PROPUESTA DE OPTIMIZACIÓN DEL ALGORITMO DEL CIFRADOR EN FLUJO TRIVIUM PARA APLICACIONES EN TIEMPO REAL SOBRE LA PLATAFORMA NETFPGA 1G

Edward F. Penagos Granada
Bernardo Cuervo Benítez.

Proyecto de Grado para optar al Título de
Máster en Ingeniería de Sistemas y Computación

UNIVERSIDAD TECNOLÓGICA DE PEREIRA
Facultad de ingenierías
Maestría en ingeniería de sistemas y computación
Pereira
2017
PROPUESTA DE OPTIMIZACIÓN DEL ALGORITMO DEL CIFRADOR EN FLUJO TRIVIUM PARA APLICACIONES EN TIEMPO REAL SOBRE LA PLATAFORMA NETFPGA 1G

Edward F. Penagos Granada

Bernardo Cuervo Benítez.

Director

Ana María López Echeverry.

Docente UTP

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

Facultad de ingenierías

Maestría en ingeniería de sistemas y computación

Pereira

2017
Nota de aceptación:

________________________________________
________________________________________
________________________________________
________________________________________
________________________________________
________________________________________

Firma del presidente del jurado

________________________________________

Firma del jurado

________________________________________

Firma del jurado

Pereira, 2017
Este proyecto está dedicado a mis padres y hermanos quienes han estado siempre a mi lado brindándome apoyo incondicional, a Laura, una de mis principales motivaciones, y a toda mi familia. También va dirigido a mi compañero Belna que me disculpará la intensidad que tuve jaja, a la ingeniera Ana quien nos apoyó en todo momento y las demás personas que han hecho todo esto posible.

Dedicado a mis padres por ser únicos y creer en mí,
A mi esposa por su inmenso apoyo incondicional y a Martina por alegrar mis días,
A la Ingeniera Ana María por su apoyo siempre,
A mi compañero Edward por su GRAN colaboración,
Y a toda mi familia por su ayuda.
Contenido

RESUMEN .......................................................................................................................... 11
CAPÍTULO 1 DEFINICIÓN DEL PROBLEMA ..................................................................... 15
CAPÍTULO 2 JUSTIFICACIÓN ........................................................................................... 17
CAPÍTULO 3 OBJETIVO GENERAL ................................................................................. 19
    3.1 OBJETIVOS ESPECÍFICOS ...................................................................................... 19
CAPÍTULO 4 DISEÑO METODOLÓGICO ....................................................................... 20
    4.2 Tipo de Investigación .............................................................................................. 23
    4.3 Unidad de Análisis ................................................................................................. 23
    4.4 Variables .................................................................................................................. 23
    4.5 Diseño de instrumentos para toma de información .................................................. 24
CAPÍTULO 5 MARCO TEÓRICO ..................................................................................... 25
    5.1 Cifradores en Flujo .................................................................................................. 28
    5.2 Componente Matemático ......................................................................................... 40
CAPÍTULO 6 MARCO LEGAL ......................................................................................... 91
    6.1 Ley 1273 de 2009 Delitos informáticos [46] .............................................................. 91
    6.2 Ley de inteligencia y contrainteligencia (Ley No. 1621 de 2013) ......................... 92
    6.3 Circular Externa 052 de 2007 de la SUPERINTENDENCIA FINANCIERA DE COLOMBIA [48] ... 92
CAPÍTULO 7 ESTADO DEL ARTE .................................................................................... 94
CAPÍTULO 8 IMPLEMENTACIÓN DEL CIFRADOR EN FLUJO TRIVIUM BAJO LA PLATAFORMA NETFPGA 1G ......................................................................................................................... 99
    8.1 Implementación del algoritmo en software ............................................................... 100
    8.2 Implementación del algoritmo en hardware .......................................................... 104
    8.3 Implementación del algoritmo sobre la plataforma NetFPGA 1G. ......................... 111
CAPÍTULO 9 IMPLEMENTACIÓN DEL CIFRADOR EN FLUJO TRIVIUM OPTIMIZADO BAJO LA PLATAFORMA NETFPGA 1G ......................................................................................................... 139
CAPÍTULO 10 CONCLUSIONES ..................................................................................... 158
CAPÍTULO 11 TRABAJO FUTURO .................................................................................. 160
CAPÍTULO 12 RECURSOS ........................................................................................................................................ 161
12.1 Técnicos.................................................................................................................................................. 161
CAPÍTULO 13 CRONOGRAMA........................................................................................................................ 162
CAPÍTULO 14 REFERENCIAS BIBLIOGRÁFICAS............................................................................................ 163
INDICE DE TABLAS

TABLA 4.1 INSTRUMENTO PARA EL REGISTRO DE LA INFORMACIÓN DE LA IMPLEMENTACIÓN............................................. 24
TABLA 5.1 PROBABILIDADES DE OBTENER 0 Y 1 PARA UN CIFRADO PARA DIFERENTES TIPOS DE COMBINACIONES.................................................................................................................................................. 32
TABLA 5.2 ESTADOS DE TRANSICIÓN DE LA ESTRUCTURA (5) .............................................................................................................. 46
TABLA 8.1 RESULTADOS DE LA IMPLEMENTACIÓN EN SOFTWARE CPU PENTIUM R DE 1981 MHZ.................. 102
TABLA 8.2 RESULTADOS DE LA IMPLEMENTACIÓN EN SOFTWARE CPU AMD FX 6100 DE 3314.9 MHZ.. 103
TABLA 8.3 SEÑALES DEL TRIVIUM_GENERATOR. ................................................................................................................................. 106
TABLA 8.4 RESULTADOS DE LA IMPLEMENTACIÓN SOBRE FPGAS. ........................................................................................................ 107
TABLA 8.5 RESULTADOS DE LA IMPLEMENTACIÓN SOBRE FPGAS (2). ................................................................................................... 107
TABLA 8.6 RESULTADOS DE LA IMPLEMENTACIÓN SOBRE LA NetFPGA 1G. FUENTE: LOS AUTORES...... 136
TABLA 9.1 DESEMPEÑO DEL DISEÑO DEL MÓDULO. ................................................................................................................................. 142
TABLA 9.2 RESULTADOS DE LAS PRUEBAS DEL MÓDULO. FUENTE: LOS AUTORES.................................................. 154
TABLA 9.3 RESULTADOS DE LAS PRUEBAS DEL MÓDULO. FUENTE: LOS AUTORES................................. 155
TABLA 9.4 RESULTADOS DE LAS PRUEBAS DEL MÓDULO (2). FUENTE: LOS AUTORES................................................. 156
### INDICE DE FIGURAS

<table>
<thead>
<tr>
<th>FIGURA</th>
<th>DESCRIPCIÓN</th>
<th>PÁGINA</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIGURA 4.1</td>
<td>METODOLOGÍA EMPLEADA EN EL PROYECTO</td>
<td>21</td>
</tr>
<tr>
<td>FIGURA 5.1</td>
<td>ESTRUCTURA DE LOS CIFRADORES EN BLOQUE Y EN FLUJO</td>
<td>25</td>
</tr>
<tr>
<td>FIGURA 5.2</td>
<td>ESQUEMA DE UN CIFRADO EN FLUJO</td>
<td>30</td>
</tr>
<tr>
<td>FIGURA 5.3</td>
<td>ESQUEMA DE CIFRADO Y DESCIFRADO DE UN CIFRADO EN FLUJO</td>
<td>31</td>
</tr>
<tr>
<td>FIGURA 5.4</td>
<td>INTERIOR DE UN CIFRADOR EN FLUJO</td>
<td>32</td>
</tr>
<tr>
<td>FIGURA 5.5</td>
<td>ESQUEMA DE CIFRADO CON UN PRNG</td>
<td>36</td>
</tr>
<tr>
<td>FIGURA 5.6</td>
<td>FUNCIONAMIENTO DE UN LFSR (1)</td>
<td>37</td>
</tr>
<tr>
<td>FIGURA 5.7</td>
<td>FUNCIONAMIENTO DE UN LFSR (2)</td>
<td>37</td>
</tr>
<tr>
<td>FIGURA 5.8</td>
<td>ESQUEMA GENERAL DE UN LFSR</td>
<td>39</td>
</tr>
<tr>
<td>FIGURA 5.9</td>
<td>EJEMPLO DE REGISTROS ACTIVOS</td>
<td>42</td>
</tr>
<tr>
<td>FIGURA 5.10</td>
<td>ESTRUCTURA DE UN FIBONACCI LFSR</td>
<td>43</td>
</tr>
<tr>
<td>FIGURA 5.11</td>
<td>ESTRUCTURA DE UN GALOIS LFSR</td>
<td>44</td>
</tr>
<tr>
<td>FIGURA 5.12</td>
<td>GRAFO $G_n$ CON $N=1$</td>
<td>47</td>
</tr>
<tr>
<td>FIGURA 5.13</td>
<td>GRAFOS $G_n$ CON $N=3$ Y $4$</td>
<td>48</td>
</tr>
<tr>
<td>FIGURA 5.14</td>
<td>GRAFO DEL ESTADO DE TRANSICIÓN DEL MAPEO (2)</td>
<td>52</td>
</tr>
<tr>
<td>FIGURA 5.15</td>
<td>ILUSTRACIÓN DEFINICIÓN 1</td>
<td>53</td>
</tr>
<tr>
<td>FIGURA 5.16</td>
<td>ILUSTRACIÓN DEFINICIÓN 2</td>
<td>53</td>
</tr>
<tr>
<td>FIGURA 5.17</td>
<td>ILUSTRACIÓN DEFINICIÓN 3</td>
<td>54</td>
</tr>
<tr>
<td>FIGURA 5.18</td>
<td>PASOS PARA REDUCIR UN FEEDBACK GRAPH DE UN FIBONACCI NLFSR</td>
<td>55</td>
</tr>
<tr>
<td>FIGURA 5.19</td>
<td>GRAFO DE ESTADO DE TRANSICIÓN DEL EJEMPLO 2</td>
<td>57</td>
</tr>
<tr>
<td>FIGURA 5.20</td>
<td>REPRESENTACIÓN GRÁFICA DE LOS VALORES DE UNA VARIABLE A TRAVÉS DEL TIEMPO</td>
<td>58</td>
</tr>
<tr>
<td>FIGURA 5.21</td>
<td>REDUCCIÓN DE UN NLFSR CUALQUIERA CON EL VALOR TAO IDENTIFICADO (1)</td>
<td>62</td>
</tr>
<tr>
<td>FIGURA 5.22</td>
<td>REDUCCIÓN DE UN NLFSR CUALQUIERA CON EL VALOR TAO IDENTIFICADO (2)</td>
<td>63</td>
</tr>
<tr>
<td>FIGURA 5.23</td>
<td>REDUCCIÓN DE UN NLFSR CUALQUIERA CON EL VALOR TAO IDENTIFICADO (3)</td>
<td>63</td>
</tr>
<tr>
<td>FIGURA 5.24</td>
<td>REDUCCIÓN DE UN NLFSR CUALQUIERA CON EL VALOR TAO IDENTIFICADO (4)</td>
<td>63</td>
</tr>
<tr>
<td>FIGURA 5.25</td>
<td>ILUSTRACIÓN DEFINICIÓN 2 GALOIS-GALOIS (1)</td>
<td>73</td>
</tr>
<tr>
<td>FIGURA 5.26</td>
<td>ILUSTRACIÓN DEFINICIÓN 2 GALOIS-GALOIS (2)</td>
<td>74</td>
</tr>
<tr>
<td>FIGURA 5.27</td>
<td>ILUSTRACIÓN DEFINICIÓN 2 GALOIS-GALOIS (3)</td>
<td>74</td>
</tr>
<tr>
<td>FIGURA 5.28</td>
<td>ILUSTRACIÓN DEFINICIÓN 2 GALOIS-GALOIS (4)</td>
<td>75</td>
</tr>
<tr>
<td>FIGURA 5.29</td>
<td>ILUSTRACIÓN DEFINICIÓN 2 GALOIS-GALOIS (5)</td>
<td>75</td>
</tr>
<tr>
<td>FIGURA 5.30</td>
<td>ESTRUCTURA DEL TRIVIUM PROPUESTA POR SUS CREADORES C. DE CANNIERE, B. PRENEEL</td>
<td>83</td>
</tr>
<tr>
<td>FIGURA 5.31</td>
<td>ESTRUCTURA DEL TRIVIUM (1)</td>
<td>84</td>
</tr>
<tr>
<td>FIGURA 5.32</td>
<td>ESTRUCTURA DEL TRIVIUM (2)</td>
<td>85</td>
</tr>
<tr>
<td>FIGURA 5.33</td>
<td>ESTRUCTURA DEL TRIVIUM PROPUESTA POR ELENA DUBROVA</td>
<td>86</td>
</tr>
<tr>
<td>FIGURA 8.1</td>
<td>ESQUEMA DEL MÓDULO TRIVIUM_CORE, FUENTE: MICHAEL CALVIN MCCOY</td>
<td>105</td>
</tr>
<tr>
<td>FIGURA 8.2</td>
<td>ESQUEMA DEL MÓDULO TRIVIUM_GENERATOR, FUENTE: MICHAEL CALVIN MCCOY</td>
<td>106</td>
</tr>
<tr>
<td>FIGURA 8.3</td>
<td>ESTRUCTURA INTERNA DEL MÓDULO TRIVIUM_GENERATOR, FUENTE: LOS AUTORES</td>
<td>108</td>
</tr>
<tr>
<td>FIGURA 8.4</td>
<td>ESTRUCTURA INTERNA DEL MÓDULO TRIVIUM_GENERATOR AMPLIADA, FUENTE: LOS AUTORES</td>
<td>109</td>
</tr>
<tr>
<td>FIGURA 8.5</td>
<td>SIMULACIÓN DEL TRIVIUM_CORE I, FUENTE: LOS AUTORES</td>
<td>109</td>
</tr>
<tr>
<td>FIGURA 8.6</td>
<td>SIMULACIÓN DEL TRIVIUM_CORE II, FUENTE: LOS AUTORES</td>
<td>110</td>
</tr>
</tbody>
</table>
Figura 8.7 Simulación del Trivium_Generator I. Fuente: Los Autores. ....................................... 110
Figura 8.8 Simulación del Trivium_Generator II. Fuente: Los Autores ..................................... 111
Figura 8.9 Diseño sobre el pipeline de referencia de la NetFPGA. Fuente: NetFPGA .......... 115
Figura 8.10 Comunicación inter-módulos. Fuente: NetFPGA .................................................. 117
Figura 8.11 Estructura de directorios de la NetFPGA. Fuente: NetFPGA ......................... 118
Figura 8.12 Estructura de directorios de la NetFPGA. Fuente: NetFPGA ......................... 119
Figura 8.13 Estructura de directorios de la carpeta lib. Fuente: NetFPGA ......................... 119
Figura 8.14 Estructura de directorios de la carpeta crypto_nic. Fuente: NetFPGA ........ 120
Figura 8.15 Diseño sobre el pipeline de referencia de la NetFPGA con el módulo Trivium agregado. Fuente: Modificada por el autor. ................................................................. 120
Figura 8.16 Metodología de desarrollo sobre NetFPGA. Fuente: Elaboración propia. .... 123
Figura 8.17 Sistema de registros de la NetFPGA, fuente: NetFPGA.org ......................... 124
Figura 8.18 Señales del bus de registros. Fuente: NetFPGA ................................................. 125
Figura 8.19 Interacción de un paquete a través de las diferentes capas. Fuente: NetFPGA. 127
Figura 8.20 Definición del módulo crypto. Fuente: Los autores ...................................... 128
Figura 8.21 Proceso de cifrado en el payload y la llave. Fuente: Modificada por el autor. 128
Figura 8.22 Diseño sobre el pipeline de referencia de la NetFPGA con el módulo Trivium agregado. Fuente: Modificada por el autor. ................................................................. 129
Figura 8.23 Archivo xml del proyecto crypto_nic. Fuente: NetFPGA ................................. 130
Figura 8.24 Archivo xml del proyecto I. Fuente: NetFPGA .................................................. 130
Figura 8.25 Archivo xml del proyecto II. Fuente: NetFPGA .................................................. 131
Figura 8.26 Estructura interna del módulo crypto_nic. Fuente: NetFPGA ....................... 132
Figura 8.27 Estructura interna de un paquete. Fuente: NetFPGA ...................................... 132
Figura 8.28 Ejemplo de implementación del proyecto crypto_nic. Fuente: NetFPGA ..... 134
Figura 8.29 Consumo de recursos de la FPGA. Fuente: Modificada por el autor. .......... 135
Figura 8.30 Consumo de recursos de la FPGA Virtex II 2vp50. Fuente: Modificada por el autor................................................................. 136
Figura 8.31 Consumo de recursos de la FPGA. Fuente: Modificada por el autor .......... 136
Figura 8.32 Consumo de recursos sobre la FPGA del Trivium_Core_Opt. Fuente: Modificada por el autor ................................................................. 137
Figura 8.33 Consumo de recursos de la FPGA. Fuente: Modificada por el autor .......... 137
Figura 8.34 Consumo de recursos de la FPGA xc2vp50. Fuente: Modificada por el autor. 138
Figura 8.35 Consumo de recursos de la FPGA. Fuente: Modificada por el autor .......... 138
Figura 9.1 Resumen del diseño del Trivium core optimizado sobre la NetFPGA 1G. Fuente: Modificada por el autor ................................................................. 140
Figura 9.2 Consumo de recursos del Trivium core optimizado sobre la NetFPGA 1G. Fuente: Modificada por el autor ................................................................. 140
Figura 9.3 Resumen del diseño del Trivium Generator optimizado sobre la NetFPGA 1G. Fuente: Modificada por el autor ................................................................. 140
Figura 9.4 Consumo de recursos del Trivium Generator Opt sobre la NetFPGA 1G. Fuente: Modificada por el autor ................................................................. 141
Figura 9.5 Arquitectura del módulo para encriptar. Fuente: Modificada por el autor. .... 143
Figura 9.6 Arquitectura del módulo para desencriptar. Fuente: Modificada por el autor. 144
Figura 9.7 Líneas agregadas al MAIN_TRIVIUM. Fuente: Modificada por el autor .......... 145
FIGURA 9.8 Funciones actualizadoras del TRIVIUM_CORE. Fuente: Modificada por el autor. .......................................................................................................................................................................................... 145
FIGURA 9.9 Diseño sobre el pipeline de referencia de la NetFPGA con el módulo TRIVIUM optimizado agregado. Fuente: Modificada por el autor. .......................................................................................................................... 146
FIGURA 9.10. Simulación del TRIVIUM_Core_Opt I. Fuente: Los autores. .......................................................... 147
FIGURA 9.11 Simulación del TRIVIUM_Core_Opt II. Fuente: Los autores. .......................................................... 147
FIGURA 9.12 Simulación del TRIVIUM_Core_Opt III. Fuente: Los autores. ....................................................... 148
FIGURA 9.13 Simulación del TRIVIUM_Core_Opt IV. Fuente: Los autores. ....................................................... 148
FIGURA 9.15 Ubicación de la cadena cifrante CORE_OUT. Fuente: Los autores. .......................... 149
FIGURA 9.16 Ubicación de la operación de cifrado. Fuente: Los autores. .............................................. 150
FIGURA 9.17 Cadena cifrante CORE_OUT. Fuente: Los autores. .............................................................. 150
FIGURA 9.18 Señales Plaintext_in, Ciphertext_out y core_out. Fuente: Los autores. ...................... 151
FIGURA 9.19 Arreglos Plaintext_in, Ciphertext_out y core_out con valores. Fuente: Los autores. .................................................................................................................................................. 151
FIGURA 9.20 Periodo del diseño. Fuente: Los autores. .................................................................................... 152
FIGURA 9.21 Simulación del TRIVIUM_Generator_Opt I. Fuente: Los autores. ............................................ 152
FIGURA 9.22 Simulación del TRIVIUM_Generator_Opt II. Fuente: Los autores. ...................................... 153
FIGURA 9.23 Simulación del TRIVIUM_Generator_Opt III. Fuente: Los autores. .................................... 153
FIGURA 9.24 Funciones actualizadoras con valores. Fuente: Los autores. .............................................. 154
FIGURA 9.25 Comparación del consumo de recursos del TRIVIUM_Core y el TRIVIUM_Core_Opt. Fuente: Los autores. .................................................................................................................................................. 155
Las organizaciones deben compartir información de carácter confidencial entre sus dependencias o con otras organizaciones, sin embargo con el aumento de los incidentes de seguridad de la información, es necesario presentar una propuesta de fácil acceso para que las organizaciones puedan cumplir con la integridad, seguridad y confidencialidad, a través del cifrado de datos.

La idea central del proyecto es presentar una optimización del algoritmo del cifrador el flujo Trivium para aplicaciones en tiempo real y realizar la implementación en hardware bajo la plataforma NetFPGA 1G.

Se presenta entonces en este proyecto información asociada con la seguridad, relacionados con el cifrado y la integridad de datos, además de una propuesta de diseño de implementación en hardware para el algoritmo Trivium.

A lo largo de este proyecto se realizará una recopilación de varios estudios realizados por múltiples investigadores de distintas instituciones, con el objetivo de transformar un No Linear Feeback Shift Register (NLFSR) que se encuentra en una configuración Fibonacci a una configuración en Galois NLFSR y posteriormente extender dicha transformación a un caso más general de shift registers en una configuración Galois-Galois NLFSR para optimizar su funcionamiento reduciendo el retardo de la propagación.

Con el algoritmo desarrollado a lo largo de este documento se pretende modificar un tipo de Galois NLFSR como es el caso del cifrador en flujo Trivium, para determinar si es posible aplicar los resultados de esta optimización y obtener resultados más eficientes en aplicaciones en tiempo real implementadas sobre la plataforma NetFPGA 1G. Por último, se dará una descripción de la plataforma y del montaje del algoritmo en la NetFPGA 1G.
En cuanto a los beneficios concretos que la plataforma brindaría a la comunidad académica y científica están:

- Garantizar la integridad y seguridad de los datos
- Disminuir los costos de tecnología especializada
- Generar soluciones eficientes y efectivas.

**Palabras clave**— Cifrador en flujo, Fibonacci, Galois, NetFPGA 1G, NLFSR, Optimización, Trivium.
INTRODUCCIÓN

La criptografía, con el paso de los años, ha surgido como una herramienta para la protección de la información a través del ocultamiento de la misma, y ha venido evolucionando desde la antigüedad hasta nuestros días.

Las organizaciones presentan dificultades e incidentes de seguridad de la información cuando deben almacenar, transferir o transmitir datos entre sus dependencias o con otras organizaciones.

Es necesario entonces contar con soluciones de seguridad de la información que velen por la seguridad e integridad de los datos, sin que se generen cuellos de botella en los procesos de negocio y en la información que puede circular en la organización.

Con respecto al contenido del documento en primera instancia se realizará un estudio sobre el funcionamiento de los cifradores en flujo, a su vez que se realizara un estado del arte referente al área de investigación del mismo buscando en revistas electrónicas, cursos y libros.

En segunda instancia se procederá a realizar el proceso de implementación del cifrador en flujo Trivium, de acuerdo a las especificaciones de los creadores del cifrador, bajo la plataforma NetFPGA 1G y se realizarán pruebas de cifrado.

Posteriormente se optimizará el cifrador en flujo Trivium utilizando un proceso de transformación de un Fibonacci NLFSR a un Galois NLFSR y posteriormente extender la transformación de Fibonacci-Galois a un caso más general de shift registers Galois-Galois con conexiones feedback y feedforward. Para ello se utilizarán las investigaciones.

Más adelante se implementará el cifrador en flujo Trivium con la optimización propuesta bajo la plataforma NetFPGA 1G.
Para terminar, basados en los resultados obtenidos en las implementaciones y en el desarrollo de las pruebas se procederá evaluar los resultados de las pruebas de implementación del algoritmo trívium y el algoritmo trívium optimizado.

Se presenta adicionalmente conclusiones y posibilidades de trabajo futuro que se desprenden del presente proyecto.
CAPÍTULO 1 DEFINICIÓN DEL PROBLEMA

La criptografía, con el paso de los años, ha surgido como una herramienta para la protección de la información a través del ocultamiento de la misma [1]. Esta ha venido evolucionando desde la antigüedad hasta nuestros días [2]. No obstante, así como evolucionaban los métodos de cifrado, también lo hacían las técnicas de criptoanálisis [3], por esta razón, se debieron realizar fuertes estudios en técnicas de cifrado a tal punto que hoy en día se le considere como una ciencia compleja, soportada en teorías matemáticas como la teoría de los números, matemática discreta, complejidad algorítmica[4][5] y en especial la teoría de la información que fue formulada por Shannon en su artículo “A Mathematical Theory of Communication” de 1948[6]. En este, Shannon determinó que el método de cifrado OTP [7] (1917) implementado de la forma correcta posee un secreto perfecto, y por lo tanto es imposible de romper. El problema del OTP yace en que la llave debe ser igual a la longitud del mensaje a cifrar, y no es práctico utilizarlo [8] por el gran tamaño de los paquetes usados hoy en día. Actualmente, se siguen realizando fuertes investigaciones e implementaciones en el campo criptográfico en conjunto con el ámbito legal, como lo es la ley de inteligencia y contrainteligencia (Ley No. 1621 de 2013), la cual establece que los operadores de servicios de telecomunicaciones deberán ofrecer el servicio de llamadas de voz cifradas a personal del alto gobierno y de inteligencia. Por otra parte, en el capítulo décimo segundo de la Superintendencia Financiera de Colombia [9], se obliga al uso de aplicaciones criptográficas para todo tipo de organización ya sea financiera, militar, comercial y demás.

Debido a estos requisitos legales y a las diversas necesidades durante el proceso de cifrado, como la velocidad, el rendimiento, la eficiencia y la seguridad [10], solicitadas por parte de los usuarios, se han desarrollado tres esquemas de cifrado: cifrado con llave privada, llave pública y cifradores de resumen de mensajes, sin embargo, los dos primeros son los más conocidos e implementados. Dentro de los cifradores con llave secreta se encuentran los cifradores en bloque y en flujo, siendo
uno de estos últimos el objeto de investigación de este trabajo. Los cifradores en flujo codifican y decodifican un mensaje bit a bit, por consiguiente, son muy eficientes si se implementan sobre hardware para aplicaciones que requieren cifrado en tiempo real como las video llamadas [11] o tecnologías de alta velocidad como la 5G [12]. Sin embargo, este tipo de cifrado no ha tenido mucha acogida debido a que en su mayoría se ha demostrado no poseer características estadísticas seguras y por consiguiente, pueden ser susceptibles a ataques del tipo “key recovery” y “distinguishing attacks”. Por esta razón, se ha optado por migrar al cifrado en bloque, aunque este no posea las ventajas del cifrador en flujo como la velocidad y la simplicidad, desaprovechándose de esta manera los recursos existentes en hardware ya sean de mucha o poca capacidad de cómputo pero que igual han demostrado ofrecer resultados óptimos [13][14].

Motivados por los beneficios ofrecidos por los cifradores en flujo, mejorar la seguridad de estos, y evitar que se desaprovechen los recursos de hardware, la comunidad europea propuso una iniciativa para la realización de cifradores en flujo con un código abierto. Este proyecto fue llamado el eSTREAM Proyect [15] (2004-2008) y dio como resultado tres finalistas, más conocidos como cifradores de perfil tipo 2, para ser implementados en hardware. Uno de los finalistas fue el cifrador Trivium, en el cual se centra la presente investigación, que se encuentra en una configuración Galois NLFSR el cual pretende optimizarse con esta investigación. Recientemente se han realizado estudios para realizar optimizaciones y que han tomado como referencia al trivium [16][17][18][19] pero todos estos se han realizado de manera independiente, por consiguiente, la presente investigación pretende acoplar los resultados de esos estudios para responder a la pregunta de investigación ¿Es posible proponer la optimización del algoritmo del cifrador en flujo trivium para aplicaciones en tiempo real y de esta manera ofrecer los mejores resultados para distintos tipos de necesidades y recursos disponibles como lo es la plataforma NetFPGA 1G?, teniendo en cuenta que de dicha plataforma disponen los investigadores de este proyecto a través del grupo en telecomunicaciones Nyquist y sus instalaciones en el Centro de Desarrollo Vecinal.
CAPÍTULO 2 JUSTIFICACIÓN

La investigación y creación de nuevos ambientes de cifrado en pequeñas y medianas empresas que buscan satisfacer necesidades específicas de sus clientes y aplicaciones o que simplemente desean ofrecer un valor agregado a través de la innovación y publicación de resultados para ayudar a la comunidad nacional e internacional en la búsqueda de mejores diseños de cifrado, resulta ser un gran contratiempo debido a la falta de recursos económicos e intelectuales disponibles para realizar esta tarea [20].

Según el informe “ESET Security Report Latinoamérica 2015” [21], se puede apreciar que sólo el 10% de las empresas asignan más del 10% de su presupuesto a la seguridad de la información, que se ven invertidos en el diseño de políticas de seguridad de la información, firewalls, infraestructura tecnológica, pólizas contra el ciber-crimen, etc. No obstante, estos recursos se están invirtiendo en productos y herramientas consolidadas pero no hay un rubro específico para el desarrollo, investigación e innovación desde un enfoque académico para generar nuevas posibles herramientas o mejorar las existentes.

Por esta razón, la presente investigación aporta e impulsa a la academia para la generación de nuevo conocimiento en desarrollos criptográficos entregando unas buenas bases sobre el funcionamiento de las técnicas de cifrado, apoyado por el grupo de investigación Nyquist que ha adelantado trabajos enfocados en seguridad de la información [23] y el uso de la plataforma NetFPGA, lo cual ha ido fomentando la investigación y la innovación a nivel de pregrado, especialización y maestría, trabajando sobre dicha plataforma para realizar implementaciones de algoritmos criptográficos en hardware y software.

La optimización del algoritmo Trivium en hardware pretende beneficiar los departamentos de TI de las pequeñas, medianas y grandes empresas de la región y del país, donde se presten servicios específicos de aplicaciones en tiempo real,
preservando la seguridad de los datos sensibles, la red empresarial y protegiendo el activo más importante de las organizaciones “la información”, lo anterior articulado con el desarrollo de una política de seguridad de la información se convierte en una estrategia global de seguridad para la empresa.

Adicionalmente, al realizar la optimización del algoritmo, se permiten ver mejoras a nivel ecológico y ambiental puesto que el consumo normal del algoritmo se ha demostrado que se puede reducir en un 20% sin afectar su rendimiento [24], a través de las investigaciones en las cuales se apoya este trabajo, por lo tanto, se busca aprovechar tanto la tecnología ofrecida por la plataforma NetFPGA [25] como las optimizaciones propuestas sobre el cifrador en flujo Trivium para generar un aporte a la comunidad NetFPGA con la contribución del proyecto para la tarjeta 1G, ya que hasta ahora no se ha implementado un proyecto igual.
CAPÍTULO 3 OBJETIVO GENERAL

Proponer una optimización del algoritmo del cifrador en flujo Trivium para aplicaciones en tiempo real sobre la plataforma NetFPGA 1G.

3.1 OBJETIVOS ESPECÍFICOS

- Realizar un estudio sobre el funcionamiento de los cifradores en flujo.
- Implementar el cifrador en flujo Trivium bajo la plataforma NetFPGA 1G.
- Optimizar el cifrador en flujo Trivium utilizando un proceso de transformación de un Fibonacci NLFSR a un Galois NLFSR y posteriormente extender la transformación de Fibonacci-Galois a un caso más general de shift registers Galois-Galois con conexiones feedback y feedforward.
- Implementar el cifrador en flujo Trivium optimizado bajo la plataforma NetFPGA 1G.
- Evaluar los resultados de las pruebas de implementación del algoritmo Trivium y el algoritmo Trivium optimizado.
CAPÍTULO 4 DISEÑO METODOLÓGICO

En primera instancia se realizará un estudio sobre el funcionamiento de los cifradores en flujo, a su vez que se realizara un estado del arte referente al área de investigación del mismo buscando en revistas electrónicas, cursos y libros.

En segunda instancia se procederá a realizar el proceso de implementación del cifrador en flujo Trivium, de acuerdo a las especificaciones de los creadores del cifrador, bajo la plataforma NetFPGA 1G y se realizarán pruebas de cifrado.

Posteriormente se optimizará el cifrador en flujo Trivium utilizando un proceso de transformación de un Fibonacci NLFSR a un Galois NLFSR y posteriormente extender la transformación de Fibonacci-Galois a un caso más general de shift registers Galois-Galois con conexiones feedback y feedforward. Para ello se utilizarán las investigaciones [16][17][18][19][20][21][22].

Más adelante se implementará el cifrador en flujo Trivium con la optimización propuesta bajo la plataforma NetFPGA 1G.

Para terminar, basados en los resultados obtenidos en las implementaciones y en el desarrollo de las pruebas se procederá evaluar los resultados de las pruebas de implementación del algoritmo trívium y el algoritmo trívium optimizado.
Figura 4.1 Metodología empleada en el proyecto.
4.1 Hipótesis.

¿Será posible proponer la optimización del algoritmo del cifrador en flujo Trivium para aplicaciones en tiempo real sobre la plataforma NetFPGA 1G?

1. Realizar un estudio sobre el funcionamiento de los cifradores en flujo.
   - Este objetivo se logrará a través de la documentación de cómo operan los cifradores en flujo, cuál es su importancia y sus principales características, basado en la revisión de la información bibliográfica recolectada.

2. Implementar el cifrador en flujo Trivium bajo la plataforma NetFPGA 1G. Este objetivo se logrará de la siguiente manera:
   - Instalación: La implementación se va a realizar en el laboratorio del grupo Nyquist en Parquesoft.
   - Lenguaje: El algoritmo se implementará en código VHDL o Verilog.
   - Recursos: Se cuenta con 3 tarjetas NetFPGA 1G.
   - Sistema operativo: Se va a instalar Fedora Linux en cada equipo.
   - Se va a diseñar el conjunto de pruebas de operación del algoritmo.

3. Optimizar el cifrador en flujo Trivium utilizando un proceso de transformación de un Fibonacci NLFSR a un Galois NLFSR y posteriormente extender la transformación de Fibonacci-Galois a un caso más general de shift registers Galois-Galois con conexiones feedback y feedforward.

4. Implementar el cifrador en flujo Trivium optimizado bajo la plataforma NetFPGA 1G.
   - Se deben aplicar el mismo conjunto de pruebas diseñado para contar con elementos de comparación.
5. Evaluar los resultados de las pruebas de implementación del algoritmo trivium y el algoritmo trivium optimizado.
   • Una vez el algoritmo esté correctamente implementado se procederá a procesar las pruebas tipo benchmark, para recoger datos y medir el desempeño del mismo.
   • Se recopilará la información en los formatos destinados para este objetivo.
   • Se realizará un análisis de los datos recogidos para sacar conclusiones sobre el rendimiento.

4.2 Tipo de Investigación.

Apunta al tipo de enfoque cuantitativo, ya que utiliza la recolección de datos para probar la hipótesis con base en la medición numérica y el análisis estadístico, con el fin de establecer pautas de comportamiento y probar teorías. Se mide en porcentaje (%).

4.3 Unidad de Análisis.

El objeto concreto que se pretende investigar y medir es la ejecución del algoritmo del cifrador en flujo Trivium a través de la implementación en hardware sobre la plataforma NetFPGA 1G.

4.4 Variables.

Las variables que se van a medir son Throughput, área, Throughout/Area Ratio, frecuencia máxima y tiempo de cifrado.
**Throughput [Mbps]**: Se refiere a la tasa que se mide cuando se produce una nueva salida con respecto al tiempo, o también la información procesada, comúnmente se mide en bits por segundo.

**Area [CLB slices] o (um2)**: Esta medida se determina de acuerdo al número de compuertas lógicas y Lookup Tables (LUT) usadas en el proceso de síntesis del algoritmo, o Slices CLB para Xilinx.

**Throughput/Area Ratio (Mbps/slice)**: Normalmente usada para medir la eficiencia del diseño.

**Frecuencia Máxima [MHz]**: También conocida como Maximum Clock Frecuency. Hace referencia a la frecuencia máxima alcanzada durante la ejecución o síntesis del algoritmo

**Tiempo de cifrado**: Se determina el tiempo en segundos que tarda el proceso para cifrar y descifrar una determinada entrada.

### 4.5 Diseño de instrumentos para toma de información.

- Formato para el registro de la información de la implementación del cifrador en flujo por hardware.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Tabla 4.1 Instrumento para el registro de la información de la implementación.
¿Qué es un cifrador?

Dentro de los tipos de algoritmos criptográficos se encuentran los algoritmos de llave pública y llave secreta (entre los que se encuentran el RSA y el Trívium [26] respectivamente), de este último se tienen los cifradores en bloque y cifradores en flujo [27]. Diferenciar uno de otro es sencillo, como se ve en la figura 5.1, un cifrador en flujo recibe una cadena de bits y la procesa bit a bit junto con la llave secreta para obtener el texto cifrado empleando para ello la función lógica XOR.

Por otra parte, los cifradores en bloque toman bloques de datos (Ej. 64, 128 bits) al mismo tiempo y los procesan simultáneamente para obtener bloques de texto cifrado.

Algunos de los modos de cifrado en bloque como ECB, CBC, PCBC, CFB, OFB, CTR, están clasificados como “Iterated block ciphers” [28] ya que utilizan varias iteraciones (o rondas) para crear la confusión y difusión que garantizan aleatoriedad e independencia del texto cifrado respecto al texto plano [29]. Estas características se obtienen de las S y P boxes.

Ventajas y desventajas de los cifradores en flujo.

Actualmente, los cifradores en bloque son más utilizados que los cifradores en flujo [30] aunque estos últimos se emplean más que todo para aplicaciones que necesitan cifrado en tiempo real o de dispositivos que disponen de pocos recursos computacionales debido a que los algoritmos tienden a ser pequeños y rápidos. Una
de las tecnologías que utiliza este tipo de cifrado son los celulares (Red Móvil GSM) cuyo hardware no posee de un gran espacio físico y capacidad computacional elevada. El cifrador A5/1 se implementó en estos dispositivos móviles, sin embargo, no tuvo mucho éxito ya que la periodicidad de su llave era muy corta y se podía someter fácilmente a criptoanálisis [31]. Otro cifrador en flujo pero que fue utilizado para cifrar el tráfico de internet fue el RC4 que en su momento tuvo gran acogida pero a través de criptoanálisis también pudieron romperlo [32] y ya no se recomienda utilizarlo.

La razón por la que se utilizaban los cifradores en flujo se debía a que estos se consideraban más eficientes al momento de encriptar ya que optimizan los recursos tanto en software (menos instrucciones de procesador o ciclos de procesador) como en hardware (menos uso de compuertas) en comparación a los utilizados por un cifrador en bloque, no obstante la seguridad de estos cifradores la cual consiste en que no exista correlación de la clave con la cadena cifrada, o que se puedan extraer segmento del texto plano desde el texto cifrado [33], ha sido un tema en el que no les ha ido muy bien y ya existen cifradores en bloque modernos que han demostrado ser tan eficientes como los cifradores en flujo, como es el caso del AES.

**Encriptación y decriptación de los cifradores en flujo.**

La función que encripta y decripta un mensaje en los cifradores en flujo es la misma, lo único que varía son los argumentos ingresados a la función.

Esta función trabaja en módulo 2 [0,1] ya que el mensaje que se va a cifrar o descifrar se realiza sobre bits. De acuerdo a la definición de [34] se tiene que cada bit de un mensaje en texto plano $M$ se cifra junto con los bits de una key stream $S$ para obtener un mensaje cifrado $D$ de la siguiente manera:

$$C_i = M_i \oplus S_i \mod 2 \text{ Donde } C_i M_i S_i \in \{0,1\} \ (1)$$

Y la función para decriptar $D$ y obtener el texto plano $M$ es

$$D_i = C_i \oplus S_i \mod 2 \ (2)$$
La razón por la cual ambas encriptan y decifran de la misma forma se puede demostrar a partir de la expresión (2), si se reemplaza (1) dentro de (2) se tiene que:

\[ D_i = (M_i \oplus S_i) \oplus S_i \mod 2 \]

\[ D_i = M_i \oplus S_i \oplus S_i \mod 2 \]

\[ D_i = M_i \oplus 2S_i \mod 2 \]

\[ D_i = M_i \oplus 0 \mod 2 \]

\[ D_i = M_i \mod 2 \quad (3) \]

De esta manera se comprueba que descifrando el mensaje con la misma llave con la cual fue encriptado se obtiene el mismo mensaje.

Para la criptografía y los cifradores en flujo la compuerta XOR es de gran importancia frente a otras compuertas como las OR, AND o NAND, debido a que esta ofrece una probabilidad del 50% de predictibilidad [35], por lo tanto, para un atacante es lo mismo que en un mensaje cifrado haya un 0 o un 1 puesto que ninguno de los dos aporta mayor información.

En el artículo “Trivium specifications” [36] escrito por los autores de este cifrador, se presentan las especificaciones del Trivium, junto con un resumen de las propiedades criptográficas del algoritmo y un pseudo-código para que este sea comprendido más fácilmente. En este artículo, también se encuentra tabla con las compuertas lógicas estimadas para implementaciones de hardware desde 1-bit hasta 64-bits. Algunas de los resultados que se han obtenido al implementar el trivium y otros finalistas del proyecto eSTREAM se puede ver en el artículo “Hardware performance of eSTREAM phase-III stream cipher candidates” [37]. Los resultados son presentados de manera gráfica, se definen claramente cada una de las métricas a utilizar, y concluye con una tabla de resultados cuantificables del desempeño general de cada uno de los cifradores de flujo donde se evidencia que el Trivium lidera en flexibilidad y consumo de potencia.
Gracias a que el trívium posee altos niveles de paralelización con lo cual se puede reducir el consumo de potencia durante el cifrado, se han realizado algunos desarrollos para demostrar que este sí es eficiente, tal es el caso del artículo “Low power implementation of trivium stream cipher” [38] el cual se basa en técnicas de paralelización de corrimiento de registros. Este desarrollo fue simulado en Modelsim, y se pudo determinar que el diseño en efecto reduce el consumo de potencia en un 20% obteniendo el mismo rendimiento.

5.1 Cifradores en Flujo

Dentro de los tipos de algoritmos criptográficos se encuentran los algoritmos de llave pública y llave secreta, de este último se tienen los cifradores en bloque y cifradores en flujo [27].
El objetivo de estos métodos de cifrado ha consistido en ocultar la información de potenciales atacantes o personal no autorizado, es decir, que en el proceso de intercambio de datos esta solo sea visible y entendible para el emisor y el receptor o receptores legítimos. La idea es garantizar que el cripto-sistema sea computacionalmente seguro aun cuando el atacante tenga conocimiento de cómo se realiza el proceso de cifrado. De acuerdo a los principios formulados por Kerckhoffs,1, el sistema debe seguir siendo seguro aun cuando el atacante conoce todos los detalles sobre el sistema incluyendo los algoritmos de cifrado y descifrado, lo único que debe permanecer oculto es la llave secreta (secret key).

En cierta medida se podría llegar a pensar que la mejor alternativa para lograr que un sistema sea más seguro es a través del ocultamiento de los algoritmos o sus

---
detalles, esto es conocido como seguridad por ocultamiento (security by obscurity)\(^2\). No obstante, esta no es una buena práctica, ya que la historia ha demostrado que este tipo de algoritmos se pueden romper cuando se les realiza ingeniería inversa o se haya infiltrado su diseño, pero en resumidas cuentas solo un algoritmo criptográfico que sea de conocimiento público para que sea sometido a pruebas por toda la comunidad internacional y aun así siga ofreciendo seguridad en el proceso de cifrado solo con el ocultamiento de llave, si se puede considerar seguro.

Dentro de los algoritmos de cifrado simétricos, que usan la misma llave para cifrar y descifrar, se encuentran los cifradores en flujo como como el A5/1 o RC4 que actualmente se consideran inseguros, y los cifradores en bloque en sus diferentes modos CTR (Counter Mode), CFB (Cipher Feedback Mode), entre otros\(^3\), sin embargo los más conocidos hoy en día son el DES (Data Encryption Standard) y el AES\(^4\) (Advanced Encryption Standard). El DES ya no es recomendado para su uso sino una de sus variantes como es el 3DES.

A diferencia de los cifradores en bloque que toman bloques de datos (Ej. 64, 128 bits) al mismo tiempo y los procesan simultáneamente para obtener bloques de texto cifrado, los cifradores en flujo encriptan bits individualmente, adicionando una cadena de bits criptográficamente seguros (key stream) a cada bit del texto plano. Existen cifradores en flujo y en bloque en los que la key stream solo depende de la llave y otros en los que también depende del texto plano, estos se conocen como síncronos y asíncronos respectivamente.

Actualmente, los cifradores en bloque son más utilizados que los cifradores en flujo\(^5\), sin embargo, estos últimos son más utilizados para aplicaciones que necesitan cifrado en tiempo real o de dispositivos que disponen de pocos recursos computacionales debido a que los algoritmos tienden a ser pequeños y rápidos.

Una de las primeras aplicaciones criptográficas utilizadas en productos de consumo fue el algoritmo A5/1 6 del estándar GSM. No obstante, su debilidad fue que la periodicidad de su secuencia de bits seguros era muy corta y se podía someter fácilmente a criptoanálisis. Otro cifrador en flujo para cifrar el tráfico de internet fue el RC4 7 que en su momento tuvo gran acogida pero a través de criptoanálisis también pudieron romperlo y ya no se recomienda utilizarlo.

Por otra parte, pese a los avances que se han ido realizando con el paso de los años en el campo criptográfico, el esquema general de un cifrador en flujo se sigue conservando como se ve en la Figura 5.2. Un cifrador recibe el texto plano y una cadena criptográficamente segura para arrojar un texto cifrado criptográficamente seguro.

![Figura 5.2 Esquema de un cifrado en flujo.](image)

Al aplicar este mismo esquema tanto para el proceso de cifrado como de descifrado (Fig. 5.2) se obtiene lo siguiente:

---


Figura 5.3 Esquema de cifrado y descifrado de un cifrado en flujo.

En la figura se puede apreciar que tanto el emisor como el receptor poseen la misma llave para poder realizar el proceso de cifrado y descifrado correctamente. La antena de transmisión se utiliza para hacer énfasis en que el proceso de comunicación entre dos partes no se hace punto a punto entre emisor y receptor sino que hay elementos intermedios pero esta comunicación siempre va cifrada, como se vio en el apartado “Encriptación y decriptación de los cifradores en flujo”.

Como se mencionó en párrafos anteriores, la operación de cifrado se realiza bit a bit, es decir, no se opera en el campo de los números decimales sino en el binario (0s y 1s) o dicho de otra forma en módulo 2.

Para comprender un poco más sobre su funcionamiento, se procede a abrir una de las cajas azules de la figura 5.3. Al interior del algoritmo de cifrado se encuentra una compuerta lógica XOR (Fig. 5.4), que es un equivalente a la operación de adición en módulo 2.
A nivel general se ha hablado de un cifrador en flujo como un algoritmo que recibe una llave y un mensaje en texto plano, pero de ahora en adelante se hablará de ellos como un algoritmo que arroja una cadena de bits criptográficamente seguros para cifrar el mensaje.
La compuerta XOR [35] tiene un gran valor en el proceso de cifrado, ya que la cadena de bits criptográficamente seguros o key stream arrojados por el cifrador para adicionar a los bits del mensaje, tiene una probabilidad del 50% tanto para obtener un uno como para obtener un cero.
Observe la siguiente tabla.

<table>
<thead>
<tr>
<th>M_i</th>
<th>K_i</th>
<th>C_i</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Cuando se hace el XOR entre el bit de un mensaje y un bit de la key stream, cuando el valor de esta última tiene un 0 (cero), el texto plano sale igual y cuando tiene un
1 (uno) cambia. Esto quiere decir que si la key stream cumple con los requisitos criptográficos, que serán vistos más adelante, el mensaje viajará cifrado solo en un 50%, pero esta es la parte compleja, y es que un atacante no sabe si el bit que él está observando es realidad un 0 o un 1, y por consiguiente para él es irrelevante el valor del bit capturado ya que no le está aportando mayor información. Por lo tanto, es más fácil tratar de adivinar la llave que tratar de descifrar un mensaje bit a bit (Recuerde que el algoritmo de cifrado es conocido por el atacante, pero aun así este debe ser seguro). Otra opción es intentar obtener por fuerza bruta la clave con la cual fue cifrado el mensaje, que es lo único que permanece secreto en todo el sistema. Sin embargo, actualmente realizar esta tarea resulta compleja puesto que las llaves empleadas son del orden de 80 a 256 bits y solamente para la de 80 bits se tardaría varias décadas.

Por ejemplo, la cantidad de posibles claves que tiene que probar un atacante para descifrar un mensaje que ha sido cifrado con una llave de 80 bits es de $2^{80}$ bits. Suponga que el atacante tiene 1'000.000 de computadores con procesadores a 5GHz ejecutando este proceso durante 5 años los 365 días las 24 horas, aun con estas condiciones, no alcanzaría a probar todas las claves posibles, de allí que esta opción sea la última alternativa y la más costosa computacionalmente.

Ya que el tratar de romper el sistema por fuerza bruta es complejo, otra manera quizá más eficiente, es observando el criptograma y a partir de este recuperar la llave o el mensaje o alguna parte de estos dos, por lo tanto, es de importancia en el campo criptográfico que el criptograma tenga apariencia aleatoria.

Existen tres tipos de aleatoriedad.

- Generadores de verdaderos números aleatorios (TRNG)\(^8\).

---

• Estos números se obtienen de procesos físicos aleatorios como lo es lanzar una moneda al aire, el ruido presente en los semiconductores, movimiento del mouse, espacio entre digitación, etc.

• Los TRNG tienen una propiedad específica y es que no pueden ser recreados o simulados por un proceso no aleatorio, salvo que este tenga una probabilidad grande de ocurrir. Por ejemplo lanzar una moneda al aire 4 veces genera resultados aleatorios pero puede ser replicada fácilmente, en este caso se tiene que el evento es aleatorio pero no tiene propiedades criptográficas ya que posee un periodo muy corto.

• Normalmente estos son utilizados para generar llaves de sesión o como semillas para generar secuencias pseudo aleatorias.

- Generadores de números pseudo aleatorios (PRNG)\textsuperscript{9,10}.
  
  • Estos no son aleatorios completamente y pueden ser computados, es decir, son determinísticos, de allí su nombre de pseudo aleatorios, sin embargo, estos tienen propiedades que los hacen aptos para fines criptográficos (en algunos casos).

  • La función que genera estos bits, en algunos casos parte desde una semilla que puede ser TRNG.

  • El matemático Solomon W. Golomb propuso tres postulados que son unas condiciones necesarias aunque ya no son suficientes, para que secuencias pseudo aleatorias parezcan aleatorias, así como también existen tests como el chi-cuadrado para verificar propiedades estadísticas en la secuencia de un PRNG.


• Generadores de números pseudo aleatorios seguros criptográficamente (CSPRNG)\textsuperscript{11}.
  
  o Estos siguen siendo un tipo de PRNG solo que tienen un propiedad adicional y es que son impredecibles igual que un TRNG, no obstante, al ser generados por un proceso matemático y computacional poseen cierto grado de dependencia, sin embargo, esta es tan pequeña que se consideran seguros para usos criptográficos.

Por ejemplo, cuando se genera una secuencia verdaderamente aleatoria, hay una probabilidad del 50\% de tener tanto un 0 como un 1, pero en el caso de los CSPRNG, se trabaja con una épsilon sumado a la probabilidad, pero este épsilon debe ser tan pequeño que para un atacante la mejor opción siga siendo realizar fuerza bruta.

\[
Probabilidad (P) = \frac{1}{2} + \epsilon
\]

Lo que se quiere decir con impredecibles es que si un atacante tiene conocimiento sobre una secuencia de bits \(S_0, S_1, ..., S_{i-1}\) le sea imposible computacionalmente construir \(S_i\).

**Generadores de secuencias Pseudoaleatorias.**

Una de las maneras utilizadas para crear secuencias pseudo aleatorias de gran longitud, es utilizando Linear Feedback Shift Registers (LFSRs)\textsuperscript{12}. Estas se utilizan en muchos cifradores en flujo, no en todos, ya que son fáciles de implementar ya sea en hardware o software.

---


Una de las características de estas secuencias es que sean computacionalmente seguras, es decir, que solo teniendo recursos infinitos se podrían romper, no obstante, conseguir esta cantidad de recursos es prácticamente imposible, por ejemplo, en muchos casos la mejor alternativa para atacar un cifrador en flujo es mediante fuerza bruta para descifrar la llave, en el caso del algoritmo de cifrado de llave pública RSA se basa en la factorización de un gran número en dos primos, si bien esto es posible realizarlo, no existen los medios computacionales actualmente para hacerlo de forma eficiente, de allí que sean seguros computacionalmente. Otro tipo de seguridad en un cifrador es que sea seguro incondicionalmente, es decir, que no se pueda romper incluso teniendo recursos computacionales infinitos, tal es el caso del cifrador en flujo OTP\textsuperscript{13}. Este debe cumplir dos condiciones para ser seguro incondicionalmente.

- La llave debe ser TRNG.
- La llave solo se puede usar una vez.

Visto gráficamente un PRNG, luciría de la siguiente manera.

![Figura 5.5 Esquema de cifrado con un PRNG.](image)

Al interior de estos PRNG se encuentran los LFSR que generaran la aleatoriedad.

**LFSRs.**

*Definición 1:* Un LFSR no es más que un flip flop o registro de un solo bit que almacena en su interior un bit que este a la entrada cuando reciba una señal como

lo es por ejemplo un flanco de reloj, y el bit que está en su interior sale y se convierte en la entrada de otro y así sucesivamente.

Un shift register o un registro de corrimiento a la izquierda con una entrada de “011” luce de la siguiente manera.

Estos registros trabajan con la función XOR para retroalimentar nuevamente el sistema, y ya que el XOR es una operación lineal, y la salida de estos se convierten en la entrada del registro que se encuentra más hacia la izquierda y de otros más (según el diseño) se llaman lineales y de retroalimentación, de allí el nombre de LFSR.
En este caso, con la misma entrada (011) la salida antes de que se repita de nuevo sería la siguiente:

- 1101001

Como se puede apreciar este LFSR es un LFSR con una secuencia máxima de $2^3 - 1$, note que no puede existir la entrada o la secuencia de todos los flip flops con un valor de 0 puesto que quedaría estancado.

La secuencia máxima de un LFSR viene dada por $2^n - 1$, donde n representa la cantidad de registros o flip flops. Algo que se debe tener en cuenta es que no todos los LFSRs tiene una secuencia máxima, algunas son más cortas que otras esto depende de los coeficientes de retroalimentación (feedback coefficients). Para implementar LFSRs que generan máximas longitudes se utilizan los polinomios primitivos, estos polinomios están relacionados con los números primos y no pueden ser factorizados.

Del ejemplo anterior se tiene que estos LFSRs se puede expresar como polinomios dando este resultado.

$$P(X) = X^3 + X^2 + 1$$

Como se mencionó previamente, los LFSRs tienen una dependencia lineal, por consiguiente, si se quisiera conocer el bit $S_i$ del ejemplo anterior se puede lograr a partir de un patrón general.

$$S_3 = S_2 + S_0 \mod 2$$
$$S_4 = S_3 + S_1 \mod 2$$
$$...$$
$$S_i = S_{i-1} + S_{i-3} \mod 2$$

El esquema general para construir un LFSR se puede especificar gráficamente de la siguiente manera:
En este caso, cada coeficiente $C_i \in \{0,1\}$ sirve como compuerta para saber cuáles XOR estarán operando. El comportamiento de estos coeficientes viene dado de la siguiente manera:

\[
\begin{cases}
C_i = 1, & S_i \oplus S_{i-m} \\
C_i = 0, & 0 \oplus S_{i-1}
\end{cases}
\]
donde $m$ es un registro más a la derecha que $i$.

La notación de un LFSR algunas veces se especifica como:

\[ P(X) = X^i + C_{i-1}X^{i-1} + \cdots + C_1X^1 + C_0X^0, C_0 = 1 \]

Para el ejemplo tratado anteriormente el polinomio quedaría así:

\[ P(X) = X^3 + C_2X^2 + C_1X^1 + C_0X^0, C_2 = C_0 = 1, C_1 = 0 \]

\[ P(X) = X^3 + 1X^2 + 0X^1 + X^0 \]

\[ P(X) = X^3 + X^2 + 1 \]

Expresado de forma general se tiene lo siguiente.

\[
S_i \equiv S_{i-1} C_{i-1} + S_{i-2} C_{i-2} + \cdots + S_1 C_1 + S_0 C_0 \mod 2
\]

\[
S_{i+1} \equiv S_i C_{i-1} + S_{i-1} C_{i-2} + \cdots + S_2 C_1 + S_1 C_0 \mod 2
\]

\[
\vdots
\]

\[
S_{i+m} \equiv \sum_{j=0}^{i-1} S_{m+j} \times C_j \mod 2
\]
Dado que las salidas de estos sistemas dependen de otras, los ataques más implementados consisten en hallar la correlación que exista entre bits para tratar de encontrar la llave o el texto plano, de allí que sea importante que la cadena de bits tenga propiedades criptográficas y que la llave empleada (que es una semilla creada a partir de un TRNG) solo se utilice una vez o en su defecto se emplee un vector de inicialización (IV) para que la misma llave se pueda seguir utilizando pero la secuencia obtenida sea diferente, esta secuencia de bits también se conoce como “nonce” o número usado una vez. Este IV puede ser un contador, una secuencia aleatoria, una dirección IP, etc, y puede ser pública ya que el atacante podrá saber cuál es su valor, pero mientras no sepa la clave será incapaz de replicar la cadena de bits, existen algunos ataques que requieren que este IV no sea predictable.

Ciffradores más modernos siguen utilizando LFSRs pero no uno solo como se ha visto en este capítulo sino que ya combinan varios de ellos sin que haya componentes lineales usados para encontrar la salida de cada registro.

5.2 Componente Matemático

Shift Registers.

Un LFSR ejecuta dos instrucciones cuando este recibe una señal externa como lo es un flanco de reloj. La primera es almacenar en su interior el bit que se encuentre en la entrada del registro cuando este recibe la señal externa, y la segunda es enviar el bit que está en su interior a la entrada de otro registro o a la salida.\textsuperscript{14}

Sin embargo, pese a que estos producen una secuencia de salida (key stream) de gran longitud, su estructura puede someterse a criptoanálisis, aunque se mantenga su estado inicial en secreto. Este criptoanálisis se puede efectuar debido a que los LFSRs poseen una dependencia lineal de sus estados anteriores y por ello, ya no son usados puramente en los algoritmos de cifrado.\textsuperscript{14} En consecuencia surgieron

\textsuperscript{14} S. Ullah, “Algorithm for Non-Linear Feedback Shift Registers Delay Optimization”, Dept. of Information and Communication Technology, Royal Institute of Technology (KTH), Stockholm, Sweden, Tesis de Maestría, 2010.
los No Linear Feedback Shift Registers (NLFSRs) que son una generalización de los LFSRs en el que su estado actual no es función lineal de un estado anterior haciéndolos más seguros.

Definición 2: Un NLFSR está compuesto por \( n \) elementos de almacenamiento binario llamados “stages” o registros, los cuales tienen asociada una variable \( x_i \) y una función de retroalimentación \( f_i \colon \{0,1\}^n \rightarrow \{0,1\} \) que determina cómo se actualizarán los valores de las variables. Estas funciones de retroalimentación usualmente se representan usando la forma normal algebraica (ANF). Esta forma es un polinomio en \( G_F(2) \) del tipo:

\[
f(x_0, \ldots, x_{n-1}) = \sum_{i=0}^{2^n-1} c_i \cdot x_0^{i_0} \cdot x_1^{i_1} \cdot \ldots \cdot x_{n-1}^{i_{n-1}}
\]

Donde \( c_i \in \{0,1\} \) y \( (i_0, i_1, \ldots, i_{n-1}) \) es la expansión binaria de \( i \) con \( i_0 \) siendo el bit menos significativo. A través de este capítulo, se llamará monomio a los términos de un ANF.

Los valores \( x_i \) de los registros son la entrada de las funciones actualizadoras. Si dicha función solo depende del valor del registro anterior a ella, esta es del tipo:

\[
f_i = x_{i+1}
\]

Y se dice entonces que el NLFSR tiene una configuración Fibonacci (Fig. 5.10), en caso contrario posee una configuración Galois (Fig. 5.11), pero en cualquiera de los dos casos la salida siempre corresponderá al valor que se encuentre en el registro 0.

Por otra parte, si los valores de dichos registros hacen parte de una función actualizadora que no sea del tipo (4), se les conoce como registros activos.

---

Ejemplo: En el registro de la (Fig. 5.9), se puede apreciar que este posee 7 registros y por consiguiente 7 funciones actualizadoras, sin embargo, las funciones actualizadoras $f_2$ y $f_5$ no son del tipo (4) ya que hay más de un registro influenciando en el cálculo del nuevo valor para estas funciones.

Estos registros representados de color verde $x_6, x_4, x_3, x_1$ se consideran como los registros activos para las funciones $f_5$ y $f_2$ respectivamente.

Figura 5.9 Ejemplo de Registros activos.

Las funciones actualizadores de este registro se pueden ver a continuación:

$$f_6 = x_0$$
$$f_5 = x_6 \oplus x_4 \ [no\ es \ del\ tipo\ (4)]$$
$$f_4 = x_5$$
$$f_3 = x_4$$
$$f_2 = x_3 \oplus x_1 \ [no\ es \ del\ tipo\ (4)]$$
$$f_1 = x_2$$
$$f_0 = x_1$$

Por otra parte, es importante tener en cuenta que aunque se tengan dos NLFSRs en configuraciones distintas (Fibonacci o Galois) \(^{19}\), si el conjunto de salidas que arrojan son equivalentes, dichos NLFSRs también lo serán\(^{21}\).

FIBONACCI.

En un LFSR o NLFSR Fibonacci, el único valor que se actualiza es el que se encuentra en el registro $s_{i-1}$ aplicando la función XOR entre los valores previos donde los registros están activos\(^{22}\).

$(s_0 \oplus s_1 \oplus ... \oplus s_{i-2})$, como se ve en la Figura 5.10.

![Figura 5.10 Estructura de un Fibonacci LFSR.](image)

Las demás funciones son del tipo (4).

GALOIS.

En una configuración Galois LFSR o NLFSR hay una función actualizadora por cada registro al igual que ocurre con Fibonacci, la diferencia radica en que en los shift register en Galois el registro $s_{i-1}$ no es el único que se actualiza con la sumatoria de las funciones XOR sino que cualquier otro puede hacerlo\(^{23,24}\). En la figura 5.10 se puede observar como la salida de cada registro (donde los registros están activos) actualiza el registro $s_{i-1}$, mientras que en la figura 5.11, la salida de cada registro actualiza a todos los registros previos.


En el caso de Galois, los registros se actualizan de la siguiente manera. Se hace una sumatoria entre \( s_0 \) con los registros que se encuentren activos para cada función actualizadora. Ej. En la figura 5.11, \( s_1 \) se actualiza realizando un XOR entre \( s_0 \) y \( s_2 \). Es decir, cada registro se actualiza haciendo un XOR entre los valores que se encuentran a su derecha y el valor del registro de la izquierda. Únicamente en \( s_{i-1} \) se presenta una salvedad, y es que \( s_{i-1} \) no tiene registros previos, por consiguiente, este registro se actualiza a sí mismo, ya que su salida va hacia \( s_{i-1} \) también. De allí que haya un \( s'_{i-1} \), porque el \( s_{i-1} \) nuevo es \( s_0 \oplus s_{i-1} \oplus ... 

![Figura 5.11 Estructura de un Galois LFSR.](image)

Como se pudo observar, las funciones actualizadoras tienen asociadas una serie de términos, los cuales influyen en el valor que se obtendrá cuando se ejecute la función, estos términos representan el conjunto de dependencia o conjunto de valores de los cuales depende la función actualizadora.

**CONJUNTO DE DEPENDENCIA.**

Una función booleana \( f_i \) establece una dependencia entre una variable de salida "S" y un conjunto de variables de entrada "a, b, c,...". Una correspondencia entre el conjunto de valores de las variables de entrada y el valor de la variable de salida es lo que se conoce como el conjunto de dependencia o \( \text{dep}(f_i) \).}

---


Ej.: Los valores de dependencia de la función $f_5$ son $x_6$ y $x_4$ $[dep(f_5) = \{x_6, x_4\}]$. Porque son los términos que la función actualizadora utiliza para calcular su nuevo valor.

$$f_5 = x_6 \oplus x_4$$

**Período de un shift register.**

Otro aspecto deseable en las cadenas pseudo-aleatorias como se indicó al comienzo de este capítulo, es que sean de gran longitud. Estas a menudo son generadas usando LFSRs. Las ventajas de los LFSRs incluyen facilidad de implementación, simplicidad, velocidad y la habilidad para generar el máximo ciclo de la secuencia con la misma distribución estadística uniforme de 0’s y 1’s en una secuencia verdaderamente aleatoria. La principal desventaja de los LFSRs es su linealidad, dando lugar a un fácil criptoanálisis. Como alternativa, surgieron los NLFSRs cuyo estado actual no es una función lineal de su estado anterior. Un numero de diferentes implementaciones de NLFSRs han mostrado ser más resistentes a los ataques criptoanálíticos que los LFSRs, sin embargo, la construcción de NLFSRs con periodos largos todavía es un problema que permanece abierto$^2$, ya que hasta ahora no se ha descubierto un algoritmo sistemático para la síntesis de un NLFSR con periodos de longitud $2^n - 1$, en especial para los Galois NLFSR cuya secuencia de salida no es necesariamente igual a la longitud del ciclo más largo de sus estados. Es decir, puede que el ciclo de estados sea máximo pero no el ciclo de salida. En un Fibonacci NLFSR, el periodo de la secuencia de salida siempre es igual a la secuencia del ciclo más largo de sus estados.

Ejemplo: Observe las funciones actualizadoras de un Shift Register de 4 registros (5), cuya secuencia de estados se puede observar en la Tabla 5.2.
\[
\begin{pmatrix}
  x_0 \\
  x_1 \\
  x_2 \\
  x_3
\end{pmatrix}
\rightarrow
\begin{pmatrix}
  x_1 \\
  x_2 \\
  x_3 \\
  x_0 \oplus x_1
\end{pmatrix}
\tag{5}
\]

Tabla 5.2 Estados de transición de la estructura (5)

<table>
<thead>
<tr>
<th>Estados</th>
<th>Salida</th>
</tr>
</thead>
<tbody>
<tr>
<td>X3</td>
<td>X2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Fuente: Autores.

De esta tabla se obtiene que el shift register (5) tiene un periodo o secuencia máxima (En Inglés m-sequence) igual a \(2^n - 1\), recuerde que el estado \((0,0,0,0)\) no es deseable ya que es un estado atrapante, y adicionalmente, ninguno de estos estados se repite, con lo cual se puede decir que tanto la secuencia de estados como la secuencia de salida tienen un periodo igual a 15. Esta característica aplica para cualquier Shift Register en configuración Fibonacci, de allí la definición que un Fibonacci LFSR o NLFSR tiene un periodo igual a la cantidad de estados diferentes que este posea. En cambio, en un Galois NLFSR esta condición no se cumple.
necesariamente. Se puede dar el caso que todos sus estados sean diferentes y el periodo sea más corto, siendo esto, uno de los principales inconvenientes al construir NLFSRs en una configuración en Galois.

Para los Fibonacci NLFSR existen $2^{2^{n-1}-n+1}$ diferentes configuraciones con periodo $2^n − 1^{27}$. Considere un grafo $G_n$ el cual tiene $2^n$ nodos, representando todos los posibles estados de un NLFSR de n-bits, y $2^{n+1}$ aristas, representando todas las posibles transiciones entre estados. Así, cada nodo de $G_n$ tiene dos posibles predecesores y dos posibles sucesores. Suponga un grafo con $n = 1$ (Fig. 5.12), de acuerdo a la definición deberían existir 4 aristas ($2^{1+1}$) como se ve a continuación:

![Figura 5.12 grafo $G_n$ con $n=1$.](image)

A continuación se presenta un caso más extenso (Fig. 5.12) tomado de [14]:

---

Figura 5.13 grafos $G_n$ con n=3 y 4.

En el ejemplo de la izquierda $n = 3$, debe haber $2^3$ nodos y $2^{3+1}$ aristas, y en el ejemplo de la derecha $n = 4$ debe haber $2^4$ nodos y $2^{4+1}$ aristas.

Con estos estados de transición definidos, se pueden encontrar todos los ciclos de longitud $2^n$ en $G_n$ encontrando todos los posibles caminos hamiltonianos:

a. Un camino Euleriano$^{28}$ en un grafo $G$ es un camino que pasa por todas las aristas de $G$. Euler demostró que es posible encontrar tal camino a través de $G$ una vez, si y solo si $G$ está conectado, es decir, si, para cualquier par de vértices, existe un trayecto que los une (En los grafos dirigidos debe tomarse en cuenta la dirección de las aristas para esta definición, formando el par de vértices en cualquier orden), y el mismo número de aristas que entran a cada vértice es igual al número de aristas que salen.

b. Un camino Hamiltoniano$^{29}$ en un grafo $G$ es un camino que pasa a través de todos los vértices de $G$ una y solo una vez. Una solución al problema de hallar

---


un camino hamiltoniano completo cuando el número de vértices \( n \) es muy grande, se puede observar en las figuras 5.12 y 5.13 de los grafo de interconexión \( G_n \) de los shift registers.

Si existe un camino Euleriano a través de las aristas de \( G_{n-1} \), se pueden interpretar estas aristas como los vértices del grafo \( G_n \), por consiguiente un camino Euleriano en \( G_{n-1} \), define un camino Hamiltoiano en \( G_n \) ya que en un caso base, el ciclo \((01\ o\ 10\ en\ la\ figura\ 5.12)\) es un camino Hamiltoniano a través de \( G_1 \), por lo tanto hay una solución al problema del ciclo completo para \( n \) vértices.

De Brujin demostró que si el grafo \( G_n \) tiene \( k \) caminos Hamiltonianos completos, entonces el grafo \( G_{n+1} \) tiene \( 2^{2^{n-1}-1} \ast k \) caminos Hamiltonianos, y teniendo en cuenta que el grafo \( G_1 \) posee un camino Hamiltoniano, por inducción se puede llegar a la conclusión que el número de caminos Hamiltonianos en \( G_n \) esta dado por \( 2^{2^{n-1}-n} \).

Para formar un ciclo de longitud \( 2^n - 1 \) de un ciclo de longitud \( 2^n \), se puede remover uno de los bucles presentes en el nodo \( 00, \ldots, 0 \) o en el nodo \( 11, \ldots, 1 \) de \( G_n \). Debido a que en \( G_n \) hay exactamente dos ciclos de longitud \( 2^n - 1 \) para cada ciclo de longitud \( 2^n \). Por lo tanto, el número de ciclos de longitud \( 2^n - 1 \) en \( G_n \) es \( 2 \ast 2^{2^{n-1}-n} \), dando como resultado los \( 2^{2^{n-1}-n+1} \) NLFSRs con periodos \( 2^n - 1 \) en configuración Fibonacci.

**Transformación de Fibonacci a Galois.**

La mayoría de los NLFSR considerados en este documento son funciones de retroalimentación singulares del tipo \(^{26}\):

\[
 f_i = x_{i+1} \oplus g_i\{x_0, \ldots, x_i, x_{i+2}, \ldots, x_{n-1}\} \quad (6)
\]

Donde cada función \( g_i \) no depende de la variable \( x_{i+1} \), esta condición es muy importante en criptografía y se debe a la invertibilidad del mapeo, es decir, la función debe ser inyectiva\(^ {30}\).

La prueba de esto es la siguiente:
Dos estados consecutivos de un NLFSR se sobrelapan en \( n - 1 \) posiciones (registros de color verde), y solo cambia el valor de la nueva variable "\( S \)" que está llegando al registro \( n - 1 \) como se ve en la Figura 5.14.

![Registro en un momento t](image)

![Registro en un momento t+1](image)

Figura 5.14 Sobrelapamiento de \( n \)-1 registros.

Esto implica que solo pueden existir dos posibles valores que entran y dos posibles valores que salen.

Si \( f \) se encuentra en la forma (6), entonces los 2 posibles estados futuros (\( x \) o \( y \)) del NLFSR que corresponden a la \( n \)-tupla del registro en un tiempo "\( t + 1 \)" podrían ser \( x = \{0, 1, ..., x_{i+1}, ..., x_{n-1}\} \) o \( y = \{1, 1, ..., x_{i+1}, ..., x_{n-1}\} \), es decir, si en un caso dado se da el estado contenido en el registro "\( x \)", este no puede ser igual al estado que se hubiera generado si se hubiera dado el registro "\( y \)" (y viceversa). Por lo tanto si el valor de \( g_i \) tanto para "\( x \)" como para "\( y \)" es el mismo, quiere decir que \( f \) no es

---

de la forma (6), ya que la función estaría mapeando un mismo estado desde dos estados diferentes y por consiguiente no sería una función inyectiva. Con esto se puede decir entonces que una función evaluada en 0 debe ser diferente a cuando se evalúe en 1.

Adicionalmente, $g_i$ no puede depender de la variable $x_{i+1}$, en el sentido que si por alguna probabilidad todos los valores que tome la función $g_i$ en un momento dado son iguales a los valores de $x_{i+1}$ con el que se realiza la operación XOR, dicha función puede caer en un ciclo donde todos sus estados siempre serán cero anulando el mensaje con el que se está trabajando.

Para los NLFSRs en configuración Fibonacci, esta singularidad garantiza que el Grafo de Estado de Transición (STG – State Transition Graph) consista de ciclos puros$^{32}$, es decir, que de un estado se pueda llegar a uno y solo un estado, en caso de suceder lo contrario, la secuencia de salida se reduciría y no sería máxima.

**Grafo de transición y grafo de retroalimentación.**

Los LFSRs y NLFSRs tienen asociado un grafo de estado de transición (STG) que no es más que un grafo dirigido en el que los nodos representan los estados y las aristas indican la posible transición entre estados$^{26}$. El grafo de estado de transición de un Fibonacci NLFSR de n-bits es “branchless”$^{33}$ es decir, que de dos estados distintos no se puede llegar a uno igual, es un ciclo puro.

Una forma de representar un STG es a través de n-tuplas donde cada una de estas representa un estado $(x_0, x_1, ..., x_{n-1})$ por ejemplo el mapeo (7) con el estado $(1,1,1,1) = (x_0, x_1, x_2, x_3)$ tiene asociado un STG que se puede visualizar en la Figura 5.14.

---


En el artículo [24] se introduce una estructura llamada “feedback graph” (o grafo de retroalimentación) que refleja la relación entre variables de las funciones de retroalimentación y en ese mismo artículo se llega a la conclusión que si el grafo de retroalimentación puede ser reducido a un solo vértice (con el algoritmo que será descrito más adelante), existe una recurrencia no lineal equivalente de orden \( n \) tanto para Fibonacci como para Galois. Los NLFSR que cumplen está condición se llaman NLFSRs uniformes.

Usualmente, hay muchos \( n \)-bits Galois NLFSRs que son equivalentes a un \( n \)-bit Fibonacci NLFSR, sin embargo, no todo \( n \)-bit Galois NLFSR tiene un \( n \)-bit Fibonacci NLFSR equivalente. Esto se debe a que una secuencia de salida de cada Fibonacci NLFSR puede ser descrita por una recurrencia no lineal de orden \( n \), mientras que para un \( n \)-bit Galois NLFSR dicha recurrencia puede no existir[24].

**Nota:** Las siguientes definiciones son tomadas de [24].

**Definición 3:** El grafo de retroalimentación de un \( n \)-bit NLFSR es un grafo dirigido con \( n \) vértices \( v_0, \ldots, v_{n-1} \) los cuales representan los bits \( \{0,1, \ldots, n-1\} \) de un NLFSR...
respectivamente. Por otra parte, hay una arista \((v_i, v_j)\) que va desde \(v_i\) hacia \(v_j\) si \(i \in \text{dep}(f_j)\).

Ej.: Suponga que la función actualizadora de \(f_j\) es \(f_j = x_{j+1} \oplus x_i\), la cual posee el siguiente conjunto de dependencia \(\text{dep}(f_j) = \{x_{j+1}, x_i\}\), cumpliendo así la definición anterior \(i \in \text{dep}(f_j)\). Entonces el grafo se vería así:

Figura 5.15 Ilustración definición 1.

**Definición 4:** La operación sustitución, denotada de forma general como \(\text{sub}(v_i, v_j)\), está definida por cualquier vértice \(v_i\) el cual solo tiene un único predecesor \(v_j\). La sustitución \(\text{sub}(v_i, v_j)\) remueve \(v_i\) del grafo de retroalimentación y para cada sucesor \(v_k\) de \(v_i\), reemplaza la arista \((v_i, v_k)\) por una arista \((v_j, v_k)\) [Fig. 5.16].

Figura 5.16 Ilustración definición 2.

**Definición 5:** Dado un grafo de retroalimentación \(G\), el grafo reducido de \(G\) es un grafo obtenido al aplicar repetidamente la sustitución en cada vértice de \(G\), hasta que no se puedan aplicar más sustituciones.

---

\(^{34}\) La operación \(\text{sub}(v_i, v_j)\) de la definición 2, tiene un significado distinto a la arista que une dos vértices \((v_i, v_j)\) de la definición 1.
**Lema:** Si el grafo de retroalimentación de un n-bit NLFSR puede reducirse hasta un único vértice, entonces existe una recurrencia no lineal de orden $n$ describiendo la secuencia de valores del bit $i$.

**Demostración:**

Sea $v_i$ un vértice de un grafo de retroalimentación el cual tiene un único predecesor $v_j$, y $x$ sucesores $v_{k_1}, \ldots, v_{k_x}$, donde cada $k_y \in \{0,1, \ldots, n-1\}$, $y \in \{1,2, \ldots, x\}$ (Fig. 5.17).

![Diagrama](image)

Figura 5.17 Ilustración definición 3.

Por definición 4, si $v_i$ solo tiene un predecesor $v_j$, quiere decir que el valor actual del registro $s_i$ en un tiempo $t$ es igual a $s_i(t) = s_i(t-1)$, y cada sucesor $v_{k_y}(t)$ depende de $s_i(t-1)$.

Por consiguiente la sustitución $sub(v_i, v_j)$ reemplaza la variable $s_i(t-1)$ en la ecuación de cada $s_{k_y}(t)$ por $s_i(t-2)$, eliminando de esta manera una variable y por consiguiente una ecuación que equivaldría a una función actualizadora. Al aplicar reiteradamente este proceso se logra reducir el grafo a un solo vértice describiendo una secuencia de estados en términos de un solo bit.
En el estudio [18], se reduce el STG (Fig. 5.18) de un NLFSR, con una función de retroalimentación \( f_3 = x_0 \oplus x_1 \oplus x_2 \oplus x_1x_3 \), y cuya secuencia de estados se puede describir a través de las siguientes ecuaciones:

\[
\begin{align*}
  s_3(t) &= s_0(t - 1) \oplus s_1(t - 1) \oplus s_2(t - 1) \oplus s_1(t - 1)s_3(t - 1) \\
  s_2(t) &= s_3(t - 1) \\
  s_1(t) &= s_2(t - 1) \\
  s_0(t) &= s_1(t - 1)
\end{align*}
\]

Siguiendo los pasos del algoritmo como se muestra a continuación:

![Figura 5.18 Pasos para reducir un feedback graph de un Fibonacci NLFSR.](image)

1. Grafo inicial, b) después de aplicar \( sub(v_0, v_1) \), c) después de aplicar \( sub(v_1, v_2) \), d) después de aplicar \( sub(v_2, v_3) \).

2. \( sub(v_0, v_1) \) reduce el grafo a la Figura 5.18(b). Esto es equivalente a sustituir \( s_0(t) \) por \( s_1(t - 1) \) en \( s_3 \) como se indica a continuación:

\[
  s_3(t) = s_1(t - 2) \oplus s_1(t - 1) \oplus s_2(t - 1) \oplus s_1(t - 1)s_3(t - 1)
\]

3. \( sub(v_1, v_2) \) reduce el grafo a la Figura 5.18 (c). Esto es equivalente a sustituir \( s_1(t) \) por \( s_2(t - 1) \) en \( s_3 \) como se indica a continuación:

\[
  s_3(t) = s_2(t - 3) \oplus s_2(t - 2) \oplus s_2(t - 1) \oplus s_2(t - 2)s_3(t - 1)
\]

4. \( sub(v_2, v_3) \) reduce el grafo a la Figura 5.18 (d). Esto es equivalente a sustituir \( s_2(t) \) por \( s_3(t - 1) \) en \( s_3 \) como se indica a continuación:

\[
  s_3(t) = s_3(t - 4) \oplus s_3(t - 3) \oplus s_3(t - 2) \oplus s_3(t - 3)s_3(t - 1)
\]

Esta última arroja la recurrencia no lineal que describe la secuencia de valores del bit 3.
Otro NLFSR presente en [35], posee una secuencia máxima de estados como se observa en la (Fig. 5.19), y no se le ha aplicado el proceso de reducción36, sus funciones actualizadoras se muestran a continuación:

\[
\begin{align*}
    s_3(t) &= s_0(t - 1) \oplus s_1(t - 1) \\
    s_2(t) &= s_3(t - 1) \\
    s_1(t) &= s_2(t - 1) \\
    s_0(t) &= s_1(t - 1)
\end{align*}
\]

1. \(\text{sub}(v_0, v_1)\) reduce el grafo a la Figura 5.18 (b). Esto es equivalente a sustituir 
   \(s_0(t)\) por \(s_1(t - 1)\) en \(s_3\) como se indica a continuación:
   \[
   s_3(t) = s_1(t - 2) \oplus s_1(t - 1)
   \]
2. \(\text{sub}(v_1, v_2)\) reduce el grafo a la Figura 5.18 (c). Esto es equivalente a sustituir 
   \(s_1(t)\) por \(s_2(t - 1)\) en \(s_3\) como se indica a continuación:
   \[
   s_3(t) = s_2(t - 3) \oplus s_2(t - 2)
   \]
3. \(\text{sub}(v_2, v_3)\) reduce el grafo a la Figura 5.18 (d). Esto es equivalente a sustituir 
   \(s_2(t)\) por \(s_3(t - 1)\) en \(s_3\) como se indica a continuación:
   \[
   s_3(t) = s_3(t - 4) \oplus s_3(t - 3)
   \]

---

36 La intención de hacer un ejemplo adicional es para visualizar que este algoritmo se puede aplicar a diferentes NLFSRs.
De esta manera, al comprobar que un STG se puede describir como una recurrencia no lineal de una sola variable, se procede a aplicar la transformación de Fibonacci a Galois.

Así quedaría descrito de forma matemática:

Sea $s_i$ la función que denota el valor del estado de la variable $x_i$ del bit $i$ en un tiempo $t$. La secuencia de estados de un NLFSR de $n$-bits con funciones de retroalimentación singulares puede describirse por un sistema de $n$ ecuaciones no lineales como las que se muestran a continuación:

$$
\begin{align}
    s_{n-1}(t) &= s_0(t-1) \oplus g_{n-1}(s_1(t-1), s_2(t-1), ..., s_{n-1}(t-1)) \\
    s_{n-2}(t) &= s_{n-1}(t-1) \oplus g_{n-2}(s_0(t-1), s_1(t-1), ..., s_{n-2}(t-1)) \\
    &\vdots \\
    s_0(t) &= s_1(t-1) \oplus g_0(s_0(t-1))
\end{align}
$$

Estas ecuaciones están indicando la secuencia de estados que tiene cada variable en un tiempo $t$.

En términos generales, con estas funciones lo que se busca lograr es definir las variables en términos de sus estados anteriores para diferentes valores de $t$, por ejemplo:
El valor del registro \( s_0 \) en un LFSR de \( n \) registros (Fig. 5.20), se puede escribir en términos de la misma variable a través del tiempo. 

\[ s_0 = s_1(t - 1) \]
\[ s_0 = s_2(t - 2) \]
\[ \ldots \]
\[ s_0 = s_{n-1}(t - n - 1) \]

Esto significa que cada vez que haya un corrimiento, el valor de \( s_0 \) será el mismo que haya tenido \( s_1, \ldots, s_{n-1} \) en un tiempo \( t \) dado.

No obstante, en (8) las variables se encuentran en términos de \( (t - 1) \). Por ejemplo, en la ecuación:

\[ s_0(t) = s_1(t - 1) \oplus g_0(s_0(t - 1)) \]

La secuencia de valores para \( s_0(t) \) que pueden describirse en términos de \( (t - 1) \) únicamente podría ser \( s_0(t - 1) \), no puede ser \( s_1(t - 1) \) debido a (6), y tampoco puede ser una variable mayor a \( s_1 \) porque no estaría en términos de \( (t - 1) \). Por ejemplo, para \( s_2, s_0(t) = s_2(t - 2) \). No obstante, a medida que el índice de cada variable \( s_i \) aumenta, los valores de los que depende \( g_i \) también aumentarán, puesto que existen más variables que puedan describirse en términos de \( (t - 1) \).

**Definición 6**: Sea \( f_a \) y \( f_b \) las funciones de retroalimentación de los bits \( a \) y \( b \) de un NLFSR. La operación de corrimiento denotada como \( f_a P \rightarrow f_b \), mueve un monomio \( P \subseteq A f_a \) desde el ANF de \( f_a \) hasta el ANF de \( f_b \). El índice de cada variable \( x_i \) de cada monomio en \( P \) se actualiza como \( x_{(i-a+b)mod n} \), donde \( i \) representa el índice del monomio que se va a desplazar, \( a \) representa el índice de la función donde originalmente se encuentra el monomio, \( b \) representa el índice de la función hasta donde se va a desplazar el monomio.
Ejemplo\(^3\): Observe las siguientes funciones actualizadoras

\[ f_1 = x_2 \oplus x_3 \]
\[ f_0 = x_1 \]

Si se desea desplazar el monomio \( x_3 \) de \( f_1 \) a \( f_0 \) ([\( f_1 x_3 f_0 \)], de acuerdo a la notación, este cambiaría así:

\[ f_1 = x_2 \]
\[ i = 3, \ a = 1, \ b = 0, \ x_{(i-a+b)} mod n = x_{(3-1+0)} mod n = x_2 \]

Finalmente \( f_0 \) quedaría como:

\[ f_0 = x_1 \oplus x_2 \]

Con el fin de preservar la equivalencia de un NLFSR después de un corrimiento, este desplazamiento no se puede realizar hacia una función \( f_i \) tal que \( i < \tau \).

Nota: El min\_index\( (f) \) y el max\_index\( (f) \) denotan el índice más pequeño y más grande de las variables en dep\( (f) \) respectivamente.

Ejemplo: Si \( f = x_1 \oplus x_2 x_3 x_4 \), el min\_index\( (f) \) = 1, y el max\_index\( (f) \) = 4

Así, \( \tau \) se calcula como\(^2\):

\[ \tau = \max(\max\_index\( (f) \) - \min\_index\( (f) \)) \forall f \in F, \]

Donde \( F \) es el conjunto de todos los monomios de las funciones de retroalimentación de un Fibonacci NLFSR.

Ejemplo:

Calcular el bit \( \tau \) de una función cuyos monomios son los siguientes: [An Improved Implementation of Grain]

\[ f_{15} = x_0 \oplus x_{12} x_{13} \oplus x_7 \oplus x_4 x_7 x_9 \oplus x_2 x_5 \oplus x_2 x_3 \oplus x_{10} x_{14} \oplus x_6 x_9 x_{11} \oplus x_6 x_7 x_{10} x_{12} \oplus x_1 \]
\[ \tau = \max(13 - 12, 9 - 4, 5 - 2, 3 - 2, 14 - 10, 11 - 6, 12 - 6) \]
\[ \tau = \max(1, 5, 3, 1, 4, 5, 6) \]
\[ \tau = 6 \]

Este valor \( \tau \) está indicando que cuando se haga un corrimiento, el máximo desplazamiento que se podrá hacer será \( f_{15} \rightarrow f_6 \), y para el resto de funciones con

\(^3\) Para el ejemplo suponga el valor de \( n>4 \)
$i < \tau$, estas deben ser del tipo (4). Siguiendo el ejemplo, el resto de las funciones menores a 6 quedarían así:

$$f_5 = x_6$$

$$\ldots$$

$$f_0 = x_1$$

Para el resto de funciones se desplazarán los monomios de acuerdo a la definición 6.

Para desplazar algunos de los monomios a modo de ejemplo, los cuales se seleccionan de forma arbitraria, se partirá desde $f_{15}$ hasta cualquiera de las otras funciones con $i \geq \tau$. Los monomios que se seleccionen en cada paso se resaltarán en rojo, para mayor comprensión.

La actualización de los índices se realiza a través de $x_{(i-a+b) \mod n}$.

1. $f_{15} \xrightarrow{x_6x_7x_{10}x_{12}} f_{14}$.

   $f_{15} = x_0 \oplus x_{12}x_{13} \oplus x_7 \oplus x_4x_7x_9 \oplus x_2x_5 \oplus x_2x_3 \oplus x_{10}x_{14} \oplus x_6x_9x_{11} \oplus x_6x_7x_{10}x_{12} \oplus x_1$

   $x_6x_7x_{10}x_{12} =$

   $$x_{(6-15+14) \mod n} = x_5,$$

   $$x_{(7-15+14) \mod n} = x_6,$$

   $$x_{(10-15+14) \mod n} = x_9,$$

   $$x_{(12-15+14) \mod n} = x_{11},$$

   Así, $f_{14} = x_{15} \oplus x_5x_6x_9x_{11}$

2. $f_{15} \xrightarrow{x_6x_9x_{11}} f_{13}$. 
\[ f_{15} = x_0 \oplus x_{12}x_{13} \oplus x_7 \oplus x_4x_7x_9 \oplus x_2x_5 \oplus x_2x_3 \oplus x_{10}x_{14} \oplus x_6x_9x_{11} \oplus x_1 \]

\[ x_6x_9x_{11} = \]

\[ x_{(6-15+13)} mod n = x_4, \]

\[ x_{(9-15+13)} mod n = x_7, \]

\[ x_{(11-15+13)} mod n = x_9, \]

Así, \( f_{13} = x_{14} \oplus x_4x_7x_9 \)

3. \( f_{15} \rightarrow f_6 \)

\[ f_{15} = x_0 \oplus x_{12}x_{13} \oplus x_7 \oplus x_4x_7x_9 \oplus x_2x_5 \oplus x_2x_3 \oplus x_{10}x_{14} \oplus x_1 \]

\[ x_{12}x_{13} = \]

\[ x_{(12-15+6)} mod n = x_3, \]

\[ x_{(13-15+6)} mod n = x_4 \]

Así, \( f_6 = x_7 \oplus x_3x_4 \)

... 

Este proceso se repite tantas veces de acuerdo al objetivo deseado, se podría llegar al caso en que \( f_{15} \) solo quedara expresada como:

\[ f_{15} = x_0 \oplus x_1 \]

Y el resto de los monomios distribuidos entre las funciones \( f_i \) con \( i \geq \tau \).

**Definición 7**: Un NLFSR es uniforme si el bit terminal \( \tau \) del NLFSR satisface las siguientes condiciones:
a) Todas las funciones de retroalimentación tal que $i < \tau$ son singulares del tipo (4), y
b) Para todos los bits $i$, tal que $i > \tau$, la siguiente condición debe cumplirse:

$$\text{max\_index}(g_i) \leq \tau.$$  

*Lemma 1:* Si un NLFSR es uniforme, entonces su grafo de retroalimentación puede reducirse a un único vértice de acuerdo a las definiciones 2 y 3.

Para probar este lemma, se retoma el concepto para la reducción de grafo de retroalimentación, el cual puede reducirse a un solo vértice.

Ya se ha visto cómo se calcula el bit terminal $\tau$ de un NLFSR. Ahora suponga que se desea reducir un NLFSR cualquiera (Fig. 5.21), siguiendo los conceptos anteriores se podría tener un ejemplo como el que se muestra a continuación.

![Diagrama de NLFSR reducido](image)

Figura 5.21 Reducción de un NLFSR cualquiera con el valor Tao identificado (1).

Observe que $f_2, f_3$ y $f_0$ son de la forma $f_i = x_{i+1}$, por lo tanto se pueden reducir hasta el bit terminal $\tau$ que en este caso estaría ubicado en la función $f_3$, eso quiere decir que al reducirlo por definición 4, el origen de todas las aristas estarían saliendo desde $f_3$, el cual se visualizaría de la siguiente manera:

---

38 El bit terminal $\tau$ del ejemplo, en realidad no es el 3, pero se escogió así para visualizar el concepto.
De aquí se puede observar que todos los registros con valores de $i \in \{\tau + 1, \tau + 2, ..., n - 1\}$ reciben solo dos aristas, uno que viene de $x_{i+1}$ y otra de $v_\tau$, a excepción de $n - 1$. Por lo tanto, por definición 5, ahora se puede aplicar la reducción $sub(v_{n-1}, v_\tau)$, quedando así:

Al aplicar reiteradamente este proceso, $sub(v_{n-2}, v_\tau), ..., sub(v_{\tau+1}, v_\tau)$, se puede reducir el grafo a $v_\tau$, el cual quedaría como:
Lemma 2: Dado un NLFSR uniforme con un bit terminal $a$, el corrimiento $f_a^P f_b, P \subseteq A_{f_a}, b < a$, da como resultado un NLFSR equivalente si el NLFSR transformado también es uniforme.

Suponga que el NLFSR transformado es uniforme, por consiguiente, de acuerdo al lemma 2, este puede reducirse a un solo vértice $v_b$ que corresponde al bit terminal $b$ del NLFSR transformado después del corrimiento $f_a^P f_b$. Por lo tanto, por el lemma 1 y la definición 5, existe una relación de recurrencia no lineal describiendo la secuencia de valores del bit $b$. Solo resta probar que esta recurrencia es la misma del NLFSR inicial. Para ello, es suficiente considerar el caso cuando un único monomio $p \in A_{f_a}$ del tipo $p = x_k x_a$, para algún $k < a$ desplazado desde $g_a$ hasta $g_b$, con $b < a$.

Si el monomio desplazado es $x_k x_a$, entonces la función $g_a^{39}$ puede ser representada como

$$g_a = g_a^* \oplus x_k x_a,$$

donde $g_a^*$ representa el conjunto de dependencia de las todas funciones $g_i$ antes de realizar el corrimiento del monomio $x_k x_a$, y $g_a$ representa a la función $f_a$ después del corrimiento. Por ejemplo: Observe la siguiente función de retroalimentación.

$$f_2 = x_3 \oplus x_1 x_0 \oplus x_k x_a$$

$$f_1 = x_2$$

Al aplicar el desplazamiento $f_a^P f_b \rightarrow f_a^{x_k x_a} f_b$, de acuerdo a la descripción anterior, $g_a^*$ quedaría expresado así:

$$f_2 = x_3 \oplus f_2^* \oplus x_k x_a,$$

para este ejemplo $f_2^* = x_1 x_0$.

---

39 La función $g_a^*$ representa el conjunto de dependencia de las todas funciones $g_i$ antes de realizar el corrimiento del product-term $x_k x_a$, y la función $g_a$ representa el estado de las funciones $g_i$ después de realizar el desplazamiento.
Por lo tanto, el NLFSR antes del corrimiento puede representarse por el siguiente conjunto de ecuaciones:

\[
\begin{align*}
    s_{n-1}(t) &= s_0(t - 1) \oplus g_{n-1}(s_0(t - 1), s_1(t - 1), ..., s_b(t - 1)) \\
    s_a(t) &= s_{a+1}(t - 1) \oplus g_a^*(s_0(t - 1), s_1(t - 1), ..., s_b(t - 1)) \oplus s_k(t - 1)s_a(t - 1) \\
    s_{a-1}(t) &= s_a(t - 1) \\
    s_0(t) &= s_1(t - 1)
\end{align*}
\] (9)

Observe que \(g_a^*\) está presente en la ecuación de \(s_a(t)\), indicando que sobre esa función se realizará el corrimiento del product-term \(s_k(t - 1)s_a(t - 1)\).

Note además que cada una de las funciones \(g_{n-1}, g_{n-2}, ..., g_a\) depende de variables con índices menores o iguales a "b" ya que este es el bit terminal, o dicho de otra manera, este es el índice de la función hasta donde se puede hacer un desplazamiento desde \(a\), por ello al hacerse el corrimiento debe preservarse la condición 2 de la definición 7. En este caso \(\tau = b\).

La sustitución \(sub(v_i, v_{i+1})\), de acuerdo a la definición 5, es equivalente a reemplazar la variable \(s_i(t - 1)\) en la ecuación de cada sucesor de \(v_i\) por \(s_{i+1}(t - 2)\). Después de una secuencia de \(a\) sustituciones \(sub(v_0, v_1), ..., sub(v_{a-1}, v_a)\). Cada \(s_i(t - 1)\) es reemplazado por:

\[s_a(t - 1 - (a - i))\]

Así, el sistema de ecuaciones (9) queda reducido de la siguiente manera:

\[
\begin{align*}
    s_{n-1}(t) &= s_a(t - a - 1) \oplus g_{n-1}(s_a(t - a - 1), s_a(t - a), ..., s_a(t - 1 - (a - b))) \\
    s_a(t) &= s_{a+1}(t - 1) \oplus s_a(t - 1 - a + k)s_a(t - 1) \\
    &\quad \oplus g_a^*(s_a(t - a - 1), s_a(t - a), ..., s_a(t - 1 - (a - b)))
\end{align*}
\] (10)
El conjunto de dependencia de \( g_{n-1} \) en \( s_{n-1}(t) \), se obtiene variando los valores de \( i \) (con valores menores e iguales que \( b \)), y dado que el corrimiento se hace desde \( a \) hasta \( b \) \((f_a \rightarrow f_b)\), \( a \) no puede ser menor que \( b \), en un caso básico se puede observar que al realizarse un desplazamiento de \( f_a \rightarrow f_b \) con \( i = 0 \), reemplazándolo en \( s_a(t - 1 - (a - i)) \) se obtiene:

\[
s_a(t - 1 - (a - 0)) = s_a(t - a - 1)
\]

\( i = 1 \)

\[
s_a(t - 1 - (a - 1)) = s_a(t - a)
\]

\( i = 2 \)

\[
s_a(t - 1 - (a - 2)) = s_a(t - a + 1)
\]

\( i = b \)

\[
s_a(t - 1 - (a - b)) = s_a(t - 1 - (a - b))
\]

De esta manera se puede observar que toda la ecuación queda descrita en términos de \( a \), que es lo que se desea para eliminar la dependencia entre variables, por ello, a \( s_a(t) = s_{a+1}(t - 1) \) en (8) no se le aplica la sustitución \( s_a(t - 1 - (a - i)) \), porque en ese segmento de la ecuación la variable ya aparece en términos de \( a \), mientras que en \( g_a \) hay variables en términos de \( b \), por consiguiente, en ese segmento si debe aplicarse la sustitución. Adicionalmente, en (9), se puede observar que de los términos \( s_k(t - 1)s_a(t - 1) \) los cuales corresponden al monomio que se va a mover desde \( a \) hasta \( b \), solo uno de ellos aparece en términos de \( a \), esto quiere decir que sobre ese segmento no debe aplicarse la sustitución, en cambio \( s_k(t - 1) \) se encuentra en términos de \( k \) si. Al finalizar estos términos quedan expresados como \( s_a(t - 1 - a + k)s_a(t - 1) \). Para este término en particular se debe tener en cuenta que el valor que varía cuando se realiza el desplazamiento es \( k \) y no \( i \) por lo tanto la sustitución es \((t - 1 - (a - k))\) en vez de \((t - 1 - (a - i))\).
De esta manera se obtienen las ecuaciones descritas en (10).

Para reducir las expresiones en (9) se utilizará la siguiente abreviación:

\[ \tilde{s}_a := (s_a(t - a - 1), s_a(t - a), ..., s_a(t - 1 - (a - b))) \]

Y la notación \( \tilde{s}_a(i) \) significa que cada elemento \( s_a(x) \) de \( \tilde{s}_a \) se reemplaza por \( s_a(x + i) \). Por ejemplo,

\[ \tilde{s}_a(-1) = (s_a(t - a - 2), s_a(t - a - 1), ..., s_a(t - 2 - (a - b))) \]

Explicación: Cada una de las componentes de \( s_a \),

\( (s_a(t - a - 1), s_a(t - a), ..., s_a(t - 1 - (a - b))) \)

Son los \( s_a(x) \) de \( \tilde{s}_a \), y "i" es \(-1\), entonces al remplazarlo en las variables se expresarán así:

\[ \tilde{s}_a(-1) = (s_a(t - a - 1 - (1)), s_a(t - a - (1)), ..., s_a(t - 1 - (a - b) - (1))) \]

Y al terminar de realizar las operaciones se obtiene:

\[ \tilde{s}_a(-1) = (s_a(t - a - 2), s_a(t - a - 1), ..., s_a(t - 2 - (a - b))) \]

Dicho esto, el sistema de ecuaciones (10) puede ser reescrito de la siguiente manera:

\[ s_{n-1}(t) = s_a(t - a - 1) \oplus g_{n-1}(\tilde{s}_a) \]

\[ ... \]

\[ s_a(t) = s_{a+1}(t - 1) \oplus s_a(t - 1 - a + k) s_a(t - 1) \oplus g_a^*(\tilde{s}_a) \]

Después de una secuencia de \( n - a - 1 \) sustituciones \( sub(v_{n-1}, v_{n-2}), ..., sub(v_{a+1}, v_a) \), se consigue una recurrencia no lineal describiendo los valores del bit \( a \), utilizando la expresión:

\[ \tilde{s}_a(n - a - 1) := (s_a(t - a - 1), s_a(t - a), ..., s_a(t - 1 - (a - b))) \]
Para obtener:

\[ s_a(t) = s_a(t - n) \oplus g_{n-1}(\tilde{s}_a(-n + a + 1)) \oplus g_{n-2}(\tilde{s}_a(-n + a)) \oplus ... \oplus g_a(\tilde{s}_a) \]
\[ \oplus s_a(t - 1 - a + k)s_a(t - 1) \]

Explicación: al aplicar

\[ \tilde{s}_a(n - a - 1) := (s_a(t - a - 1), s_a(t - a), ..., s_a(t - 1 - (a - b))) \]

Se puede observar que al reemplazar \( n - a - 1 \) en \( t - a - 1 \) se obtiene:

\[ (t - a - 1 - (n - a - 1)) = (t - a - 1 - n + a + 1) = (t - n) \]

\[ (t - a - (n - a - 1)) = (t - a - n + a + 1) = (t - n + 1) \]

\[ ... \]

\[ (t - 1 - (a - b) - (n - a - 1)) = (t - 1 - a + b - n + a + 1) = (t - n + b) \]

\[ (t - 1 - (a + 1) - (n - a - 1)) = (t - 1 - a - 1 - n + a + 1) = (t - n - 1) \]

\[ (t - 1 - (a - b + 1) - (n - a - 1)) = (t - 1 - a + b - 1 - n + a + 1) \]

\[ = (t - n + b - 1) \]

Y así sucesivamente, para cada valor de \( s_a \).

Después de expandir la abreviación \( \tilde{s}_a \), la recurrencia queda expresada como:

\[ s_a(t) = s_a(t - n) \oplus g_{n-1}(s_a(t - n), s_a(t - n + 1), ..., s_a(t - n + b)) \]
\[ \oplus g_{n-2}(s_a(t - n - 1), s_a(t - n), ..., s_a(t - n + b - 1)) \oplus ... \]
\[ \oplus g_a(s_a(t - a - 1), s_a(t - a), ..., s_a(t - 1 - a + b)) \]
\[ \oplus s_a(t - 1 - a + k)s_a(t - 1) \]

\[ ^{40} \text{Este componente que acompaña a los } \tilde{s}_a, \text{ son factores comunes dentro de cada expresión de } \tilde{s}_a, \text{ que al expandirse se obtiene el resultado que se muestra en el párrafo siguiente.} \]

68
Hasta este momento se ha realizado el análisis del NLFSR para demostrar que este puede quedar descrito en términos de usa sola variable. El siguiente paso a realizar es aplicar el mismo proceso al NLFSR después de que se realizó el corrimiento, el cual puede ser representado por el siguiente sistema de ecuaciones:

\[ s_{n-1}(t) = s_0(t-1) \oplus g_{n-1}(s_0(t-1),s_1(t-1),...,s_b(t-1)) \]

... 

\[ s_a(t) = s_{a+1}(t-1) \oplus g_a(s_0(t-1),s_1(t-1),...,s_b(t-1))^{41} \]

\[ s_{a-1}(t) = s_a(t-1) \]

... 

\[ s_b(t) = s_{b+1}(t-1) \oplus s_{k-(a-b)}(t-1)s_b(t-1)^{42} \]

... 

\[ s_0(t) = s_1(t-1) \]

Después de b sustituciones \( sub(v_0,v_1),...,sub(v_{b-1},v_b) \), se utiliza el mismo procedimiento de sustitución usado antes del corrimiento el cual está definido por \( s_b(t-1-(b-i)) \) para obtener lo siguiente:

\[ i = 0 \]

\[ s_b(t-1-(b-0)) = s_b(t-b-1) \]

---

41 Observe que este \( g_a \) ya no es \( g_a^* \), esto se debe a lo que el corrimiento ya se realizó y por lo tanto el monomio ya no se encuentra presente.

42 La actualización de los índices se realiza a través de \( x_{(i-a+b)} mod n \), como se describió en una sección anterior.
\[ i = 1 \]
\[ s_b(t - 1 - (b - 1)) = s_b(t - b) \]

\[ i = 2 \]
\[ s_b(t - 1 - (b - 2)) = s_b(t - b + 1) \]

\[ i = b \]
\[ s_b(t - 1 - (b - b)) = s_a(t - 1 - (b - b)) = s_b(t - 1) \]

Con estas componentes, se llega al siguiente sistema de ecuaciones expresadas en términos de \( b \):

\[ s_{n-1}(t) = s_b(t - b - 1) \oplus g_{n-1}(s_b(t - b - 1), s_1(t - b), ..., s_b(t - 1)) \]

...\[ s_a(t) = s_{a+1}(t - 1) \oplus g_a^*(s_b(t - b - 1), s_1(t - b), ..., s_b(t - 1))^{43} \]

\[ s_{a-1}(t) = s_a(t - 1) \]

...

\[ s_b(t) = s_{b+1}(t - 1) \oplus s_b(t - 1 - a + k)s_b(t - 1) \]

Para reducir la longitud de las expresiones se utilizará la siguiente abreviación:

\[ \overline{s}_b := (s_b(t - b - 1), s_b(t - b), ..., s_b(t - 1))^{44} \]

Con esto se puede reescribir el sistema de ecuaciones así:

---

43 Acá vuelve a aparecer \( g_a^* \) dado que es el nuevo \( g_a \) a partir del cual se pueden realizar futuros corrimientos.

44 En esta expresión no aparece \( -(a - b) \) presente en \( \overline{s}_a \) debido a que la expresión tomada acá es \( (t - 1 - (b - i)) \), y cuando \( i \) tome el valor de \( b \), esta se cancela.
\[
  s_{n-1}(t) = s_b(t - b - 1) \oplus g_{n-1}(\bar{s}_b)
  
  ... 
  
  s_a(t) = s_{a+1}(t - 1) \oplus g_a^*(\bar{s}_b)
  
  s_{a-1}(t) = s_a(t - 1)
  
  ... 
  
  s_b(t) = s_{b+1}(t - 1) \oplus s_b(t - 1 - a + k)s_b(t - 1)
\]

Después de una secuencia de \( n - b - 1 \) sustituciones

\( \text{sub}(v_{n-1}, v_{n-2}), ..., \text{sub}(v_{b+1}, v_b) \), se consigue una recurrencia no lineal
describiendo los valores del bit \( b \) igual a la encontrada con el bit \( a \).

\[
  s_b(t) = s_b(t - n) \oplus g_{n-1}(\bar{s}_b(-n + b + 1)) \oplus g_{n-2}(\bar{s}_b(-n + b)) \oplus ...
  
  \oplus g_a^*(\bar{s}_b(-(a - b) \text{\footnote{A diferencia del } g_a^* \text{ en } s_a, \text{ aquí aparece el término } -(a - b) \text{, pero solo porque en } s_b \text{ aparece como factor común, sin embargo, al expandir las abreviaciones los resultados obtenidos en } s_a \text{ y } s_b \text{ son iguales.}})) \oplus s_b(t - 1 - a + k)s_b(t - 1)
\]

De igual manera, después de expandir la abreviación \( \bar{s}_b \), la recurrencia queda
derivada como:

\[
  s_b(t) = s_b(t - n) \oplus g_{n-1}(s_b(t - n), s_b(t - n + 1), ..., s_b(t - n + b))
  
  \oplus g_{n-2}(s_b(t - n - 1), s_b(t - n), ..., s_b(t - n + b - 1)) \oplus ... \quad (12)
  
  \oplus g_a(s_b(t - a - 1), s_b(t - a), ..., s_b(t - 1 - a + b))
  
  \oplus s_b(t - 1 - a + k)s_b(t - 1)
\]

Con esto se tiene que las recurrencias no lineales (11) y (12) son las mismas, por
consiguiente, los dos NLFSRs son equivalentes.

**Caso especial de transformación Galois NLFSR a Galois NLFSR.**
Hasta el momento se ha trabajado en realizar una transformación desde un NLFSR en configuración Fibonacci a un NLFSR en configuración Galois. El siguiente paso a seguir es extender dicha transformación esta vez de un NLFSR que ya se encuentra en Galois, a otro NLFSR también en configuración Galois pero esta vez para hacerlo más eficiente.

Esta transformación no presenta grandes variaciones con respecto a la presentada anteriormente, solo requiere de algunos ajustes para extender dicha transformación.

Como se ha indicado, un shift register de n bits o n-bit shift register realiza un mapeo de un estado a otro \( \{0,1\}^n \to \{0,1\}^n \), el cual se puede representar de la siguiente manera:

\[
\begin{pmatrix}
  x_0 \\
  \vdots \\
  x_{n-1}
\end{pmatrix}
\begin{pmatrix}
  f_0(x_0, \ldots, x_{n-1}) \\
  \vdots \\
  f_{n-1}(x_0, \ldots, x_{n-1})
\end{pmatrix}
\]

Donde cada función \( f_i, i \in \{0,1, \ldots, n-1\} \) es del tipo:

\[
f_i = x_{i+1} \oplus g_l\{x_0, \ldots, x_i, x_{i+2}, \ldots, x_{n-1}\}
\]

Esta transformación sigue preservando tanto la invertibilidad como el periodo.

También se mencionó que el gráfico de estado transición (STG) es un grafo dirigido donde los nodos representan los estados y las aristas las posibles transiciones entre estados, las cuales pueden representarse a través de n-tuplas.

Para que la extensión de esta transformación preserve la estructura del ciclo del STG se deben cumplir 3 condiciones:

1. El monomio de una función del tipo (4), no puede desplazarse a otra función. Solo se podrá realizar un desplazamiento sobre funciones del tipo (6).

   **Explicación:**

   Dado que las funciones que son del tipo \( f_i = x_{i+1} \), son funciones como por ejemplo:
\[ f_2 = x_3 \]
\[ f_1 = x_2 \]
\[ f_0 = x_1 \]

Esto quiere decir que no es posible desplazar el monomio \( x_3 \) a \( f_1 \) o \( f_0 \), porque la función \( f_3 \) no tendría ninguna fuente de actualización y sería visto como si un circuito quedara desconectado.

2. Un monomio puede moverse tanto a la derecha como a la izquierda \( x \) veces, siempre y cuando las funciones \( f_i \) a lo largo del movimiento hasta alcanzar \( x \) sean del tipo \((4)\) y el índice del monomio que se esté desplazando a medida que se actualiza no puede ser igual al índice de alguna de las funciones \( f_i \) por las que se va moviendo.

**Explicación:**
Para este punto se presentan dos situaciones particulares.
La primera indica lo siguiente: Un monomio puede moverse tanto a la derecha como a la izquierda \( x \) veces, siempre y cuando las funciones \( f_i \) a lo largo del movimiento hasta alcanzar \( x \) sean del tipo \((4)\) y el índice del monomio que se esté desplazando a medida que se actualiza no puede ser igual al índice de alguna de las funciones \( f_i \) por las que se va moviendo.

Para dar claridad en este punto se realizará un ejemplo: Observe el siguiente shift register.

![Shift Register Diagram](image)

Figura 5.25 Ilustración definición 2 Galois-Galois (1).

Cuyas funciones actualizadoras son:

\[ f_4 = x_0 \]
\[f_3 = x_4 \oplus x_2\]
\[f_2 = x_3\]
\[f_1 = x_2 \oplus x_0\]
\[f_0 = x_1\]

De este shift register únicamente las funciones \(f_4\), \(f_2\) y \(f_0\) son del tipo (4), ya que solo se están actualizando con los valores \(x_i\) arrojados por el registro inmediatamente anterior a ellos como se visualiza en las líneas rojas de las siguientes tres gráficas.

![Figura 5.26 Ilustración definición 2 Galois-Galois (2).](image)

![Figura 5.27 Ilustración definición 2 Galois-Galois (3).](image)

Teniendo esto presente, se puede ver con facilidad, que de acuerdo a la primera parte de la definición 2 de la transformación Galois-Galois, el monomio \(x_0\) de \(f_1\) solo se puede mover a la derecha hasta \(f_0\) o a la izquierda hasta \(f_2\).
Figura 5.28 Ilustración definición 2 Galois-Galois (4).

Lo segunda parte de la condición 2 indica que las funciones del tipo (6) a medida que se desplazan y su índice se actualiza, este no se pueden cruzar con un monomio de igual índice ya presente en la función. Para explicar este segmento se usara el mismo ejemplo para la parte $a$.

![Diagrama de LFSR](image)

Figura 5.29 Ilustración definición 2 Galois-Galois (5).

Como se indicó previamente, el monomio $x_0$ de $f_1$ solo se puede mover a la derecha hasta $f_0$ o a la izquierda hasta $f_2$. Sin embargo y dado que la estructura de un LFSR se puede ver de manera circular, suponga que este monomio se puede mover tantas veces a la izquierda como a la derecha se desee. Esto quiere decir que el monomio $x_0$ de $f_1$ podría llegar hasta $f_3$, no obstante, la condición dice que a medida que se actualice el índice del monomio que se desplaza, este no se puede cruzar con otro índice igual en otra función. Observe.

La función actualizadora para el índice es la siguiente: $x_{(i-a+b)mod n}$, y las funciones actualizadores del registro son las mismas que se ven en el shift register:

$$f_4 = x_0$$
$$f_3 = x_4 \oplus x_2$$
$$f_2 = x_3$$
$$f_1 = x_2 \oplus x_0$$
\[ f_0 = x_1 \]

Por lo tanto, al mover el monomio \( x_0 \) de \( f_1 \) a \( f_2 \) este quedaría como:

\[
i = 0, a = 1, b = 2, x_{(0-1+2)mod \ n} = x_1
\]

\[
f_4 = x_0 \\
f_3 = x_4 \oplus x_2 \\
f_2 = x_3 \oplus x_1 \\
f_1 = x_2 \\
f_0 = x_1
\]

Si este continúa con el desplazamiento quedaría como:

\[
i = 1, a = 2, b = 3, x_{(1-2+3)mod \ n} = x_2
\]

\[
f_4 = x_0 \\
f_3 = x_4 \oplus x_2 \oplus x_2 \\
f_2 = x_3 \\
f_1 = x_2 \\
f_0 = x_1
\]

Como puede observar el monomio que se desplaza tiene igual índice que otro de los monomios presente en la función \( f_3 \), y esta es una condición que no debe suceder dado que recorta la longitud del ciclo.

Ejemplo, un shift register cuyas funciones actualizadoras se muestran a continuación posee una m-sequence, como se ve en la tabla (5.3), partiendo desde un estado inicial (1,1,1,1).

\[
\begin{align*}
    f_3 &= x_0 \oplus x_3 \\
    f_2 &= x_3 \oplus x_1 x_2 \\
    f_1 &= x_2 \\
    f_0 &= x_1
\end{align*}
\]

Tabla 5.3 Estados de transición de un shift register.

<table>
<thead>
<tr>
<th>Estados</th>
<th></th>
<th>Salida</th>
</tr>
</thead>
<tbody>
<tr>
<td>X3</td>
<td>X2</td>
<td>X1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Fuente: Autores.

Sin embargo, al desplazar el monomio \( x_3 \) de \( f_3 \) a \( f_2 \) \([f_3 \rightarrow f_2]\), este se actualiza a \( x_2 \) el cual ya existe en la función destino, ya que las funciones actualizadoras quedarían modificadas de la siguiente manera:

\[
\begin{align*}
    f_3 &= x_0 \\
    f_2 &= x_3 \oplus x_1 x_2 \oplus x_2
\end{align*}
\]
\[ f_1 = x_2 \]
\[ f_0 = x_1 \]

Por lo tanto, al calcular nuevamente la salida con un estado inicial \((1,1,1,1)\) se obtiene que la secuencia queda reducida, como se observa en la tabla (5.4).

Tabla 5.4 Estados de transición reducidos de un shift register - 1. Fuente: Autores.

<table>
<thead>
<tr>
<th>Estados</th>
<th>Salida</th>
</tr>
</thead>
<tbody>
<tr>
<td>X3</td>
<td>X2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Para validar esto, se realizará el proceso comenzando con otro estado \((0,0,0,1)\), como se ve en la tabla 5.5.

Tabla 5.5 Estados de transición reducidos de un shift register - 2. Fuente: Autores.

<table>
<thead>
<tr>
<th>Estados</th>
<th>Salida</th>
</tr>
</thead>
<tbody>
<tr>
<td>X3</td>
<td>X2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Con esto queda ejemplificado el porque no se pueden cruzar dos monomios iguales durante el desplazamiento.

**Nota:** Inicialmente el shift register puede tener varios monomios iguales en una misma función actualizadora, esta condición aplica únicamente cuando se realice un desplazamiento.

3. El corrimiento definido como \( f_a^p \rightarrow f_b \), no puede cruzar las fronteras \( n - 1 \) y 0 del registro si \( f_a \) es un registro cuya salida se toma para generar la keystream, en caso contrario si se pueden cruzar.

   Este punto indica que si el registro sobre el cual se aplicará el movimiento posee un monomio que es utilizado para calcular la salida, este monomio en su desplazamiento no puede pasar las fronteras \( n - 1 \) y 0 del shift register.

   Por ejemplo: El siguiente shift register tiene unas funciones actualizadoras de la siguiente manera.

   \[
   \begin{align*}
   f_4 &= x_0 \\
   f_3 &= x_4 \oplus x_2 \\
   f_2 &= x_3 \\
   f_1 &= x_2 \\
   f_0 &= x_1 \oplus x_0
   \end{align*}
   \]

   Esto quiere decir que la salida de la cadena esta dada por \( x_0 \) y \( x_1 \) de \( f_0 \). Por lo tanto, si se desplaza el monomio \( x_0 \), este no puede cruzar las fronteras, es decir, solo se puede mover hasta \( f_4 \) hacia la izquierda siempre y cuando no viole alguna de las otras condiciones, pero no puede moverse hacia la derecha.
Nota: Si se quiere preservar una secuencia debido a que tiene buenas propiedades criptográficas, para cualquier corrimiento que se realice no puede cruzar las fronteras \( n - 1 \) y 0.

TRIVIUM

El Trivium \(^{46,47}\) es un cifrador de flujo síncrono orientado a hardware, su diseño fue pensado en crear un cifrador simple pero que no sacrificara la seguridad, velocidad y flexibilidad. Este algoritmo permite codificar la información bit a bit. Trivium posee un diseño enfocado en la flexibilidad, esto quiere decir que es ideal en ambientes con restricciones en el número de compuertas lógicas, eficiente en cuanto al consumo de potencia en plataformas con recursos limitados de potencia y rápido en aplicaciones que requieren cifrado de alta velocidad. Además genera una secuencia de hasta \( 2^{64} \) bits a partir de un vector de inicialización y una clave de 80 bits.

De acuerdo a las especificaciones del Trivium en \(^{33,34}\), la estructura de este puede verse como 3 registros de 84, 93 y 111 campos los cuales se cargan de la siguiente manera:

• En el registro de 93 bits se carga la llave secreta (Key) la cual es de 80 bits, el resto de campos se rellena con ceros.

\[(K_1, K_2, \ldots, K_{80}, 0, 0, \ldots, 0)\]

• En el registro de 84 bits se carga el vector de inicialización (Inicialization Vector) que también es de 80 bits, y el resto de campos se rellena con ceros.

\[(IV_1, IV_2, \ldots, IV_{80}, 0, 0, \ldots, 0)\]

---

46 C. De Canniere, B. Preneel, “Trivium Specifications”, en coordination & support action ECRYPT-CSA research network ECRYPT-NET, Nessie project.

- El registro de 111 bit se carga únicamente con ceros a excepción de los últimos 3 bits que son 1s.

\[(0,0,...,1,1,1)\]

El cifrado con este algoritmo es una tarea repetitiva en la cual se extraen 15 valores de unos campos específicos de los registros que se usan para actualizar otros 3 bits que finalmente servirán para computar la cadena de salida o key stream \(Z_i\).

Para llegar hasta el momento en que el algoritmo arroja la cadena de salida, el algoritmo presenta dos etapas. La primera consiste en la etapa de calentamiento o warm up phase y la segunda es la etapa de producción de bits criptográficamente seguros.

La warm up phase consiste en rotar 4 ciclos completos del algoritmo sin generar bits de salida. Cada ciclo del algoritmo realiza un recorrido por los 288 bits, por lo tanto la fase de calentamiento es realizar un ciclo hasta 1152, como se indica a continuación:
\[ for \ i = 1 \ to \ 4 * 288 \ do \]
\[ t_1 = x_{66} \oplus x_{91} x_{92} \oplus x_{93} \oplus x_{171} \]
\[ t_2 = x_{162} \oplus x_{175} x_{176} \oplus x_{177} \oplus x_{264} \]
\[ t_3 = x_{243} \oplus x_{286} x_{287} \oplus x_{288} \oplus x_{69} \]
\[ (x_1, x_2, \ldots, x_{93}) = (t_3, x_1, x_2, \ldots, x_{92}) \]
\[ (x_{94}, x_{95}, \ldots, x_{177}) = (t_1, x_{94}, x_{95}, \ldots, x_{176}) \]
\[ (x_{178}, x_{179}, \ldots, x_{288}) = (t_2, x_{178}, x_{179}, \ldots, x_{287}) \]
\[ end \ for \]

Una vez se ha ejecutado este proceso, la siguiente iteración (1253) produce el primer bit de salida de la key stream, esta es la fase de producción. Para esta tarea, el algoritmo en esencia permanece similar, solo se deben realizar pequeños ajustes para que arroje la salida como se muestra a continuación:

\[ for \ i = 1 \ to \ N \ do \]
\[ t_1 = x_{66} \oplus x_{93} \]
\[ t_2 = x_{162} \oplus x_{177} \]
\[ t_3 = x_{243} \oplus x_{288} \]
\[ Z_i = t_1 \oplus t_2 \oplus t_3 \]
\[ t_1 = t_1 \oplus x_{91} x_{92} \oplus x_{171} \]
\[ t_2 = t_2 \oplus x_{175} x_{176} \oplus x_{264} \]
\[ t_3 = t_3 \oplus x_{286} x_{287} \oplus x_{69} \]
\[(x_1, x_2, \ldots, x_{93}) = (t_3, x_1, x_2, \ldots, x_{92})\]

\[(x_{94}, x_{95}, \ldots, x_{177}) = (t_1, x_{94}, x_{95}, \ldots, x_{176})\]

\[(x_{178}, x_{179}, \ldots, x_{288}) = (t_2, x_{178}, x_{179}, \ldots, x_{287})\]

\textit{end for}

El esquema del trívium en su versión original se muestra a continuación:

Figura 5.30 Estructura del trívium propuesta por sus creadores C. De Canniere, B. Preneel.

Una manera que permite visualizar su estructura con mejor claridad es la siguiente:
La siguiente imagen (Fig.5.32) es equivalente a la anterior, en esta se observan los registros del 1 al 288. Estos valores se obtienen al sumar la longitud de cada registro con el valor específico a hallar.

Ejemplo: El registro 69 del shift register de 84 posiciones se calcula como:

$$93 + 69 = 162$$

Y así sucesivamente con los demás.
Por otra parte, de acuerdo a la notación utilizada en el esquema anterior, las funciones actualizadoras son del tipo:

$$f_i = x_{i-1}$$

Y teniendo en cuenta que las funciones actualizadoras hablando en términos de Fibonacci y Galois, estas deben ser del tipo:

$$f_i = x_{i+1}$$

Por este motivo, es necesario realizar una transformación en la notación de los registros como se indica a continuación:

Observe que la estructura del trívium pese a que teóricamente se analiza como tres registros, en la práctica es un solo registro de 288 bits, que puede ser visto de manera circular por lo tanto no hay inconveniente en rotar las posiciones de los tres registros siempre y cuando la estructura en general no se vea afectada, como se muestra en la Figura 5.33.
Por otra parte, como no todas las funciones actualizadoras del trívium son del tipo (4) se tiene que este no se encuentra en una configuración Fibonacci sino Galois.

Figura 5.33 Estructura del Trivium propuesta por Elena Dubrova.

En este gráfico se puede observar que los bits tomados inicialmente correspondían a los valores de los registros $x_{91}, x_{92}$ y $x_{93}$ pero ahora son $x_0, x_1$ y $x_3$.

La forma de calcularlos es la siguiente:

$93 - 66 = 27$, este es el número de desplazamientos que se deben hacer de derecha a izquierda en el trívium original para llegar a uno de los valores que se utilizan para la salida. No obstante, como la estructura del Trivium ha rotado y el que aparece de primero ahora es el registro de 111 bits, por lo tanto, como se ve en la Figura 5.33, el “27” que fue hallado previamente debe sumarse al valor de “195” para encontrar su posición equivalente así:
\[ 93 - 66 = 27, 27 + 195 = 222 \]
\[ 93 - 69 = 24, 24 + 195 = 219 \]

Este proceso se repite para los demás registros.

Registro de 111 bits:
\[ 288 - 243 = 45, 45 + 0 = 45 \]
\[ 288 - 264 = 24, 24 + 0 = 24 \]

Registro de 84 bits:
\[ 177 - 171 = 6, 6 + 111 = 117 \]
\[ 177 - 162 = 15, 15 + 111 = 126 \]

Antes de continuar se procede a realizar la siguiente notación.

Registro de 93 bits = **Registro A**.

Registro de 84 bits = **Registro B**.

Registro de 111 bits = **Registro C**.

Teniendo en cuenta esta notación, observe la siguiente relación:

El registro A, recibe 4 bits del registro C y 1 bit del registro A.

\[ f_{287} = x_0 \oplus x_1 x_2 \oplus x_{45} \oplus x_{219} \]

El registro B, recibe 4 bits del registro A y 1 bit del registro B.

\[ f_{194} = x_{195} \oplus x_{196} x_{197} \oplus x_{117} \oplus x_{222} \]

El registro C, recibe 4 bits del registro B y 1 bit del registro C.

\[ f_{110} = x_{111} \oplus x_{112} x_{113} \oplus x_{24} \oplus x_{126} \]
Estas 3 funciones en el estudio \cite{26} generan la cadena de salida. Adicionalmente, de estas funciones se puede observar que existen algunos términos que se pueden desplazar a otras funciones, de tal manera que el procesamiento realizado no se haga sobre una función específica sino sobre varias, así se evitan retardos en la operación.

El Trivium ya se encuentra en una configuración Galois, por lo tanto no debe pasar por el proceso inicial Fibonnaci-Galois descrito inicialmente, por ello se procederá a explicar el siguiente paso el cual es un caso más general para shift registers en una configuración Galois-Galois.

El primer paso será transformar el registro $A$.

$$f_{287} = x_0 \oplus x_1 x_2 \oplus x_{45} \oplus x_{219}$$

De este registro, el término $x_0$ no puede ser desplazado por la condición 1 de la transformación Galois-Galois, y con el objetivo de preservar los bits que influyen en la salida el término $x_{45}$ tampoco será desplazado. Así, solo queda el término $x_1 x_2$ y $x_{219}$. Para este caso, observe que $x_{219}$ solo se puede desplazar hasta $x_{195}$ o $x_{287}$ porque solo hasta allí, son funciones del tipo (4) condición 2, y no puede moverse más allá de $x_{287}$ porque estaría cruzando una de las fronteras $n - 1$ y 0 Condición 3. Con esto se da cumplimiento a las 3 condiciones contempladas en el algoritmo.

Hay múltiples formas de realizar la transformación del Trivium, sin embargo, adicional a estas 3 condiciones también se quiere preservar las relaciones establecidas previamente:

- El registro $A$, recibe 4 bits del registro $C$ y 1 bit del registro $A$.
- El registro $B$, recibe 4 bits del registro $A$ y 1 bit del registro $B$.
- El registro $C$, recibe 4 bits del registro $B$ y 1 bit del registro $C$. 
\[ f_{287} = x_0 \oplus x_{45} \]

\[ f_{287}^{x_1x_2} \rightarrow f_{286}, x_{(1-287+286)} \mod n = x_0, x_{(2-287+286)} \mod n = x_1 \]

\[ f_{287}^{x_{219}} \rightarrow f_{265}, \quad x_{(219-287+265)} \mod n = x_{197} \]

\[ f_{286} = x_{287} \oplus x_1x_2 \]

\[ f_{265} = x_{266} \oplus x_{197} \]

Así, se siguen recibiendo 4 bits del registro C y 1 bit del registro A, los bits provenientes de \( x_{287} \) y \( x_{266} \) son propios de la función del tipo (4) por eso no se cuenta, situación diferente se presenta con \( x_0 \), que pese a que es del tipo (4) este hace parte de la función inicial.

Este mismo proceso se repite con las demás funciones hasta obtener los siguientes resultados.

\[ f_{194} = x_{195} \oplus x_{222} \]

\[ f_{194}^{x_{196}x_{197}} \rightarrow f_{193}, x_{(196-194+193)} \mod n = x_{195}, x_{(197-194+193)} \mod n = x_{196} \]

\[ f_{194}^{x_{117}} \rightarrow f_{188}, \quad x_{(117-194+188)} \mod n = x_{111} \]

\[ f_{193} = x_{194} \oplus x_{195}x_{196} \]

\[ f_{188} = x_{189} \oplus x_{111} \]

Y

\[ f_{110} = x_{111} \oplus x_{126} \]

\[ f_{110}^{x_{112}x_{113}} \rightarrow f_{109}, x_{(112-110+109)} \mod n = x_{111}, x_{(113-110+109)} \mod n = x_{112} \]

\[ f_{110}^{x_{117}} \rightarrow f_{90}, \quad x_{(24-110+90)} \mod n = x_4 \]
\[ f_{109} = x_{110} \oplus x_{112}x_{113} \]
\[ f_{90} = x_{91} \oplus x_4 \]

Finalmente, la función que arroja la cadena de salida viene dada por:
\[ f_{out} = f_{287} \oplus f_{194} \oplus f_{110} \]

En conclusión, las nuevas funciones actualizadores del cifrador en flujo Trivium optimizado son:
\[ f_{287} = x_0 \oplus x_{45} \]
\[ f_{286} = x_{287} \oplus x_0 x_1 \]
\[ f_{265} = x_{266} \oplus x_{197} \]
\[ f_{194} = x_{195} \oplus x_{222} \]
\[ f_{193} = x_{194} \oplus x_{195} x_{196} \]
\[ f_{188} = x_{189} \oplus x_{111} \]
\[ f_{110} = x_{111} \oplus x_{126} \]
\[ f_{109} = x_{110} \oplus x_{111} x_{112} \]
\[ f_{90} = x_{91} \oplus x_4 \]
\[ f_{out} = x_{287} \oplus x_{194} \oplus x_{110} \]
CAPÍTULO 6 MARCO LEGAL

6.1 Ley 1273 de 2009 Delitos informáticos [46]

“Por medio de la cual se modifica el código Penal, se crea un nuevo bien jurídico tutelado – denominado “de la protección de la información y de los datos”- y se preservan integralmente los sistemas que utilicen las tecnologías de la información y las comunicaciones, entre otras disposiciones.”

Capítulo I. De los atentados contra la confidencialidad, la integridad y la disponibilidad de los datos y de los sistemas informáticos.

Artículo 269B: Obstaculización ilegítima de sistema informático o red de telecomunicación.
Artículo 269C: Interceptación de datos informáticos.

De estos artículos se pueden observar algunas de las posibles amenazas a las que se encuentran enfrentadas las organizaciones ya sean grandes o pequeñas como lo son la interceptación de datos o irrupción en los sistemas informáticos o redes de telecomunicaciones que si se materializan, pueden afectar la seguridad de la información y en casos extremos la continuidad del negocio. Según el ESET Security Report Latinoamérica 2015 [47], el incidente que más tuvo crecimiento fue el acceso indebido a la información, el cual pasó del 13% en el 2011 al 44% en el año 2014, siendo este un comportamiento generalizado en Latinoamérica.

Gracias a la eficiencia y velocidad del cifrador en cuestión, se pretende garantizar la confidencialidad, integridad y disponibilidad de la información en aplicaciones que requieren el cifrado de voz o cifrado del tráfico de datos en internet para evitar que si alguna organización se ve afectada por alguno de los actos contemplados en los artículos mencionados anteriormente, estas no sufran ningún tipo de repercusión o robo real de información.
6.2 Ley de inteligencia y contrainteligencia (Ley No. 1621 de 2013)

“Establece que los operadores de servicios de telecomunicaciones deberán ofrecer el servicio de llamadas de voz cifradas a personal del alto gobierno y de inteligencia (Ley 1621 de 2013. Parágrafo 2 del artículo 44)”.

Este artículo indica que solo un segmento muy pequeño de la población puede adquirir ese servicio de cifrado, dejando por fuera, por ejemplo, a periodistas y activistas por los DDHH. De lo anterior se pueden observar dos aspectos claves a tener en cuenta. Por un lado se encuentra la relevancia que tiene el cifrado de las comunicaciones para la transmisión y recepción de datos, aunque este excluyendo a un amplio segmento de la población, no obstante, gracias al desarrollo de esta tesis se podrá implementar el cifrador en flujo trívium con pocos recursos y por consiguiente todas aquellas personas y organizaciones que se encuentran marginadas en esta ley podrán acceder a los servicios de cifrado de alta velocidad.

6.3 Circular Externa 052 de 2007 de la SUPERINTENDENCIA FINANCIERA DE COLOMBIA [48]

CAPÍTULO DÉCIMO SEGUNDO: REQUERIMIENTOS MÍNIMOS DE SEGURIDAD Y CALIDAD PARA LA REALIZACIÓN DE OPERACIONES

“Este concepto tiene relación con los requerimientos mínimos de seguridad y calidad en el manejo de información a través de medios y canales de distribución de productos y servicios”.

Los numerales 3.2.5, 4.1.5, 4.2.2, 4.2.3 y 4.2.4 de esta circular definen los requerimientos necesarios que se encuentran relacionados con el cifrado para el manejo de la información, a continuación se mencionan algunos de ellos:
• 3.2.5 Implementar mecanismos de cifrado fuerte para el envío y recepción de información confidencial con los terceros contratados.

• 4.1.5 La información que viaja entre las oficinas y los sitios centrales de las entidades deberá estar cifrada usando hardware de propósito específico, o software, o una combinación de los anteriores. Para los establecimientos de Crédito el hardware o software empleados deberán ser totalmente separados e independientes de cualquier otro dispositivo o elemento de procesamiento de información, de seguridad informática, de transmisión y/o recepción de datos, de comunicaciones, de conmutación, de enrutamiento, de gateways, servidores de acceso remoto (RAS) y/o de concentradores.

En cualquiera de los casos anteriores se deberá emplear cifrado fuerte. Las entidades deberán evaluar con regularidad la efectividad y vigencia de los mecanismos de cifrado adoptados. Los incidentes relacionados con el fraude [49] externo o interno han crecido sosteniblemente pasando del 4% en el 2011 al 12% en el 2014, es por esto que es necesario adoptar controles que disminuyan la posibilidad de que por mala fe de un empleado la información de la compañía sea vulnerada

El cifrador trivium cumple como mecanismo de cifrado y que al mismo tiempo se desarrolla sobre hardware específico como lo es la plataforma NetFPGA.
La historia de los cifradores en flujo y del OTP comienza con Gilbert Vernam en 1917, aunque es su comienzo no se entendía directamente como un cifrador en flujo, sino como se indica en su artículo [50] de 1926 este era un “cipher printing telegraph system” que fue desarrollado durante la guerra mundial. El objetivo de esta investigación era obtener un sistema capaz de cifrar y descifrar mensajes en completo secreto, rápido, preciso y práctico para utilizar. Durante las pruebas, este sistema demostró que podía enviar exitosamente mensajes de forma secreta a una velocidad muy superior a los otros esquemas usados anteriormente. El cifrado consistía básicamente en combinar carácter por carácter del texto plano con una clave almacenada en una cinta perforadora y de esta manera se obtenía el texto cifrado. La operación de descifrado se realizaba de igual manera pero utilizando el texto cifrado en vez del texto plano. La función de cifrado y descifrado XOR, fue publicada en la patente 1310719 de Estados Unidos [51].

Al comienzo, los mensajes cifrados con este mecanismo eran descifrables ya que la clave utilizada no era aleatoria y reutilizable, este se volvió más complejo cuando Joseph Mauborgne [52][53] (capitán de la armada de estados unidos) encontró que si la clave era totalmente aleatoria realizar criptoanálisis sería más complejo.

One Time Pad

Posterior al desarrollo del primer esquema de cifrado propuesto por Vernam, se creó lo que se considera un sistema de cifrado con secreto perfecto, es decir, es irrompible, así se disponga de recursos computacionales infinitos, siempre y cuando se use correctamente. A este sistema se le conoce como One Time Pad (OTP) y solo es eficiente si la clave utilizada cumple las siguientes características: Debe ser totalmente aleatoria, de la misma longitud del mensaje a cifrar y solo se emplea una vez. Al principio se consideraba este cifrado muy difícil de romper, pero más adelante Claude Shannon demostró no solo que era difícil sino imposible de romper. La razón de ello, radica en que para descifrar un mensaje cifrado con OTP la opción
más factible es utilizar fuerza bruta y aunque se utilice, suponiendo recursos computacionales infinitos, se obtendrían muchos mensajes con significado, por lo tanto el atacante no sabrá cuál fue el mensaje original.

Sin embargo, en la actualidad, donde se habla en términos de gigabytes o terabytes, emplear este sistema no es muy eficiente pese a la gran seguridad que ofrece. Esto se debe a que se debe generar una llave aleatoria con un tamaño de Gigas o Teras, su transporte es complejo y costoso, por lo tanto no es práctico teniendo en cuenta que esa llave solo se puede emplear una sola vez.

Actualmente los cifradores en flujo se utilizan en sistemas que requieren cifrado en tiempo real, el más utilizado es el RC4, creado por Ron Rivest de RCA security, no obstante, se ha descubierto que este método de cifrado no es seguro ya que se puede obtener la clave a partir del mensaje cifrado mediante análisis estadístico [54]. Otro algoritmo que se utilizó ampliamente en el estándar GSM para el cifrado de conversación entre dos terminales fue el A5/1, sin embargo este también presentó debilidad en la periodicidad de la clave y por eso tampoco se considera seguro.

**Proyecto NESSIE.**

El proyecto NESSIE [55] surge entre los años 2000-2003 en Europa como una iniciativa para encontrar nuevos diseños de cifrado, similar a como lo hizo NIST con el AES. Durante este proceso se recibieron un total de 42 algoritmos en todas las categorías (Llave privada, llave pública, firma digital, funciones hash y algoritmos MAC, identificación) y de ese total en el año 2003 solo se seleccionaron 12. Estos algoritmos seleccionados no se les descubrieron ninguna debilidad. Sin embargo, en este proyecto se presentó una situación particular y fue que ninguno de los 12 candidatos correspondía a cifradores en flujo, ya que los 6 que se habían presentado en esta categoría fueron vulnerados mediante criptoanálisis. Producto de esta situación surgió una nueva iniciativa llamada eSTREAM Project.
Proyecto eSTREAM.

El proyecto eSTREAM48 nace entre el 2004-2008 con el único objetivo de encontrar diseños seguros de cifradores en flujo a través de diseños eficientes, compactos y adecuados para su adopción generalizada. La convocatoria de las primitivas se emitió por primera vez en noviembre de 2004 y terminó en abril de 2008.

El llamado para la construcción de estos cifradores se hizo para dos perfiles específicos, perfil tipo 1 y perfil tipo 2. El perfil tipo 1 se enfocó en cifradores en flujo para desarrollarse en software para aplicaciones que requerirían altos requerimientos de rendimiento. Los cifradores de perfil tipo 2 se debían implementar sobre hardware con recursos limitados (almacenamiento, compuertas o consumo de energía).

El proceso de selección de los finalistas se realizó a través de tres fases. En la fase 1 los cifradores debían demostrar ser simples y flexibles, junto con la justificación y análisis de soporte, claridad y una documentación completa. Los cifradores del perfil tipo 1 solo se aceptaban si estos demostraban tener un rendimiento mayor al ofrecido por el AES-128 en modo contador. La fase dos se concentró en algoritmos que el proyecto eSTREAM encontró fueran más eficientes y resistentes al criptoanálisis. La fase tres terminó en el 2008 con el anuncioamiento de los finalistas para ambos perfiles. En total se aceptaron 4 cifradores de perfil tipo 1 con una llave de 128 bits, y 3 cifradores de perfil tipo 2 con una llave de 80 bits. Uno de los finalistas del perfil tipo 2 fue el Trivium.

---

**TRIVIUM**

Es un cifrador de flujo síncrono orientado a hardware, su diseño fue pensado en crear un cifrador simple pero que no sacrificará la seguridad, velocidad y flexibilidad. Este algoritmo permite codificar la información bit a bit. Trivium [57] posee un diseño enfocado en la flexibilidad, esto quiere decir que es ideal en ambientes con restricciones en el número de compuertas lógicas, eficiente en cuanto al consumo de potencia en plataformas con recursos limitados de potencia y rápido en aplicaciones que requieren cifrado de alta velocidad. Además genera una secuencia de hasta $2^{64}$ bits a partir de un vector de inicialización y una clave de 80 bits.

Los cifradores en flujo o stream ciphers como Trivium aseguran el ocultamiento total de la información en casos particulares como las imágenes. Los cifradores en bloques, en el caso de las imágenes, pueden dejar rastros de la información original en la cifrada, mientras que los stream ciphers aseguran su total ocultamiento.

**NetFPGA**

El proyecto NetFPGA [49] es un proyecto que nació en el año 2007 en la universidad de Stanford, este consiste en un desarrollo de hardware y software de código abierto para la creación rápida de prototipos de dispositivos de red, este proyecto se centró en investigadores académicos, industria y estudiantes.

La plataforma NetFPGA tiene interfaces modulares que permiten el desarrollo de diseños complejos de hardware por la integración de bloques o módulos más simples.

NetFPGA es la plataforma experimental de-facto para implementaciones de investigación en redes. A través de una amplia gama de temas de redes, desde arquitectura hasta algoritmos y desde diseño eficiente de energía hasta enrutamiento y reenvío.

Entre algunas de las temáticas que se pueden trabajar con las NetFPGA incluyen:

---

Diseño de protocolos y servicios

Ingeniería de tráfico

Análisis y optimización de sistemas

Diseño e implementación de sistemas de comunicación

Seguridad informática

Redes de nueva generación (NGN), redes inalámbricas, redes ópticas.

Optimización, nuevas tecnologías y aplicaciones de la Banda Ancha.
Gestión del servicio.

Respecto a la NetFPGA, se está trabajando ahora en la implementación de proyectos en la tarjeta 10G, el sitio oficial netfpga.org contiene 8 proyectos contribuidos por la comunidad, para implementar sobre esta referencia.
CAPÍTULO 8 IMPLEMENTACIÓN DEL CIFRADOR EN FLUJO TRIVIUM BAJO LA PLATAFORMA NETFPGA 1G

Objetivo: se procederá a realizar el proceso de implementación del cifrador en flujo Trivium, de acuerdo a las especificaciones de los creadores del cifrador, bajo la plataforma NetFPGA 1G y se realizarán pruebas de cifrado, y pruebas en software y hardware usando las FPGAs Spartan 3e – 3s500e y 5vlx110t.

Este objetivo se logrará de la siguiente manera:

- **Instalación:** La implementación se va a realizar en el laboratorio del grupo Nyquist en Parquesoft (CDV).
- **Lenguaje:** El algoritmo se implementará en código VHDL o Verilog.
- **Recursos:** Se cuenta con 4 tarjetas NetFPGA 1G.
- **Sistema operativo:** Se va a instalar Linux Fedora versión 13 en cada equipo, esta versión se eligió por su compatibilidad con el hardware.
- **Se va a diseñar el conjunto de pruebas de operación del algoritmo.**
- **Como anexo al documento se encuentra el archivo “GuiadelInstalacion.doc”, como guía de instalación y configuración para poner en marcha la plataforma NetFPGA 1G.**
8.1 Implementación del algoritmo en software

Para las pruebas que se realizaron a nivel de software, se utilizó el código de referencia en lenguaje C del cifrador en flujo Trivium\textsuperscript{50}, el cual está basado en el algoritmo de referencia del cifrador. \textsuperscript{51}

La ECRYPT, “European Network of Excellence for Cryptology” \textsuperscript{[34] \textsuperscript{52}}, o la red Europea de excelencia en Criptografía viene trabajando desde el año 2007 y sus objetivos son:

- Continuar con la intensificación de la colaboración de los investigadores Europeos y del mundo en el tema de seguridad de la información;
- Mantener y reforzar la excelencia de la investigación y la industria europea en el campo de la criptografía y obtener una integración duradera;
- Fortalecer e integrar la investigación en criptología en Europa y disminuir la fragmentación mediante la creación de una infraestructura de investigación y la organización de la investigación en laboratorios virtuales estableciendo así una agenda de investigación conjunta;
- Mejorar el estado de la técnica en la práctica y la teoría de la criptografía;
- Mejorar la comprensión de los algoritmos y protocolos existentes;
- Ampliar los fundamentos teóricos de la criptografía;
- Desarrollar una infraestructura conjunta que incluye: herramientas para la evaluación de algoritmos criptográficos y un entorno de referencia para el hardware y software criptográficos.

El Proyecto de cifradores en flujo eSTREAM \textsuperscript{[28] \textsuperscript{53}} fue un esfuerzo de varios años, desde 2004 a 2008, que se creó con el fin de promover el diseño eficiente y

\textsuperscript{50} http://www.ecrypt.eu.org/stream/svn/viewcvs.cgi/ecrypt/trunk/submissions/trivium/
\textsuperscript{51} http://www.ecrypt.eu.org/stream/ciphers/trivium/trivium.pdf
compacto de cifradores en flujo adecuados para uso general. Como resultado del proyecto se generó un portafolio de nuevos cifradores en flujo.

El portafolio de los sistemas de cifrado en flujo eStream se divide en dos perfiles. El Perfil 1 contiene cifradores adecuados para aplicaciones de software que requieren un alto rendimiento. Los sistemas de cifrado en flujo del perfil 2 son adecuados para aplicaciones de hardware con recursos restringidos o limitados, como el almacenamiento, número de puertas, o el consumo de energía.

**eSTREAM Testing Framework**

El marco de pruebas [29] 54 hace referencia a una colección de secuencias de comandos y código C que ponen a prueba tres aspectos del código presentado: el cumplimiento de la API, la correctitud, y el rendimiento.

Uno de los requisitos impuestos a todos los cifradores en flujo es que deben demostrar el potencial de ser superior al AES en al menos un aspecto significativo. El aspecto importante para los candidatos Perfil I es el rendimiento del software, mientras que para los del perfil II es el rendimiento con recursos limitados de hardware.

El rendimiento del software se puede medir de muchas maneras diferentes, como el cifrado de grandes cadenas y la tasa de cifrado de paquetes, y con el fin de hacer comparaciones lo más justo posible, eStream decide desarrollar un marco de pruebas. Este marco tiene dos objetivos:

- Asegurar que todas las propuestas de cifradores en flujo son sometidos a las mismas pruebas en las mismas circunstancias.

---

La automatización del procedimiento de prueba, tanto como sea posible de tal manera que las nuevas implementaciones optimizadas se puedan incluir y se ensayan con el menor esfuerzo posible.

Para realizar la instalación del framework se puede descargar el archivo comprimido de la última versión de la plataforma de pruebas desde el repositorio SVN de ECRYPT [30]. Este repositorio contiene las implementaciones más recientes de todos los cifradores en flujo candidatos, junto con los vectores y scripts de prueba. Se realizaron dos pruebas en software en equipos con procesadores diferentes, de un lado el Pentium R de 1981 MHz y por el otro un AMD FX 6100 de 3314.9 MHz, y se utilizaron una llave y un vector de inicialización de 80 bits y paquetes de diferentes tamaños. Las longitudes de los paquetes (40, 576, y 1500 bytes) fueron elegidos por ser los más representativos en el tráfico visto en Internet.

Tabla 8.1 Resultados de la implementación en Software CPU Pentium R de 1981 MHz.

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Trivium</th>
</tr>
</thead>
<tbody>
<tr>
<td>Profile</td>
<td>__H3</td>
</tr>
<tr>
<td>Key</td>
<td>80</td>
</tr>
<tr>
<td>IV</td>
<td>80</td>
</tr>
<tr>
<td>CPU speed</td>
<td>1981.0 MHz</td>
</tr>
</tbody>
</table>

**Testing Stream Encryption speed:** Encrypted 23 blocks of 4096 bytes

- Total time: 424694 clock ticks (214,38 usec)
- Encryption speed: 4,51 (cycles/byte)
- Encryption speed: 3515,48 Mbps

**Testing Packet encryption speed:**

- 400 packets of 40 bytes: 271109 clock ticks (136,85 usec)
- 80 packets of 576 bytes: 373376 clock ticks (188,48 usec)
- 49 packets of 1500 bytes: 389826 clock ticks (196,78 usec)

---

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Trivium</th>
</tr>
</thead>
<tbody>
<tr>
<td>Profile</td>
<td>H3</td>
</tr>
<tr>
<td>Key</td>
<td>80</td>
</tr>
<tr>
<td>IV</td>
<td>80</td>
</tr>
<tr>
<td>CPU speed</td>
<td>3314.9 MHz</td>
</tr>
<tr>
<td>Testing Stream Encryption speed: Encrypted 15 blocks of 4096 bytes</td>
<td></td>
</tr>
<tr>
<td>Total time</td>
<td>729948 clock ticks (220,20 usec)</td>
</tr>
<tr>
<td>Encryption speed</td>
<td>11,88(cycles/byte)</td>
</tr>
<tr>
<td>Encryption speed</td>
<td>2232,12 Mbps</td>
</tr>
<tr>
<td>Testing Packet encryption speed:</td>
<td></td>
</tr>
<tr>
<td>390 packets of 40 bytes</td>
<td>60 packets of 576 bytes</td>
</tr>
<tr>
<td>Total time</td>
<td>712990 clock ticks (215,08 usec)</td>
</tr>
<tr>
<td>Encryption speed</td>
<td>1828,18 (cycles/packet)</td>
</tr>
<tr>
<td>Encryption speed</td>
<td>45,70 (cycles/byte)</td>
</tr>
<tr>
<td>Encryption speed</td>
<td>580,23 Mbps</td>
</tr>
<tr>
<td>overhead</td>
<td>284,7%</td>
</tr>
<tr>
<td>Testing key agility (90 blocks of 256 bytes):</td>
<td></td>
</tr>
<tr>
<td>Total time</td>
<td>308609 clock ticks (93,10 usec)</td>
</tr>
<tr>
<td>Encryption speed</td>
<td>13,39 (cycles/byte)</td>
</tr>
</tbody>
</table>

Tabla 8.2 Resultados de la implementación en Software CPU AMD FX 6100 de 3314.9 MHz.
8.2 Implementación del algoritmo en hardware

La implementación del algoritmo Trivium se presenta en el lenguaje VHDL, está conformado por la entidad TRIVIUM_CORE (Fig. 8.1), la cual contiene la lógica central del algoritmo Trivium oficial, tiene como entradas la señal de reloj, una señal de control que se usa para controlar las operaciones de carga de la llave y del vector de inicialización, y la operación general. La llave secreta que es un vector de 80 bits, el Vector de Inicialización de 80 bits, y como salida la cadena de salida (keystream) para realizar el proceso de cifrado con los datos de entrada, también conocida como secuencia cifrante.
Descripción General del diseño

También está conformado por la entidad **Trivium_Generator** Figura 8.2, que contiene como entradas el IV_INPUT de 80 bits, el PLAINTEXT_IN de 8 bits que es la cadena a cifrar, el PLNTXT_EN de 1 bits que es un habilitador que se usa para validar la operación XOR entre la keystream y el plaintext, el RST de 1 bit usado en la máquina de estados para cambiar de estado y el SYS_CLK de 1 bit que es la señal de reloj. Y como salidas el CPHRTXT_RDY de 1 bit es una señal que confirma que el plaintext se encuentra cifrado y el CIPHERTEXT_OUT de 8 bits que el texto cifrado propiamente dicho. Esta entidad a su vez internamente crea 8 instancias del componente TRIVIUM_CORE, y usa los vectores de prueba oficiales de ECRYPT stream cipher project [29] para el KEY, IV, y las cadenas de entrada para realizar el proceso en cifrado.

---

Figura 8.2 Esquema del módulo Trivium_Generator. Fuente: Michael Calvin McCoy.

Tabla 8.3 Señales del Trivium_Generator.

<table>
<thead>
<tr>
<th>Nombre de Señal</th>
<th>Dirección</th>
<th>Descripción</th>
</tr>
</thead>
<tbody>
<tr>
<td>IV_INPUT</td>
<td>Input</td>
<td>Vector de inicialización de 80 bits.</td>
</tr>
<tr>
<td>PLAINTEXT_IN</td>
<td>Input</td>
<td>Cadena de entrada a cifrar de 8 bits.</td>
</tr>
<tr>
<td>PLNTXT_EN</td>
<td>Input</td>
<td>Señal habilitadora de 1 bit.</td>
</tr>
<tr>
<td>RST</td>
<td>Input</td>
<td>Señal de reset.</td>
</tr>
<tr>
<td>SYS_CLK</td>
<td>Input</td>
<td>Señal de reloj.</td>
</tr>
<tr>
<td>CIPHERTEXT_OUT</td>
<td>Output</td>
<td>Texto cifrado de salida de 8 bits.</td>
</tr>
<tr>
<td>CPHRTXT_RDY</td>
<td>Output</td>
<td>Señal de aviso de fin de proceso de cifrado de 1 bit.</td>
</tr>
</tbody>
</table>

Entre las herramientas utilizadas para el proceso de implementación en hardware se emplearon el Xilinx ISE Design Suite 14.7 para la descripción en HDL, Isim 14.7 para el proceso de simulación y los testbench, y las FPGA Spartan 3E XC3S500e
[33] y XUPV5-LX110T [32] de Xilinx para las pruebas y determinar el consumo de recursos de las tarjetas. En los resultados que se muestran en la Tabla No. 12, además se encontró que la Spartan 3E se consume 1649 LUTs mientras que la Virtex 5 se consume 2332 LUTS. La figura 8.3 muestra la estructura interna del componente Trivium_Generator en la cual se puede observar la instanciación del componente TRIVIUM_CORE.

Tabla 8.4 Resultados de la implementación sobre FPGAs.

<table>
<thead>
<tr>
<th>Cifrador</th>
<th>Bits de Seguridad</th>
<th>Dispositivo</th>
<th>Slice Flip Flop</th>
<th>Consumo de recursos</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trivium</td>
<td>80</td>
<td>Spartan 3e – 3s500e</td>
<td>2340 of 9312</td>
<td>35%</td>
</tr>
<tr>
<td>Trivium</td>
<td>80</td>
<td>5vlx110t</td>
<td>40 of 2372</td>
<td>3%</td>
</tr>
</tbody>
</table>

Tabla 8.5 Resultados de la implementación sobre FPGAs (2).

<table>
<thead>
<tr>
<th>Cifrador</th>
<th>Dispositivo</th>
<th>Frecuencia Máxima (MHz)</th>
<th>Area (slices)</th>
<th>Throughput (Mbps)</th>
<th>Thr/Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trivium</td>
<td>Spartan 3e – 3s500e</td>
<td>216,887</td>
<td>1649 of 4656</td>
<td>500</td>
<td>0,3032</td>
</tr>
<tr>
<td>Trivium</td>
<td>5vlx110t</td>
<td>484,914</td>
<td>2332 of 69120</td>
<td>800</td>
<td>0,3430</td>
</tr>
</tbody>
</table>

La fig. 8.4 es un esquemático donde se muestra de forma clara las diferentes instancias del Trivium Generator.
Simulación en ISim Para el proceso de simulación se utilizó la herramienta ISim 14.7, se realizaron pruebas con los testbench del Trivium_Core y del Trivium_Generator. El periodo para el Trivium_Generator_TB fue de 10000 ps o 10 ns, mientras que para el Trivium_Core es de 20 ns.

La fig. 8.5 muestra la simulación en modelsim del Trivium_Core, en el cual a los 100 ns, la señal de control cambia de 00 a 10, lo cual inicializa el key y el IV.
La figura 8.6 muestra que la señal de control pasa al valor 11 a los 120 ns.

Figura 8.6 Simulación del Trivium-Core II. Fuente: Los Autores.

La figura 8.7 hace parte de la simulación del módulo Trivium_Generator, en el cual se muestra el IV, el plaintext_in, y el ciphertext_out se encuentra en ceros.

Figura 8.7 Simulación del Trivium_Generator I. Fuente: Los Autores.
En la siguiente figura se muestra la señal cphrtxt_rdy en 1 y la salida en la señal ciphertext_out en bloques de 8 bits.

![Figura 8.8 Simulación del Trivium_Generator II. Fuente: Los Autores](image)

En los enlaces referenciados se puede ver las simulaciones realizadas en el ISE de Xilinx y en modelsim sobre el algoritmo original. 59

El componente Trivium_Generator a los $1,1575 \times 10^{-5}$ segundos genera la primera salida cifrada de 8 bits y cada 10 ns, es decir, cada $1 \times 10^{-8}$ segundos genera la siguiente salida de 8 bits. Además para generar 1Mbit se requiere cifrar 125.000 cadenas de 8 bits, lo cual se tardaría 0,00125 segundos.

### 8.3 Implementación del algoritmo sobre la plataforma NetFPGA 1G.

#### Descripción de la plataforma NetFPGA

El proyecto NetFPGA ha sido un esfuerzo por desarrollar un hardware y software abierto para un prototipado rápido de dispositivos de red y seguridad. Está dirigido a investigadores académicos, usuarios de la industria y estudiantes. El proyecto NetFPGA inició en el año 2007 como un proyecto de investigación de la universidad de Stanford, llamado NetFPGA-1G. A finales del 2010 se vendieron más de 1800 tarjetas NetFPGA, el proyecto fue acogido por cerca de 150 instituciones educativas.

59 Agregar enlaces de youtube con las simulaciones.
de 15 países del mundo. Para el 2014 cerca de 226 artículos científicos habían sido publicados respecto al uso de la plataforma.\(^6\)

Elementos de la plataforma NetFPGA:

- Tarjeta NetFPGA
- Herramientas y diseños de referencia: librerías.
- Proyectos contribuidos: código fuente, 1G 53 proyectos, 10G 16 proyectos.
- Comunidad: foros, lista de correos, 150 universidades, 15 países

Entre algunas de las temáticas que se pueden trabajar con las NetFPGA incluyen:

- Diseño de protocolos y servicios.
- Ingeniería de tráfico.
- Análisis y optimización de sistemas.
- Diseño e implementación de sistemas de comunicación.
- Seguridad informática.
- Redes de nueva generación (NGN), redes inalámbricas, redes ópticas.
- Optimización, nuevas tecnologías y aplicaciones de la Banda Ancha.
- Gestión del servicio.

Los recursos necesarios para la implementación de la plataforma son:

- Tarjeta NetFPGA 1G (Se decide realizar la implementación sobre esta tarjeta ya que cuenta con recursos limitados en relación con la NetFPGA 10G, CML y SUME)
- Equipo con puerto PCI, y espacio necesario para instalar la tarjeta.
- Manilla antiestática para manipular la NetFPGA.
- Archivo .ISO del Sistema operativo Fedora 13, se puede grabar en DVD, en caso de que el equipo no posea DVD se puede grabar en una USB booteable.
- NetFPGA Package Full 3.0.1, y 2.2.0 para test de regresión.

\[^6\) http://netfpga.org/site/#/about/\]
- 40 Gb de espacio en disco libres.
- Cables SATA, cables UTP.
- Instaladores de las herramientas CAD: ISE Xilinx 10.1, Modelsim, Módulos de memoria para simulación: Micron DDR2 SDRAM y Cypress SRAM.
- Conocimientos básicos en comandos Linux y en VHDL o Verilog para realizar las implementaciones de código.

Entre algunas de las formas que se pueden trabajar con las NetFPGA incluyen:

1. Usar la tarjeta para correr un servicio (Demonio en Linux), y convertir la máquina Linux en enrutador.
2. Reusar los diseños de referencia, para agregar una funcionalidad, módulo o bloque.
3. Crear un desarrollo nuevo o implementar un diseño completamente nuevo.

Para el caso en particular del presente proyecto se trabajará tomando la segunda opción de las formas de trabajar con la plataforma, se tomó como base el diseño del proyecto cryptoNIC, adicionando un módulo o bloque personalizado. Del cual se presentará con base a su vista funcional, física, de desarrollo y lógica.  

Para la segunda opción es necesario descargar y utilizar el hardware original provisto por la NFP, luego modificar el hardware agregando módulos extraídos de otras NFPs o agregando nuevos módulos propios. Compilar el código fuente en Verilog con las herramientas estándares, descargar el bitfile a la FPGA, el nuevo hardware puede complementarse agregando nuevo software al Host o modificando el existente.

Ejemplos:

---

- En el Router IPv4, implementar Trie Longest Prefix Match (LPM) Lookup en lugar del existente CAM LPM Lookup, para la tabla de enrutamiento del hardware.
- Modificar el Router IPv4 para implementar NAT (Network Address Translation) o un Firewall.

**Vista Funcional**
La funcionalidad se puede implementar en uno o más módulos dentro del user data path.

El pipeline está dividido en varias etapas, a continuación se habla de cada una de ellas. La Figura 8.9. muestra el pipeline de referencia de la NetFPGA.

La etapa inicial está conformada por ocho colas conocido como colas RxQ, 4 colas MAC (para puerto físico) y 4 colas CPU (para puerto lógico). Estas reciben los paquetes desde los puertos de entrada y salida como Ethernet y el PCI sobre DMA. Los paquetes que llegan a la cola CPU RxQ son lo enviados desde el software por la interfaz con el nombre nf2cX.
Figura 8.9 Diseño sobre el pipeline de referencia de la NetFPGA. Fuente: NetFPGA.

La siguiente etapa está conformada por el Input Arbiter, que no es más que un regulador de datos que decide cual cola Rx es la siguiente a atender, y ubica el paquete en manos del próximo modulo en el pipeline.

Luego el módulo de búsqueda de puerto de salida (Output Port Lookup), el cual se encarga de determinar con base en la información de las cabeceras del paquete
hacia qué puerto dirigirlo, una vez sea determinado, el paquete se entrega al módulos de colas de salida.

El módulo de colas de salida (Output Queues) almacena el paquete hasta que la Cola Tx esté disponible a aceptarlo para su transmisión, estas están asociadas a las colas TxQ.

Las colas TxQ son las encargadas de mover los paquetes desde las colas de salida hacia el puerto específico. Cada módulo tiene asociado un conjunto de registros para acceder a la información del estado y además un conjunto de señales de control.

Los módulos Input Arbiter, Output Port Lookup y Output Queues conforman el User Data Path (UDP).

**Vista Física**

La arquitectura del diseño de la NetFPGA el bus en el User Data Path tiene un ancho de banda de 64 bits a una frecuencia de 125 MHz, o sea un ancho de banda de 8 Gbps. Los paquetes son ubicados entre módulos consecutivos e interconectados, como se puede ver en la figura 8.10., usando estructuras FIFO con una interfaz simple de cuatro señales: data, ctrl, wr y rdy, las señales wr y rdy tienen un ancho de 1 bit mientras que la señal ctrl (de control) tiene un ancho de 8 bits y data (de datos) tiene un ancho de 64 bits.

Las señales de wr y rdy son señales de acuerdo entre los módulos, si el módulo i quiere enviar un paquete al módulo i+1, primero el módulo i+1 debe anunciar que está listo para recibir datos, esto lo hace estableciendo la señal rdy, cuando el módulo i detecta esta señal pone el paquete de datos en el bus de data, y establece las señales de ctrl acorde con los datos transmitidos, y eleva la señal wr cuando está listo para enviar. Si el módulo i+1 no puede aceptar ningún dato, este debe interrumpir la señal rdy por lo menos un ciclo de reloj antes de la llegada de los datos.
El bus CTRL es usado para dos propósitos. Por un lado distinguir una cabecera de módulo de otra, y por el otro determinar el final de un paquete. El bus CTRL es no-cero para una cabecera de modulo distinguiéndola de otras, en caso de múltiples cabeceras. Cuando el paquete recibido empieza después de las cabeceras de módulo, el bus CTRL es establecido a 0, en la última palabra del paquete, el CTRL indicara cual es el último byte en la última palabra. Esto se logra estableciendo a 1 en la posición del último byte.
Vista de Desarrollo

A continuación en las figuras 8.11, 8.12, 8.13 y 8.14 se muestra en detalle la estructura de directorios del package NetFPGA.

Figura 8.11 Estructura de directorios de la NetFPGA. Fuente: NetFPGA.
Figura 8.12 Estructura de directorios de la NetFPGA. Fuente: NetFPGA.

Figura 8.13 Estructura de directorios de la carpeta lib. Fuente: NetFPGA.
La siguiente figura muestra el pipeline de referencia con el módulo Trivium dentro del User Data Path.
Figura 8.15 Diseño sobre el pipeline de referencia de la NetFPGA con el módulo Trivium agregado. Fuente: Modificada por el autor.

La guía de instalación contiene la información completa y detallada para completar el proceso instalación y configuración, la cual se desarrolló con base en [25], esta ha sido desarrollada durante el proceso de investigación. El sistema operativo
Fedora 13 se recomienda ya que contiene los módulos del Kernel necesarios para que el OS reconozca la tarjeta NetFPGA. El proceso de instalación del sistema operativo es similar al de cualquier distribución de Linux. Una vez terminado el proceso de instalación del sistema, se procede a verificar las interfaces de red. Luego se realiza la instalación de las herramientas CAD como el ISE Xilinx 10.1, esta se usa para la síntesis y análisis de los diseños HDL, además permite realizar análisis de tiempos y esquemas, a continuación se debe instalar el ModelSim el cual es la herramienta que se usa para simulación de Lenguajes de Descripción de Hardware como VHDL y Verilog. Una vez terminados los pasos anteriores se procede a descargar la última versión del NetFPGA Package y a instalar los módulos de memoria necesarios para la simulación.

Para el proceso de verificación de la instalación, primero se verifican las interfaces, luego se ejecuta el Selftest y las pruebas de regresión que aseguran que todos los componentes de la plataforma están funcionando correctamente.

Luego de haber terminado el proceso de verificación se puede continuar con la implementación de los diseños de referencia como NIC, Router Kit, Control de Buffers que ayudan al entendimiento y uso de la plataforma NetFPGA.

Las fases del proceso de desarrollo en la plataforma NetFPGA se muestran a continuación en la (Fig.8.16).
Arquitectura NetFPGA

Los módulos de la NetFPGA están conectados como una secuencia de estados dentro del pipeline que consiste en ir transformando un flujo de datos en un proceso comprendido por varias fases secuenciales. Los estados se comunican usando una interfaz FIFO (First Input – First Output) [24]. El Packet-Bus se encarga de transferir los paquetes de una etapa a la siguiente usando la misma interfaz (Fig. 8.17), mientras que el Register Bus crea un ciclo con los registros de cada módulo formando una especie de anillo.

Sistema de Registros.

El sistema de registros asigna direcciones a todos los registros en un proyecto. Cada proyecto tiene un archivo de configuración que especifica los módulos que usa, y cada módulo tiene un archivo de configuración que especifica los registros
que provee. El sistema de registros lee el archivo de configuración del proyecto para identificar los módulos y el número de instancias de cada módulo; lee el archivo de configuración de cada módulo individualmente; localiza la memoria en el espacio de direcciones de registro para cada instancia de módulo y finalmente enlaza la localización del registro a los archivos Verilog, C y Perl. En la fig. 8.17 se muestra un esquema de un sistema.

![Diagrama de sistema de registros de NetFPGA](image)

**Figura 8.17** Sistema de Registros de la NetFPGA, fuente: NetFPGA.org.

Los anteriores pasos quedan mejor descritos y organizados a continuación.

El procesamiento inicia cuando el usuario invoca la generación de registro utilizando nfregistergen.pl o cuando el usuario simula o sintetiza un proyecto.

1. Leer los archivos de configuración global de lib/verilog/core/common/xml.
2. Leer el archivo de configuración project.xml de projects/\(<project_dir>/include.
3. Leer los archivos de configuración para todos los módulos especificados en la sección \(nf: use\_modules\) de project.xml.
4. Leer los archivos de configuración para cualquier modulo localmente definido.
5. Identificar los módulos usados por el proyecto como se especifica en la sección \(nf: mem\_alloc\) de project.xml.
6. Identificar y leer cualquier archivo de configuración compartido por los módulos.

7. Identificar todas las constantes definidas en el proyecto y los módulos.

8. Identificar todos los tipos definidos en los proyectos y módulos.

9. Expandir todas las constantes.
10. Verificar que el tamaño especificado de módulo es suficiente para contener el número de registros.
11. Localizar memoria a cada módulo, y para cada registro en cada módulo.

La Figura 8.18. muestra las señales del bus de registros de la NetFPGA.

Figura 8.18 Señales del bus de registros. Fuente: NetFPGA.
Paquetes

La NetFPGA trabaja paquetes a nivel de trama. Cada trama inicia con un preámbulo de 8 bytes, los cuales son eliminados del paquete mientras es empujado a las Colas Rx, la trama contiene dos direcciones, una para el destino y otra para el origen cada una utiliza seis bytes de longitud. A continuación está el campo tipo que indica al receptor qué hacer con la trama. Es decir, a que protocolo dirigir la trama. Luego vienen los datos de IP los datos de las cabeceras de red, transporte y el payload.

Cuando los paquetes llegan a las colas RxQ son fragmentados en palabras de 64 bits, esto complica un poco su manipulación. Para obtener algún campo de las cabeceras del paquete, es necesario, identificar en que número de palabra o palabras (en caso en que esté distribuido en dos o más palabras) se encuentra dicho campo, además en qué posición de ésta palabra, un ejemplo puede ser: si se requiere recuperar el campo del Time To Live (TTL) de los encabezados de la cabecera del IP.

La figura 8.19 muestra la ruta de un paquete a través de las diferentes capas:

- Paquete llega al CPU TxQ.
- Una interrupción notifica al controlador la espera del paquete.
- DMA transfer se inicializa.
- Una interrupción notifica que la transferencia es completa.
- El controlador ubica el paquete en la pila de red.
Proyecto cryptoNIC

Es un proyecto de referencia que implementa una tarjeta de interfaz de red (NIC) que cifra en la transmisión y descifra en la recepción. Este módulo utiliza la operación lógica XOR, que se representa con un ^, para realizar cifrado y descifrado.

El método consiste en realizar una XOR entre el mensaje y la llave para realizar el cifrado, y una XOR entre el texto cifrado y la llave para realizar el proceso de descifrado. El proceso de cifrado y descifrado se explican en detalle en el capítulo uno sobre cifradores en flujo. La Figura 8.20. muestra la definición de los parámetros, inputs y outputs del módulo crypto.
El proyecto cryptoNIC, usa una llave estática de 32 bits y la idea es encriptar cada palabra que se encuentre en el payload con llave. Ver figura 8.21.

La siguiente figura muestra el diseño del módulo crypto sobre el pipeline de referencia de la plataforma.
Figura 8.22 Diseño sobre el pipeline de referencia de la NetFPGA con el módulo Trivium agregado. Fuente: N.etFPGA.

Cada diseño está representado por un proyecto, el cual se encuentra ubicado en la ruta:

/netfpga/projects/<nombre_del_proyecto>

- /netfpga/projects/cryptonic

Archivo XML

El archivo project.xml da información sobre el proyecto y especifica la lista de módulos compartidos usados por el proyecto. Un ejemplo de archivo XML es como que se observa en la figura 8.23:
Figura 8.23 Archivo xml del proyecto crypto_nic. Fuente: NetFPGA.

Las figuras 8.24 y 8.25 muestran la información detallada del archivo xml del proyecto cryptonic.

### Module XML file (1)

- Each module (with registers) has an XML file

```xml
<?xml version="1.0" encoding="UTF-8"?>
<nf module="crypto">
  <nf name="/crypto">
    <nf description="/ Registers for Crypto Module \</nf description>

    <nf prefix="crypto">
      <nf description="/ Registers for Crypto Module \</nf description>

    <nf location="/udp">
      <nf description="/ Registers for Crypto Module \</nf description>

    <nf blocksize="/64">
      <nf description="/ Registers for Crypto Module \</nf description>

  </nf>

</nf>
```

Name/description

Prefix appears before register names in source code

Location: where in the design should this module be instantiated?

udp = user data path

Amount of memory to allocate to the block (in bytes, use k/m to indicate kilo/megabytes)

Figura 8.24 Archivo xml del proyecto I. Fuente: NetFPGA.
Para implementar el proyecto crypto_nic se deben agregar las siguientes líneas al archivo `bashrc`:

- Export NF_DESIGN_DIR=$NF_ROOT/projects/crypto_nic
- Export PERLIB5=$NF_ROOT/lib/perl5:$NF_DESIGN_DIR/lib/perl5

Luego se deben copiar los siguientes archivos de la ruta `/netfpga/lib/verilog/core` a `/netfpga/projects/crypto_nic/src`

- User_data_path/reference_user_data_path/src/user_data_path.v
- Module_template/src/module_template.v

Luego se crea crypto.v desde module_template.v y se adiciona al user data path.

Luego se realizan las declaraciones para el output port lookup, luego se crean las instanciaciones para insertar el nuevo módulo en el pipeline.
En el proyecto de cryptonic se asume que todos los paquetes son IPv4 por simplicidad. La siguiente Figura presenta la estructura interna del módulo crypto_nic.

![Diagrama de la estructura interna del módulo crypto_nic](image)

Figura 8.26 Estructura interna del módulo crypto_nic. Fuente: NetFPGA.

La siguiente figura muestra la estructura interna de un paquete y las partes que se van a encriptar.

![Diagrama de la estructura interna de un paquete](image)

Figura 8.27 Estructura interna de un paquete. Fuente: NetFPGA.
La idea es implementar una máquina de estados dentro del archivo crypto.v, usando una llave estática inicialmente. Para crear la llave estática se declara en el módulo como parámetro local, es necesario implementar la máquina de estados sin modificar el paquete, luego se actualiza la máquina de estados para modificar el paquete con el XOR entre la llave y el payload. Luego se definen los parámetros y los registros en el archivo .v, luego se adiciona la lógica combinacional para leer los datos en FIFO.

Para la simulación en el ISE/ModelSim, se crea el proyecto, se adiciona el archivo crypto.v al proyecto, se adicionan los otros módulos necesarios que se encuentra en /core/utils/ y se adiciona también los archivos de definición como:

Registers.v, NF_2.1_defines.v y udp_defines.v

Luego se agregan los includes a generic_regs.v, crypto_nic.v y a NF_2.1_defines.v

Para probar el proyecto CryptoNIC es necesario:

- Ejecutar make en el directorio netfpga/projects/crypto_nic/synth
- Descargar el bitfile nf2_top.bit sobre la NetFPGA con el comando nf_download nf2_top.bit
- Se debe conectar las interfaces nf2c0 de las dos NetFPGAs con un cable
- Se debe configurar las direcciones IP de ambas NIC.
- Se hace ping entre las interfaces
Verificación de los hardware tests

- netfpga/projects/crypto_nic/test/both_crypto_encrypt/run.py
- netfpga/projects/crypto_nic/test/both_crypto_decrypt/run.py

La Figura 8.28 muestra cómo se debe realizar la implementación del proyecto crypto_nic, para esto es necesario contar con dos equipos con la plataforma NetFPGA configurada, uno de los equipos se utiliza para encriptar y el otro para desencriptar.

Figura 8.28 Ejemplo de implementación del proyecto crypto_nic. Fuente: NetFPGA.

Las imágenes previas, son imágenes de referencia que aparecen en la mayoría de artículos y presentaciones de cursos como en [28] sobre la arquitectura y el pipeline de referencia de la NetFPGA 1G.
A continuación se explican cada una de las variables que se van a medir en las pruebas.

**Throughput [Mbps]:** Se refiere a la tasa que se mide cuando se produce una nueva salida con respecto al tiempo, o también la información procesada, comúnmente se mide en bits por segundo.

**Area [CLB slices] o (um2):** Esta medida se determina de acuerdo al número de compuertas lógicas y Lookup Tables (LUT) usadas en el proceso de síntesis del algoritmo, o Slices CLB para Xilinx.

**Throughput/Area Ratio (Mbps/slice):** Normalmente usada para medir la eficiencia del diseño.

**Frecuencia Máxima [MHz]:** También conocida como Maximum Clock Frequency. Hace referencia a la frecuencia máxima alcanzada durante la ejecución o síntesis del algoritmo

**Tiempo de cifrado:** Se determina el tiempo en segundos que tarda el proceso para cifrar y descifrar una determinada entrada.

Las siguientes figuras muestran el consumo de recursos en las diferentes tarjetas.

<table>
<thead>
<tr>
<th>Logic Utilization</th>
<th>Used</th>
<th>Available</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Slice Registers</td>
<td>2332</td>
<td>69120</td>
<td>3%</td>
</tr>
<tr>
<td>Number of Slice LUTs</td>
<td>2369</td>
<td>69120</td>
<td>3%</td>
</tr>
<tr>
<td>Number of fully used LUT-FF pairs</td>
<td>2329</td>
<td>2372</td>
<td>98%</td>
</tr>
<tr>
<td>Number of bonded I/Os</td>
<td>100</td>
<td>640</td>
<td>15%</td>
</tr>
<tr>
<td>Number of BUF/BUFCTRLs</td>
<td>1</td>
<td>22</td>
<td>3%</td>
</tr>
</tbody>
</table>

Figura 8.29 Consumo de recursos de la FPGA. Fuente: Modificada por el autor.
Figura 8.30 Consumo de recursos de la FPGA Virtex II 2vp50. Fuente: Modificada por el autor.

Device utilization summary:

Selected Device: 2vp50ff1152-0

<table>
<thead>
<tr>
<th>Logic Utilization</th>
<th>Used</th>
<th>Available</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Slices</td>
<td>1574</td>
<td>23615</td>
<td>6%</td>
</tr>
<tr>
<td>Number of Slice Flip Flops</td>
<td>2361</td>
<td>47232</td>
<td>4%</td>
</tr>
<tr>
<td>Number of 4 input LUTs</td>
<td>2747</td>
<td>47232</td>
<td>5%</td>
</tr>
<tr>
<td>Number used as logic</td>
<td>2729</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number used as Shift registers</td>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of IOs</td>
<td>190</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of bonded IOBs</td>
<td>190</td>
<td>692</td>
<td>14%</td>
</tr>
<tr>
<td>Number of GCLKs</td>
<td>1</td>
<td>16</td>
<td>6%</td>
</tr>
</tbody>
</table>

Figura 8.31 Consumo de recursos de la FPGA. Fuente: Modificada por el autor.

Tabla 8.6 Resultados de la implementación sobre la NetFPGA 1G. Fuente: Los Autores.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1649</td>
<td>122,8</td>
<td>500</td>
<td>0.3032</td>
</tr>
<tr>
<td>2</td>
<td>2747</td>
<td>124,8</td>
<td>750</td>
<td>0.2732</td>
</tr>
<tr>
<td>3</td>
<td>2800</td>
<td>124,5</td>
<td>800</td>
<td>0.2857</td>
</tr>
<tr>
<td>4</td>
<td>2747</td>
<td>124,6</td>
<td>790</td>
<td>0.2875</td>
</tr>
<tr>
<td>5</td>
<td>2747</td>
<td>124,5</td>
<td>780</td>
<td>0.2839</td>
</tr>
</tbody>
</table>
Figura 8.32 Consumo de recursos sobre la FPGA del Trivium_Core_Opt. Fuente: Modificada por el autor.

Device utilization summary:
--------------------------

Selected Device: 2vp58ff1152-6

- Number of Slices: 133 out of 23010 0%
- Number of Slice Flip Flops: 150 out of 47232 0%
- Number of 4 input LUTs: 260 out of 47232 0%
- Number of IOs: 164
- Number of bonded IOBs: 112 out of 692 16%
- Number of GCLKs: 1 out of 16 6%

--------------------------
Partition Resource Summary:
--------------------------

Figura 8.33 Consumo de recursos de la FPGA. Fuente: Modificada por el autor.
Trivium Generator

**Figura 8.34** Consumo de recursos de la FPGA xc2vp50. Fuente: Modificada por el autor.

<table>
<thead>
<tr>
<th>Logic Utilization</th>
<th>Used</th>
<th>Available</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Slices</td>
<td>947</td>
<td>23616</td>
<td>4%</td>
</tr>
<tr>
<td>Number of Slice Flip Flops</td>
<td>1286</td>
<td>47232</td>
<td>2%</td>
</tr>
<tr>
<td>Number of 4 input LUTs</td>
<td>1654</td>
<td>47232</td>
<td>3%</td>
</tr>
<tr>
<td>Number used as logic:</td>
<td>1652</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number used as Shift registers:</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of I/Os:</td>
<td>180</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of bonded I/Os:</td>
<td>89</td>
<td>out of 692</td>
<td>12%</td>
</tr>
<tr>
<td>Number of GCLKs:</td>
<td>1 out of 18</td>
<td>6%</td>
<td></td>
</tr>
</tbody>
</table>

**Figura 8.35** Consumo de recursos de la FPGA. Fuente: Modificada por el autor.
CAPÍTULO 9 IMPLEMENTACIÓN DEL CIFRADOR EN FLUJO TRIVIUM OPTIMIZADO BAJO LA PLATAFORMA NETFPGA 1G

Resumen del Capítulo

Este capítulo muestra el diseño del módulo optimizado y los resultados de la implementación sobre la plataforma NetFPGA 1G.

La arquitectura propuesta del nuevo módulo del Trivium Optimizado es sintetizada en el ISE de Xilinx y verificado usando ModelSim, en este capítulo, los resultados de varias pruebas sobre el módulo y la discusión sobre cada resultado serán explicados con más detalle.

Para cumplir con este objetivo fue necesario aplicar el mismo conjunto de pruebas realizado al diseño con el Trivium original para contar con elementos de comparación.

Resumen del diseño

Con el fin de implementar el diseño sobre la NetFPGA 1G, es importante determinar que la utilización de los recursos del módulo propuesto no exceda los recursos provistos y disponibles por la NetFPGA. La sobre utilización de recursos puede conllevar al fracaso en la implementación del diseño en la NetFPGA. Además, la utilización de recursos debe ser la primera verificación con el fin de no sobre utilizar los recursos previstos.

Las siguientes figuras muestran el consumo de recursos del diseño optimizado sobre las FPGAs.
TRIVIUM_CORE_OPT

Figura 9.1 Resumen del diseño del Trivium core optimizado sobre la NetFPGA 1G. Fuente: Modificada por el autor.

Device utilization summary:
---------------------------------
Selected Device : 2vp50ff1152-6

<table>
<thead>
<tr>
<th>Logic Utilization</th>
<th>Used</th>
<th>Available</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Slices</td>
<td>133</td>
<td>23616</td>
<td>0%</td>
</tr>
<tr>
<td>Number of Slice Flip Flops</td>
<td>159</td>
<td>47232</td>
<td>0%</td>
</tr>
<tr>
<td>Number of 4 Input LUTs</td>
<td>280</td>
<td>47232</td>
<td>0%</td>
</tr>
<tr>
<td>Number of I/Os</td>
<td>184</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of bended I/OBs</td>
<td>112</td>
<td>692</td>
<td>16%</td>
</tr>
<tr>
<td>Number of GCLKs</td>
<td>1</td>
<td>16</td>
<td>6%</td>
</tr>
</tbody>
</table>

---------------------------------

Figura 9.2 Consumo de recursos del Trivium core optimizado sobre la NetFPGA 1G. Fuente: Modificada por el autor.

De la figura anterior, el diseño no alcanza a utilizar el 1% de los Slices Flip Flop, y de los LUTs de 4 entradas. Este uso de recursos es mejor porque los recursos no usados pueden ser utilizados para la implementación de otros diseños en el futuro.
TRIVIUM GENERATOR OPT

Figura 9.3 Resumen del diseño del Trivium Generator optimizado sobre la NetFPGA 1G. Fuente: Modificada por el autor.

Device utilization summary:
-----------------------------------

Selected Device : 2vp50ff1152-6

Number of Slices: 947 out of 23616 4%
Number of Slice Flip Flops: 1286 out of 47232 2%
Number of 4 input LUTs: 1654 out of 47232 3%
Number used as logic: 1652
Number used as Shift registers: 2
Number of IOs: 100
Number of bonded IOBs: 89 out of 692 12%
Number of GCLKs: 1 out of 16 6%

Figura 9.4 Consumo de recursos del Trivium Generator Opt sobre la NetFPGA 1G. Fuente: Modificada por el autor.
De la figura anterior, el diseño alcanza a utilizar el 2% de los Slices Flip Flop, y el 3% de los LUTs de 4 entradas. Este uso de recursos permite que los recursos no usados puedan ser utilizados para la implementación de otros diseños en el futuro. La siguiente tabla muestra un resumen de los criterios definidos y hallados en el proceso de simulación y pruebas.

<table>
<thead>
<tr>
<th>Criterio</th>
<th>Desempeño</th>
</tr>
</thead>
<tbody>
<tr>
<td>Período mínimo</td>
<td>10 ns</td>
</tr>
<tr>
<td>Frecuencia máxima</td>
<td>125 MHz</td>
</tr>
<tr>
<td>Primer bloque de 8 bits cifrado</td>
<td>11575 ns</td>
</tr>
<tr>
<td>Siguiente bloque cifrado</td>
<td>Cada 10 ns</td>
</tr>
<tr>
<td>Tiempo de cifrado 1 Mbit</td>
<td>0,00125 s</td>
</tr>
<tr>
<td>Tiempo de cifrado 1 Gbit</td>
<td>1,25 s</td>
</tr>
</tbody>
</table>

Las figuras 9.5 y 9.6 muestran el diseño del módulo que será implementado en la NetFPGA, la arquitectura del diseño es conocida como módulo hace referencia a la estructura del pipeline modular de la NetFPGA, esto con el fin de cumplir con el diseño estándar para poder ser agregado al pipeline. El módulo incluye 3 partes principales:

1. **Input Processing Part** – Esta parte recibe el paquete del módulo previo y determina la ubicación del payload y lo envía al módulo Trivium_Optimizado para realizar la operación de cifrado.

2. **Trivium_Generator_Optimizado** – Este módulo a su vez contiene 8 instancias del módulo Trivium_Core, esta parte genera la secuencia cifrante para realizar la operación de cifrado con el payload que viene desde la input processing part, una vez terminado el proceso de cifrado, el payload será
enviado al output processin part. La implementación de la NIC (Network Interface Card), encripta en el proceso de transmisión y desencripta en el proceso de recepción.

3. Output Processing Part – Luego de que la operación de cifrado culmine, los datos del payload es enviado y recibido en esta parte. Los datos del payload serán enviados al siguiente bloque.

Figura 9.5 Arquitectura del módulo para encriptar. Fuente: Modificada por el autor.
Resultados y discusión

Es preciso afirmar que el esquema del módulo TRIVIUM_CORE optimizado posee la misma estructura que el TRIVIUM_CORE normal mostrado en el capítulo 8, la única diferencia son las funciones actualizadoras que pasa de tres a nueve. Las siguientes figuras muestran las diferencias en el código.
La siguiente figura muestra el pipeline de referencia con el módulo Trivium optimizado dentro del User Data Path.
Figura 9.9 Diseño sobre el pipeline de referencia de la NetFPGA con el módulo Trivium optimizado agregado. Fuente: Modificada por el autor.
En los enlaces referenciados se puede ver las simulaciones realizadas en el ISE de Xilinx y en Modelsim sobre el algoritmo optimizado.

Las siguientes figuras muestran los resultados de la simulación y la verificación de los módulos Trivium_Core_Opt y Trivium_Generator_Opt.

**Trivium_Core_Opt**

![Figura 9.10. Simulación del Trivium_Core_Opt I. Fuente: Los Autores.](image)

![Figura 9.11. Simulación del Trivium_Core_Opt II. Fuente: Los Autores.](image)

---

62 Canal de youtube grupo Nyquist, https://www.youtube.com/channel/UCV-8tsVhktFIatxEsCLC0LQ


La siguiente figura muestra en la línea 146 la variable KEY_OUT, la cual se encarga de almacenar la cadena cifrante.

**Trivium_Generator_Opt**

Figura 9.15 Ubicación de la cadena cifrante CORE_OUT. Fuente: Los Autores.

La siguiente figura muestra en la línea 236 la operación de cifrado entre la cadena cifrante y el texto en plano.
La siguiente figura muestra la variable CORE_OUT que almacena los 8 bits de la cadena cifrante.

Figura 9.17 Cadena cifrante CORE_OUT. Fuente: Los Autores.
La siguiente figura muestra las variables necesarias para el proceso de cifrado.

Figura 9.18 Señales Plaintext_in, Ciphertext_out y core_out. Fuente: Los Autores.

Figura 9.19 Arreglos Plaintext_in, Ciphertext_out y core_out con valores. Fuente: Los Autores.
Las siguientes figuras muestran el proceso de simulación del diseño optimizado.


El componente Trivium_Generator_Opt a los $1,1575 \times 10^{-5}$ segundos genera la secuencia cifrante y la primera salida cifrada de 8 bits y cada 10 ns, es decir, cada $1 \times 10^{-8}$ segundos genera la siguiente salida de 8 bits. Esto quiere decir que para generar 1Mbit se requiere cifrar 125.000 cadenas de 8 bits, lo cual se tardaría 0,00125 segundos. La siguiente tabla muestra en resumen los resultados de las pruebas realizadas al módulo sobre la plataforma.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1649</td>
<td>124,8</td>
<td>750</td>
<td>0.4548</td>
</tr>
<tr>
<td>2</td>
<td>1654</td>
<td>124,5</td>
<td>800</td>
<td>0.4836</td>
</tr>
<tr>
<td>3</td>
<td>1650</td>
<td>124,6</td>
<td>790</td>
<td>0.4787</td>
</tr>
<tr>
<td>4</td>
<td>1654</td>
<td>124,5</td>
<td>780</td>
<td>0.4715</td>
</tr>
<tr>
<td>5</td>
<td>1650</td>
<td>125</td>
<td>780</td>
<td>0.4727</td>
</tr>
</tbody>
</table>

La siguiente figura muestra una comparación del consumo de recursos de la implementación original y la optimizada.
Figura 9.25 Comparación del consumo de recursos del Trivium_Core y el Trivium_Core_Opt.
Fuente: Los Autores.

Tabla 9.3 Resultados de las pruebas del módulo. Fuente: Los Autores.

<table>
<thead>
<tr>
<th>Uso del dispositivo xc3s500e</th>
<th>Implementación</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Trivium_Core</td>
</tr>
<tr>
<td>Area (slices)</td>
<td>238 de 4656</td>
</tr>
<tr>
<td></td>
<td>5%</td>
</tr>
<tr>
<td>4 input LUTs</td>
<td>457 de 9312</td>
</tr>
<tr>
<td></td>
<td>5%</td>
</tr>
<tr>
<td>Slices / Flip Flops</td>
<td>288 de 9312</td>
</tr>
<tr>
<td></td>
<td>3%</td>
</tr>
<tr>
<td>Bonded IOBs</td>
<td>164</td>
</tr>
</tbody>
</table>
Tabla 9.4 Resultados de las pruebas del módulo (2). Fuente: Los Autores.

<table>
<thead>
<tr>
<th>Uso del dispositivo</th>
<th>Implementación</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Trivium_Generator</td>
</tr>
<tr>
<td>Area (slices)</td>
<td>1574 de 23616</td>
</tr>
<tr>
<td>4 input LUTs</td>
<td>2361 de 47232</td>
</tr>
<tr>
<td>Slices Flops</td>
<td>288 de 47232</td>
</tr>
<tr>
<td>Bonded IOBs</td>
<td>100 de 692</td>
</tr>
</tbody>
</table>

Como resultados adicionales del proceso de investigación se obtuvieron los siguientes productos:

- Ponencia en el 11 Congreso Colombiano de Computación sobre la propuesta de optimización del algoritmo del cifrador en flujo Trivium para aplicaciones en tiempo real sobre la plataforma NetFPGA 1G.

- Repositorio abierto en GitHub que contiene el código fuente en su totalidad y que puede ser utilizado para futuras investigaciones.

- Videos de la simulación en el canal del grupo Nyquist en youtube ⁶³.

---

⁶³ https://www.youtube.com/channel/UCV-8tsVhkTFiatxEsCLCOLQ
Posibles Usos del proyecto

➢ Intercambio Información Peer to Peer (P2P).

➢ Almacenamiento distribuido en ambiente de computación en la nube.

➢ Cifrado de datos privados y sensibles (Salud, de menores de edad).

➢ Banco de cadenas de secuencia cifrante prealmacenadas para procesos de cifrado.
CAPÍTULO 10 CONCLUSIONES

Cuando se trabaja con un LFSRs o un NLFSRs en una configuración Fibonacci, se crea un cuello de botella al calcular la salida debido a que la key stream arrojada por estos shift registers es la sumatoria de todos los elementos que influyen en la salida, por lo tanto, el algoritmo no podrá correr a frecuencias más altas debido a que la ruta crítica determinará la velocidad de ejecución. Por otra parte, en una configuración Galois NLFSR, la salida está divida en pequeños segmentos reduciendo de esta manera la ruta crítica, dando la posibilidad de ejecutar el algoritmo a velocidades más altas. Adicionalmente trabajar con algoritmos empleando una configuración Galois, permite que tarjetas de baja capacidad como la NetFPGA 1G, o cualquier otra, pueda optimizarse a nivel físico para obtener resultados similares a los que se conseguirían con equipos de mayor costo, ayudando así a reciclar parte de la tecnología existente sin sacrificar eficiencia.

A través de la implementación del diseño original y optimizado se pudo validar que para definir un diseño específico de hardware, se requiere considerar los aspectos asociados a la plataforma de implementación.

Con respecto al resultado de las simulaciones y las pruebas realizadas se pudo establecer que el consumo de recursos de la implementación optimizada consume menos recursos de hardware que la implementación original.

En cuanto al tiempo de generación de la cadena cifrante y el proceso de cifrado fue muy similar en ambas implementaciones, para cifrar una cadena de 1 Gbit se requiere 1,25 segundos, es decir, que el throughput es 800 Mbits/s.
En contraste con los objetivos específicos planteados para el desarrollo del proyecto, se puede notar que cada uno de ellos fue alcanzado, los productos resultado de cada uno puede verificarse en el repositorio del proyecto, y adicionalmente como parte del contenido del presente documento.
CAPÍTULO 11 TRABAJO FUTURO

Como trabajos futuros para el presente proyecto se tiene:

- Desarrollar un algoritmo que permita el uso eficiente de la optimización lograda, con relación a este trabajo, es necesario desarrollar un algoritmo que contemple las otras conexiones que se pueden dar, pero sin afectar la parte teórica del algoritmo original.

- Realizar la paralelización del algoritmo, debido a que el Trivium es altamente paralelizable, se podría preprocesar bits o realizar tareas en simultáneo antes de llegue el procesamiento y así reducir los tiempos de generación de la cadena cifrante y de cifrado de datos.

- Desarrollar un algoritmo que permita seleccionar la mejor optimización, como las funciones actualizadoras puede ser diferentes, se requiere de un algoritmo que pueda seleccionar la configuración más óptima de las funciones actualizadoras.
CAPÍTULO 12 RECURSOS

12.1 Técnicos:

- 5 equipos de cómputo
- Sala Parquesoft
- Sistema operativo CentOS 6.3 x86 – Fedora 13
- 4 tarjetas NetFPGA 1G
- Comunidad NetFPGA.org (proyectos contribuidos y de referencia, foro, artículos científicos, wiki)
- Acceso a base de datos científicas (IEEE, Proquest, ScienceDirect) UTP
- Acceso a internet
- Seminario NetFPGA, comunidad NetFPGA Latinoamérica
<table>
<thead>
<tr>
<th>Cronograma de Actividades</th>
<th>Mes 1</th>
<th>Mes 2</th>
<th>Mes 3</th>
<th>Mes 4</th>
<th>Mes 5</th>
<th>Mes 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Realizar un estudio sobre el funcionamiento de los cifradores en flujo.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 Implementar el cifrador en flujo Trivium bajo la plataforma NetFPGA 1G.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 Optimizar el cifrador en flujo Trivium utilizando un proceso de transformación de un Fibonacci NLFSR a un Galois NLFSR y posteriormente extender la transformación de Fibonacci-Galois a un caso más general de shift registers Galois-Galois con conexiones feedback y feedforward.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 Implementar el cifrador en flujo Trivium optimizado bajo la plataforma NetFPGA 1G.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 Comparar resultados.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
163

CAPÍTULO 14 REFERENCIAS BIBLIOGRÁFICAS


[17] CAMACHO, Blanca Nydia Pérez; CUMPLIDO, René. Comparación de la Eficiencia en Hardware de los Cifradores de Flujo Grain, Mickey-128 y Trivium de Ecrypt.


[34] HAMANN, Matthias; KRAUSE, Matthias. Stream Cipher Operation Modes with Improved Security against Generic Collision Attacks.


[42] SANTIAGO, SANDRA DÍAZ. GENERACION DE SUCESIONES CRIPTOGRAFICAMENTE FUERTES.


[51] United States Patent and Trademark Office, <http://pdfpiw.uspto.gov/piw?Docid=01310719&homeurl=http%3A%2F%2Fpatft.uspto.gov%2Fnetacgi%2Fnph-Parser%3FSect1%3DPTO1%2526Sect2%3DHITOFF%2526d%3DPALL%2526p%3D1%2526u%3D%2526Fnetahml%252FPTO%25252Fsrchnum.htm%2526r%3D1%2526f%3DG%25263D5
[52] BELLON, Steven M. “The one-time pad and the index of coincidence”,


[54] Silicon, “Descubren una vulnerabilidad que afecta al protocolo de cifrado web HTTPS”,


[58] NetFPGA, <http://netfpga.org/site/#/>, [on line], 2015