promedio de mejora. Esto es debido a que muchas de las instrucciones que se han eliminado en esta propuesta se encontraban en las fases de inicialización o finalización. Esta característica permite además que los dispositivos móviles, que en general tienen que cifrar textos originales de menor tamaño que los servidores, y que son los que más sufren por problemas de rendimiento debido a su bajo rendimiento en algunos casos, puedan mejorar sustancialmente sus capacidades de procesamiento y su fluidez.

Además, la mejora ofrecida en tamaños grandes de texto original son igualmente prometedores. Si se tienen en cuenta las plataformas 1 y 2 por su similitud con los dispositivos móviles, se puede apreciar una mejora de un $\sim$28 %, situándose por encima del $\sim$22,41 % en el que se encuentra la media.

En cuanto a las arquitecturas más clásicas, de las que se sirven los servidores, que son el otro gran objetivo de esta tecnología, se puede observar que la mejora alcanzada no se queda atrás, otorgando unos resultados de hasta un 21.99 % de mejora porcentual. Este sector es realmente crítico debido a la importancia de la eficiencia energética en este entorno. Si se analiza la gran cantidad de servidores que ejecutan este tipo de códigos y los requerimientos energéticos de los mismos, cualquier mejora de rendimiento producirá unos niveles de ahorro y una reducción en la energía consumida realmente significativa.

Capítulo 5

Conclusiones y líneas futuras

A lo largo del presente Trabajo de Fin de Grado, se ha realizado la implementación de todos los cifrados que actualmente se postulan como candidatos a estándar para el nuevo paradigma de telefonía móvil 5G sobre el que nos encontramos. Es más, se ha buscado ir un paso más allá y aventurarse a alcanzar los requisitos que se establecerán para la sexta generación de esa tecnología. Además de esto, se ha llevado a cabo una investigación de esos cifrados y se ha propuesto una nueva forma original de implementar el cifrado SNOW-Vi. Esta propuesta se centra principalmente en aumentar la velocidad de ejecución de la implementación para incrementar la ratio de cifrado por segundo que es posible alcanzar. Con la nueva implementación basada en la rotación de los vectores que conforman los LFSR, se logra una mejora significativa, del $\sim$22,41 % de media entre los diferentes entornos en los que se han utilizado como base para las pruebas. Entre estos entornos destaca la familia de procesadores Rockchip, basados en arquitectura ARM, en los que se obtiene un $\sim$33 % de aumento de velocidad media. Este procesador tiene una relevancia clave frente a los otros procesadores presentados, ya que son los que conforman el núcleo de dos placas SBC, y comparten una gran similitud con los dispositivos móviles, tanto en factor de forma como en las restricciones en cuanto a refrigeración. La mejora presentada se debe a la capacidad de evitar las copias temporales de registros que se ejecutaban en la implementación original, así como a la reducción en el número de operaciones al diferenciar las rondas que cumplen con ciertas condiciones. A título de ejemplo, se pueden mencionar las rondas finales, donde se han eliminado varias operaciones innecesarias, o la diferencia entre rondas de inicialización y de cifrado. Con esta mejora en la implementación y basándonos en los 92 Gbps que se logran con la implementación original, la propuesta presentada en este trabajo supera con facilidad la barrera de los 100 Gbps, alcanzando los $\sim$112,5 Gbps, y cumpliendo así con el objetivo establecido para la futura sexta generación de la telefonía móvil.

El trabajo realizado en el presente documento tiene múltiples líneas por las que continuar. En primer lugar, la base de dispositivos sobre los que se han realizado las pruebas, pues a pesar de ser bastante diversa, siempre podría completarse incluyendo un mayor número de dispositivos de diferente índole.

Otra línea interesante a seguir para el presente trabajo es la realización de una implementación acelerada del cifrado ZUC. A pesar de que este cifrado no presenta unos resultados realmente prometedores, es posible que una modificación que haga uso de las instrucciones de procesador SIMD o AES que se proponen en el resto de alternativas, pueda potenciar tanto la velocidad como la seguridad del cifrado.

Capítulo 6

Conclusions and future work

Throughout this Final Degree Project, the implementation of all the ciphers that are currently postulated as standard candidates for the new 5G mobile telephony paradigm has been done. Moreover, this work has gone one step further and venture to meet the requirements that will be established for the sixth generation of this technology. In addition to this, research on these ciphers has been carried out and a new original way of implementing SNOW-Vi encryption has been proposed. This proposal is mainly focused on increasing the execution speed of the implementation in order to increase the rate of encryption per second that can be achieved. With the new implementation based on the rotation of the vectors that make up the LFSRs, a significant improvement of $\sim$22.41 % is achieved on average between the different environments in which they have been used as a benchmark for testing. Among these environments, the Rockchip family of processors, based on ARM architecture, is the most remarkable, with an average speed increase of $\sim$33 %. This processor is particularly relevant compared to the other processors presented, as they form the core of two SBC boards, and share a strong similarity to mobile devices, both in form factor and cooling constraints. The presented improvement is due to the capacity to avoid the temporary copies of registers that were executed in the original implementation, as well as the reduction of the number of operations by differentiating the rounds that meet certain conditions. As an example, we have the final rounds, where several unnecessary operations have been eliminated, or the difference between initialization and encryption rounds. With this improvement in the implementation and based on the 92 Gbps achieved by the original implementation, the proposal presented in this work easily surpasses the 100 Gbps barrier, reaching 112.5 Gbps, and satisfying the objective established for the future sixth generation of mobile telephony.

The work carried out in this paper has multiple lines to follow. Firstly, the testing base of devices, even though it is quite diverse, could be more complete by including a larger number of devices of different types.

Another interesting line to follow for the present work is to carry out an accelerated implementation of the ZUC cipher. Even if this cipher does not show really promising results, it is possible that a modification using the SIMD or AES processor instructions proposed in the other alternatives could improve both the speed and the security of the cipher.

Capítulo 7

Presupuesto

En el presente capítulo se muestra un desglose de las horas dedicadas a cada apartado sobre el que se ha trabajado (ver Tabla 7.1), y un presupuesto basado en las horas dedicadas, con su respectivo coste, y el coste de los materiales que han sido necesarios para el desarrollo del trabajo realizado (ver Tabla 7.2).

En total se han dedicado 295 horas, repartidas entre labores de planificación, investigación, desarrollo, pruebas o redacción. Estas horas, a un coste de 30 €/h, han repercutido en un coste total de 8.850 €. Esto, sumado a los múltiples dispositivos utilizados para la realización de las pruebas y el desarrollo de las implementaciones aquí propuestas, ofrece un coste final de **17.050 €.**

Descripción	**nº horas**
Planificación del proyecto	15
Búsqueda, lectura y análisis de documentación	50
Implementación de los diferentes cifrados	40
Adaptación a ARM	30
Investigación de las implementaciones	10
Desarrollo de la propuesta	30
Ejecución y estudio de las pruebas realizadas	80
Redacción de la memoria	40
Total:	**295**

Tabla 7.1: Resumen de horas

Descripción	**Coste (€)**
Horas dedicadas (30€/h)	8.850
Orange Pi 5	120
Rock Pi 4B	130
Sobremesa 1	950
Dell XPS 9310	1400
Sobremesa 2	3150
Mac Mini	1200
MacBook Pro	1250
TOTAL	**17050**

Tabla 7.2: Presupuesto

Apéndice A

Apéndices

A.1. Artículo aceptado en congreso

Propuesta de mejora para la implementación en software del cifrado SNOW-Vi.
Gabriel Luis Freitas, Pino Caballero Gil, Jezabel Molina Gil.
VIII Jornadas Nacionales de Investigación en Ciberseguridad (JNIC).
Vigo, 21 a 23 de Junio de 2023.

Figura A.1: Jornadas Nacionales de Investigación en Ciberseguridad

A.2. Tablas de resultados

Tabla A.1: Orange Pi 5 - RK3588S

Tamaño	Porcentaje de mejora (%)
64B	58,56 %
256B	35,49 %
1KB	31,53 %
4KB	29,63 %
16KB	29,54 %
64KB	28,91 %
256KB	28,43 %
Media	34,58 %

Tabla A.2: Rock Pi 4B - RK3399

Tamaño	Porcentaje de mejora (%)
64B	55,1 %
256B	34,39 %
1KB	31,22 %
4KB	27,2 %
16KB	24,68 %
64KB	27,92 %
256KB	28,02 %
Media	32,64 %

Tabla A.3: Sobremesa 1 - Ryzen 7 1700

Tamaño	Porcentaje de mejora (%)
64B	13,75 %
256B	18,03 %
1KB	29,98 %
4KB	32,55 %
16KB	33,84 %
64KB	34,24 %
256KB	24,22 %
Media	26,66 %

Tabla A.4: Dell XPS 9310 (Portátil 1) - i7-1185G

Tamaño	Porcentaje de mejora (%)
64B	9,13 %
256B	10,69 %
1KB	19,63 %
4KB	19,1 %
16KB	19,22 %
64KB	18,26 %
256KB	17,63 %
Media	16,24 %

Tabla A.5: Sobremesa 2 - i9-13900K

Tamaño	Porcentaje de mejora (%)
64B	7,13 %
256B	12,15 %
1KB	17,45 %
4KB	20,57 %
16KB	21,99 %
64KB	21,54 %
256KB	21,94 %
Media	17,54 %

Tabla A.6: Mac Mini (Sobremesa 3) - M1 Pro

Tamaño	Porcentaje de mejora (%)
64B	19,83 %
256B	15,9 %
1KB	14,1 %
4KB	13,51 %
16KB	13,07 %
64KB	19,98 %
256KB	12,95 %
Media	14,62 %

Tabla A.7: Macbook Pro (Portátil 2) - M1 Pro

Tamaño	Porcentaje de mejora (%)
64B	19,67 %
256B	15,91 %
1KB	14,04 %
4KB	13,5 %
16KB	13,17 %
64KB	12,99 %
256KB	12,93 %
Media	14,6 %

A.3. Implementación de mejora de SNOW-Vi

```
#include "x86intrin.h"
#define XOR(a, b)     _mm_xor_si128(a, b)
#define AND(a, b)     _mm_and_si128(a, b)
#define ADD(a, b)     _mm_add_epi32(a, b)
#define SET(v)        _mm_set1_epi16((short)v)
#define SLL(a)        _mm_slli_epi16(a, 1)
#define SRA(a)        _mm_srai_epi16(a, 15)
#define TAP7(Hi, Lo)  _mm_alignr_epi8(Hi, Lo, 7 * 2)
#define SIGMA(a)      _mm_shuffle_epi8(a, _mm_set_epi64x(                          \
                            0x0f0b07030e0a0602ULL, 0x0d0905010c080400ULL));
#define AESR(a, k)    _mm_aesenc_si128(a, k)
#define ZERO()        _mm_setzero_si128()
#define LOAD(src)     _mm_loadu_si128((const __m128i*)(src))
#define STORE(dst, x) _mm_storeu_si128((__m128i*)(dst), x)
```

Listado A.1: Intrínsecos necesarios para x86

```
#include <arm_neon.h>
#define XOR(a, b) veorq_s64(a, b)
#define AND(a, b) vandq_s64(a, b)
#define ADD(a, b) vreinterpretq_s64_s32(vaddq_s32(  vreinterpretq_s32_s64(a),          \
                                                    vreinterpretq_s32_s64(b)))
#define SET(v) vreinterpretq_s64_s16(vdupq_n_s16((short)v))
#define SLL(a) vreinterpretq_s64_s16(vshlq_n_s16(vreinterpretq_s16_s64(a), 1))
#define SRA(a) vreinterpretq_s64_s16(vshrq_n_s16(vreinterpretq_s16_s64(a), 15))
#define TAP7(Hi, Lo) vreinterpretq_s64_u8(vextq_u8(vreinterpretq_u8_s64(Lo),           \
                                                   vreinterpretq_u8_s64(Hi), 14))
#define SIGMA(a)                                                                      \
  vreinterpretq_s64_s8(vqtbl1q_s8( vreinterpretq_s8_s64(a),                           \
                           vandq_u8(vreinterpretq_u8_s64(                             \
                               combine_s64(vcreate_s64(0x0d0905010c080400ULL),        \
                               vcreate_s64(0x0f0b07030e0a0602ULL))),                  \
                           vdupq_n_u8(0x8F))))
#define AESR(a, k) vreinterpretq_s64_u8(vaesmcq_u8(vaeseq_u8(vreinterpretq_u8_s64(a),  \
                                        (uint8x16_t){})) ^ vreinterpretq_u8_s64(k))
#define ZERO() vdupq_n_s64(0)
#define LOAD(src) vld1q_s64((const int64_t *)(src))
#define STORE(dst, x) vst1q_s64((int64_t *)(dst), (x))
```

Listado A.2: Intrínsecos necesarios para ARM

```
#define SNOWVi_3ROUNDS_INIT                     \
A2 = XOR(XOR(XOR(TAP7(A1, A0), B0),             \
     AND(SRA(A0),SET(0x4a6d))), SLL(A0));       \
B2 = XOR(XOR(B1, AND(SRA(B0), SET(0xcc87))),    \
     XOR(A0, SLL(B0)));                         \
A2 = XOR(A2, XOR(ADD(B1, R1), R2));             \
A0 = ADD(R2, R3);                               \
R3 = AESR(R2, A2);                              \
R2 = AESR(R1, ZERO());                          \
R1 = SIGMA(A0);                                 \
A0 = XOR(XOR(XOR(TAP7(A2, A1), B1),             \
     AND(SRA(A1),SET(0x4a6d))), SLL(A1));       \
B0 = XOR(XOR(B2, AND(SRA(B1), SET(0xcc87))),    \
```

```
     XOR(A1, SLL(B1)));                          \
A0 = XOR(A0, XOR(ADD(B2, R1), R2));              \
A1 = ADD(R2, R3);                                \
R3 = AESR(R2, A0);                               \
R2 = AESR(R1, ZERO());                           \
R1 = SIGMA(A1);                                  \
A1 = XOR(XOR(XOR(TAP7(A0, A2), B2),              \
     AND(SRA(A2),SET(0x4a6d))), SLL(A2));        \
B1 = XOR(XOR(B0, AND(SRA(B2), SET(0xcc87))),     \
     XOR(A2, SLL(B2)));                          \
A1 = XOR(A1, XOR(ADD(B0, R1), R2));              \
A2 = ADD(R2, R3);                                \
R3 = AESR(R2, A1);                               \
R2 = AESR(R1, ZERO());                           \
R1 = SIGMA(A2);

#define SNOWVi_4ROUNDS_INIT                      \
A2 = XOR(XOR(XOR(TAP7(A1, A0), B0),              \
     AND(SRA(A0),SET(0x4a6d))), SLL(A0));        \
B2 = XOR(XOR(B1, AND(SRA(B0), SET(0xcc87))),     \
     XOR(A0, SLL(B0)));                          \
A2 = XOR(A2, XOR(ADD(B1, R1), R2));              \
A0 = ADD(R2, R3);                                \
R3 = AESR(R2, A2);                               \
R2 = AESR(R1, ZERO());                           \
R1 = SIGMA(A0);                                  \
A0 = XOR(XOR(XOR(TAP7(A2, A1), B1),              \
     AND(SRA(A1),SET(0x4a6d))), SLL(A1));        \
B0 = XOR(XOR(B2, AND(SRA(B1), SET(0xcc87))),     \
     XOR(A1, SLL(B1)));                          \
A0 = XOR(A0, XOR(ADD(B2, R1), R2));              \
A1 = ADD(R2, R3);                                \
R3 = AESR(R2, A0);                               \
R2 = AESR(R1, ZERO());                           \
R1 = SIGMA(A1);                                  \
A1 = XOR(XOR(XOR(TAP7(A0, A2), B2),              \
     AND(SRA(A2),SET(0x4a6d))), SLL(A2));        \
B1 = XOR(XOR(B0, AND(SRA(B2), SET(0xcc87))),     \
     XOR(A2, SLL(B2)));                          \
A1 = XOR(A1, XOR(ADD(B0, R1), R2));              \
A2 = ADD(R2, R3);                                \
R3 = AESR(R2, A1);                               \
R2 = AESR(R1, ZERO());                           \
R1 = SIGMA(A2);                                  \
R1 = XOR(R1, LOAD(key));                         \
A2 = XOR(XOR(XOR(TAP7(A1, A0), B0),              \
     AND(SRA(A0),SET(0x4a6d))), SLL(A0));        \
B2 = XOR(XOR(B1, AND(SRA(B0), SET(0xcc87))),     \
     XOR(A0, SLL(B0)));                          \
A2 = XOR(A2, XOR(ADD(B1, R1), R2));              \
A0 = ADD(R2, R3);                                \
R3 = AESR(R2, A2);                               \
R2 = AESR(R1, ZERO());                           \
R1 = SIGMA(A0);                                  \
R1 = XOR(R1, LOAD(key + 16));                    \
```

Listado A.3: Rondas Inicialización

A.4. Repositorio de implementaciones realizadas

Enlace al repositorio: **`https://github.com/FNDme/TFG_5G-Ciphers.git`**

Cualquier propuesta de modificación que ofrezca una mejora es bienvenida.

Bibliografía

[1] Katz, M., Matinmikko-Blue, M., Latva-Aho, M.: "6Genesis Flagship Program: Building the Bridges Towards 6G-Enabled Wireless Smart Society and Ecosystem", *IEEE Latin-American Conference on Communications (LATINCOM)*, pp. 1-9, 2018. doi: 10.1109/LATINCOM.2018.8613209

[2] National Institute of Standards and Technology: .Advanced encryption standard", NIST FIPS PUB 197, 2001.

[3] Institute of Software Chinese Academy of Science: "Zu Chongzhi Algorithm: Draft of ZUC-256 Algorithm Announced", 2018. http://www.is.cas.cn/ztzl2016/zouchongzhi/201801/W020180416526664982687.pdf

[4] Ekdahl, P., Johansson, T.: "SNOW-a new stream cipher", *First open NESSIE workshop*, pp. 167-168, 2000.

[5] Sakamoto, K., Liu, F., Nakano, Y., Kiyomoto, S., Isobe, T.: Rocca: an efficient AES-based encryption scheme for beyond 5G", *IACR Transactions on Symmetric Cryptology*, pp. 1-30, 2021.

[6] Ekdahl, P., Maximov, A., Johansson, T., Yang, J.: "SNOW-Vi: An extreme performance variant of SNOW-V for lower grade CPUs", *ACM Conference on Security and Privacy in Wireless and Mobile Networks*, vol. 1, 2021.

[7] 3gpp.org: "Specification of the 3GPP Confidentiality and Integrity Algorithms UEA2 & UIA2; Document 2: SNOW 3G specification", 2015.

[8] Orhanou, G., El Hajji, S., Bentaleb, Y., Laassiri, J.: "EPS Confidentiality and Integrity mechanisms - Algorithmic Approach", *International Journal of Computer Science Issues*, vol. 7, pp. 15-23, 2010.

[9] ETSI/SAGE: "Specification of the 3GPP Confidentiality and Integrity Algorithms 128-EEA3 & 128-EIA3. Document 2: ZUC Specification, Version: 1.6", 2011.

[10] 3GPP: "The Mobile Broadband Standard". https://www.3gpp.org

[11] Ekdahl, P., Johansson, T., Maximov, A., Yang, J.: .A new SNOW stream cipher called SNOW-V", *ToSC*, pp. 1-42, 2019.

[12] Anand, R., Isobe, T.: "Differential Fault Attack on Rocca", *Information Security and Cryptology – ICISC*, pp. 283–295, 2022. https://doi.org/10.1007/978-3-031-08896-4_14

[13] Hosoyamada, A., Inoue, A., Ito, R., Iwata, T., Minematsu, K., Sibleyras, F., Todo, Y.: Çryptanalysis of Rocca and Feasibility of Its Security Claim", *IACR Transactions on Symmetric Cryptology*, vol. 2022(3), pp. 123–151, 2022. https://doi.org/10.46586/tosc.v2022.i3.123-151

[14] Hawkes, P., Rose, G.G.: "Guess-and-determine attacks on SNOW", *Selected Areas in Cryptography—SAC*, vol. 2595, pp. 37–46, 2002.

[15] Coppersmith, D., Halevi, S., Jutla, C.S.: Çryptanalysis of stream ciphers with linear masking", *Advances in Cryptology—CRYPTO*, vol. 2442, pp. 515–532, 2002.

[16] Ekdahl, P., Johansson, T.: .A New Version of the Stream Cipher SNOW", *Selected Areas in Cryptography*, vol. 2595, pp. 47–61, 2002.

[17] Ma, S., Jin, C., Guan, J., Liu, S.: "Improved differential attacks on the reduced-round SNOW-V and SNOW-Vi stream cipher", *Journal of Information Security and Applications*, vol. 71, pp. 103379, 2022. https://doi.org/10.1016/j.jisa.2022.103379

[18] Gueron, S.: "Intel advanced encryption standard (AES) new instructions set", *Intel White Paper, Rev, 3*, pp. 1-94, 2010.

[19] Koc, C.K.: .Analysis of sliding window techniques for exponentiation", *Computers and Mathematics with Applications*, vol. 30, pp. 17–24, 1995.