Analysis of Stream Cipher Based Authenticated Encryption Schemes

by

Mufeed Juma ALMashrafi

Bachelor of Engineering (Electrical and Electronics) (Sultan Qaboos University) – 1998
Master of Science (Computer Science) (Sultan Qaboos University) – 2001

Thesis submitted in accordance with the regulations for the Degree of Doctor of Philosophy

Information Security Institute
Science and Engineering Faculty
Queensland University of Technology

October 22, 2012
Keywords

Authenticated encryption, stream cipher, symmetric cipher, cryptanalysis, algebraic analysis, system of equations, SSS, ZUC, MACs, Grain-128a, 128-EIA, 128-EEA, multivariate equations, Sfinks, SNOW-3G, SOBER-128, NLSv2, UEA2, UIA2, birthday attacks, forgery attacks, algebraic attacks, collision attacks, side-channel attacks.
Abstract

Authenticated Encryption (AE) is the cryptographic process of providing simultaneous confidentiality and integrity protection to messages. This approach is more efficient than applying a two-step process of providing confidentiality for a message by encrypting the message, and in a separate pass providing integrity protection by generating a Message Authentication Code (MAC). AE using symmetric ciphers can be provided by either stream ciphers with built in authentication mechanisms or block ciphers using appropriate modes of operation. However, stream ciphers have the potential for higher performance and smaller footprint in hardware and/or software than block ciphers. This property makes stream ciphers suitable for resource constrained environments, where storage and computational power are limited. There have been several recent stream cipher proposals that claim to provide AE. These ciphers can be analysed using existing techniques that consider confidentiality or integrity separately; however currently there is no existing framework for the analysis of AE stream ciphers that analyses these two properties simultaneously. This thesis introduces a novel framework for the analysis of AE using stream cipher algorithms.

This thesis analyzes the mechanisms for providing confidentiality and for providing integrity in AE algorithms using stream ciphers. There is a greater emphasis on the analysis of the integrity mechanisms, as there is little in the public literature on this, in the context of authenticated encryption. The thesis has four main contributions as follows.

The first contribution is the design of a framework that can be used to classify AE stream ciphers based on three characteristics. The first classification applies Bellare and Namprempre’s work on the the order in which encryption and authentication processes take place. The second classification is based on the method used for accumulating the input message (either directly or indirectly) into the into the internal states of the cipher to generate a MAC. The
third classification is based on whether the sequence that is used to provide encryption and authentication is generated using a single key and initial vector, or two keys and two initial vectors.

The second contribution is the application of an existing algebraic method to analyse the confidentiality algorithms of two AE stream ciphers; namely SSS and ZUC. The algebraic method is based on considering the nonlinear filter (NLF) of these ciphers as a combiner with memory. This method enables us to construct equations for the NLF that relate the (inputs, outputs and memory of the combiner) to the output keystream. We show that both of these ciphers are secure from this type of algebraic attack. We conclude that using a key-dependent SBox in the NLF twice, and using two different SBoxes in the NLF of ZUC, prevents this type of algebraic attack.

The third contribution is a new general matrix based model for MAC generation where the input message is injected directly into the internal state. This model describes the accumulation process when the input message is injected directly into the internal state of a nonlinear filter generator. We show that three recently proposed AE stream ciphers can be considered as instances of this model; namely SSS, NLSv2 and SOBER-128. Our model is more general than a previous investigations into direct injection. Possible forgery attacks against this model are investigated. It is shown that using a nonlinear filter in the accumulation process of the input message when either the input message or the initial states of the register is unknown prevents forgery attacks based on collisions.

The last contribution is a new general matrix based model for MAC generation where the input message is injected indirectly into the internal state. This model uses the input message as a controller to accumulate a keystream sequence into an accumulation register. We show that three current AE stream ciphers can be considered as instances of this model; namely ZUC, Grain-128a and Sfinks. We establish the conditions under which the model is susceptible to forgery and side-channel attacks.
# Contents

**Front Matter**

- Keywords .......................................................... iii
- Abstract ............................................................ v
- Table of Contents .................................................. vii
- List of Figures ..................................................... xiii
- List of Tables ....................................................... xv
- List of Algorithms ................................................ xvi
- Notation .............................................................. xix
- Declaration .......................................................... xxii
- Previously Published Material ................................. xxiii
- Acknowledgements ................................................ xxv

## 1 Introduction

1.1 Aims and objectives of thesis ............................... 3
1.2 Contributions and achievements ............................ 3
1.3 Outline of thesis ............................................... 5

## 2 Background

2.1 Confidentiality using stream ciphers ...................... 7
  2.1.1 Stream ciphers ........................................ 8
      Synchronous stream ciphers .............................. 9
      Self-synchronous stream ciphers ........................ 10
  2.1.2 Cryptanalysis of stream ciphers .................... 12
      Attack measurement .................................... 13
      Main goals of attacks ................................ 13
  2.1.3 Cryptanalytic techniques for stream ciphers ....... 14
      Time-Memory-Data trade off attacks .................. 15
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Divide and conquer attacks</td>
<td>16</td>
</tr>
<tr>
<td>Correlation attacks</td>
<td>16</td>
</tr>
<tr>
<td>Fast correlation attacks</td>
<td>17</td>
</tr>
<tr>
<td>Algebraic attacks</td>
<td>19</td>
</tr>
<tr>
<td>2.2 Integrity and Message Authentication Codes (MACs)</td>
<td>20</td>
</tr>
<tr>
<td>2.2.1 Message Authentication Codes (MACs)</td>
<td>21</td>
</tr>
<tr>
<td>Properties of MAC functions</td>
<td>22</td>
</tr>
<tr>
<td>2.2.2 Statistical analysis</td>
<td>22</td>
</tr>
<tr>
<td>2.2.3 Cryptanalysis of MACs</td>
<td>24</td>
</tr>
<tr>
<td>Attack models on MACs</td>
<td>24</td>
</tr>
<tr>
<td>Attack goals on MACs</td>
<td>25</td>
</tr>
<tr>
<td>2.3 Summary and conclusion</td>
<td>26</td>
</tr>
<tr>
<td>3 Authenticated encryption using stream ciphers</td>
<td>27</td>
</tr>
<tr>
<td>3.1 Authenticated encryption using stream ciphers</td>
<td>28</td>
</tr>
<tr>
<td>3.2 Classification of methods for providing AE</td>
<td>31</td>
</tr>
<tr>
<td>3.2.1 The ordering of encryption and authentication</td>
<td>32</td>
</tr>
<tr>
<td>3.2.2 Message injection</td>
<td>35</td>
</tr>
<tr>
<td>3.2.3 Number of key-IV pairs used</td>
<td>37</td>
</tr>
<tr>
<td>3.3 Description of specific AE stream ciphers</td>
<td>37</td>
</tr>
<tr>
<td>3.3.1 SSS</td>
<td>38</td>
</tr>
<tr>
<td>3.3.2 ZUC and 128-EIA3</td>
<td>42</td>
</tr>
<tr>
<td>3.3.3 Sfinks</td>
<td>49</td>
</tr>
<tr>
<td>3.3.4 NLSv2</td>
<td>52</td>
</tr>
<tr>
<td>3.3.5 SOBER-128</td>
<td>55</td>
</tr>
<tr>
<td>3.3.6 Grain-128a</td>
<td>57</td>
</tr>
<tr>
<td>3.3.7 SNOW 3G</td>
<td>61</td>
</tr>
<tr>
<td>3.4 Summary and Conclusion</td>
<td>65</td>
</tr>
<tr>
<td>4 Algebraic analysis of SSS and ZUC stream ciphers</td>
<td>67</td>
</tr>
<tr>
<td>4.1 Modeling a combiner with memory</td>
<td>68</td>
</tr>
<tr>
<td>4.2 Algebraic analysis of SSS</td>
<td>69</td>
</tr>
<tr>
<td>4.2.1 Description of SOBER-t32 keystream generator</td>
<td>71</td>
</tr>
<tr>
<td>4.2.2 Algebraic attack on modified SOBER-t32</td>
<td>73</td>
</tr>
<tr>
<td>4.2.3 Applying algebraic attack on SSS</td>
<td>75</td>
</tr>
<tr>
<td>Constructing multivariate equations for NLF</td>
<td>75</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>5.5</td>
<td>Summary and conclusion</td>
</tr>
<tr>
<td>6</td>
<td>General model for MAC generation using indirect injection</td>
</tr>
<tr>
<td>6.1</td>
<td>MAC generation using indirect message injection</td>
</tr>
<tr>
<td>6.1.1</td>
<td>Structure of the integrity algorithm</td>
</tr>
<tr>
<td>6.1.2</td>
<td>Optional processes</td>
</tr>
<tr>
<td>6.2</td>
<td>Current proposals using this model</td>
</tr>
<tr>
<td>6.2.1</td>
<td>128-EIA3 version 1.4</td>
</tr>
<tr>
<td>6.2.2</td>
<td>128-EIA3 version 1.5</td>
</tr>
<tr>
<td>6.2.3</td>
<td>Grain-128a</td>
</tr>
<tr>
<td>6.2.4</td>
<td>Sfinks</td>
</tr>
<tr>
<td>6.3</td>
<td>Security analysis against forgery attacks</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Security of the accumulation process</td>
</tr>
<tr>
<td>6.3.1.1</td>
<td>Bit flipping forgeries</td>
</tr>
<tr>
<td>6.3.1.2</td>
<td>Bit deletion forgeries</td>
</tr>
<tr>
<td>6.3.1.3</td>
<td>Bit insertion forgeries</td>
</tr>
<tr>
<td>6.3.2</td>
<td>Security considerations for masking vector $A_0 \oplus F$</td>
</tr>
<tr>
<td>6.3.3</td>
<td>Security analysis of existing ciphers</td>
</tr>
<tr>
<td>6.3.3.1</td>
<td>Previous attack on 128-EIA3 version 1.4</td>
</tr>
<tr>
<td>6.3.3.2</td>
<td>New attack on 128-EIA3 version 1.4</td>
</tr>
<tr>
<td>6.3.3.3</td>
<td>128-EIA3 version 1.5</td>
</tr>
<tr>
<td>6.3.3.4</td>
<td>Grain-128a</td>
</tr>
<tr>
<td>6.3.3.5</td>
<td>Sfinks</td>
</tr>
<tr>
<td>6.4</td>
<td>Security analysis against timing and power attacks</td>
</tr>
<tr>
<td>6.5</td>
<td>Summary and conclusion</td>
</tr>
<tr>
<td>7</td>
<td>Conclusions and future research</td>
</tr>
<tr>
<td>7.1</td>
<td>Review of Contributions</td>
</tr>
<tr>
<td>7.1.1</td>
<td>Classifications of existing AE stream ciphers</td>
</tr>
<tr>
<td>7.1.2</td>
<td>Algebraic analysis of two AE stream ciphers</td>
</tr>
<tr>
<td>7.1.3</td>
<td>General model for MAC generation using direct injection of the input message</td>
</tr>
<tr>
<td>7.1.4</td>
<td>General model for MAC generation using indirect injection of the input message</td>
</tr>
<tr>
<td>7.2</td>
<td>Future research</td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Model for providing confidentiality</td>
<td>8</td>
</tr>
<tr>
<td>2.2</td>
<td>Structure of synchronous stream cipher</td>
<td>11</td>
</tr>
<tr>
<td>2.3</td>
<td>Structure of self-synchronous stream cipher</td>
<td>11</td>
</tr>
<tr>
<td>2.4</td>
<td>Model for providing integrity</td>
<td>21</td>
</tr>
<tr>
<td>3.1</td>
<td>Model for providing authenticated encryption</td>
<td>29</td>
</tr>
<tr>
<td>3.2</td>
<td>General structure for AE algorithms</td>
<td>30</td>
</tr>
<tr>
<td>3.3</td>
<td>MAC generation using the input message directly</td>
<td>36</td>
</tr>
<tr>
<td>3.4</td>
<td>MAC generation using indirect message injection</td>
<td>37</td>
</tr>
<tr>
<td>3.5</td>
<td>SSS keystream generator and MAC generation</td>
<td>38</td>
</tr>
<tr>
<td>3.6</td>
<td>Non-Linear Filter (NLF) of SSS</td>
<td>40</td>
</tr>
<tr>
<td>3.7</td>
<td>Structure of ZUC stream cipher</td>
<td>43</td>
</tr>
<tr>
<td>3.8</td>
<td>128-EEA3 and 128-EIA3</td>
<td>46</td>
</tr>
<tr>
<td>3.9</td>
<td>Sfinks keystream generator and MAC generation</td>
<td>49</td>
</tr>
<tr>
<td>3.10</td>
<td>NLSv2 keystream generator and MAC generation</td>
<td>52</td>
</tr>
<tr>
<td>3.11</td>
<td>SOBER-128 keystream generator and MAC generation</td>
<td>55</td>
</tr>
<tr>
<td>3.12</td>
<td>Grain-128a keystream generator and MAC generation</td>
<td>58</td>
</tr>
<tr>
<td>3.13</td>
<td>SNOW 3G keystream generator</td>
<td>61</td>
</tr>
<tr>
<td>4.1</td>
<td>SOBER-t32 keystream generator</td>
<td>71</td>
</tr>
<tr>
<td>4.2</td>
<td>Non-Linear Filter (NLF) of SOBER-t32</td>
<td>72</td>
</tr>
<tr>
<td>4.3</td>
<td>Combiner structure of SOBER-t32</td>
<td>74</td>
</tr>
<tr>
<td>4.4</td>
<td>Combiner structure of SSS</td>
<td>77</td>
</tr>
<tr>
<td>4.5</td>
<td>Possible valid combiner structure for NLF of SSS</td>
<td>78</td>
</tr>
<tr>
<td>4.6</td>
<td>Modified structure of SSS keystream generator</td>
<td>81</td>
</tr>
<tr>
<td>4.7</td>
<td>NLF combiner</td>
<td>86</td>
</tr>
<tr>
<td>4.8</td>
<td>ZUC NonLinear function $f$</td>
<td>87</td>
</tr>
<tr>
<td>4.9</td>
<td>Combiner of ZUC</td>
<td>88</td>
</tr>
</tbody>
</table>
5.1 General model for MAC generation using the input message directly 95
5.2 Message accumulation mechanism in SSS . . . . . . . . . . . . . . 106
5.3 Message accumulation mechanism in NLSv2 . . . . . . . . . . . . 108
5.4 Message accumulation mechanism in SOBER-128 . . . . . . . . 109

6.1 MAC generation using indirect message injection . . . . . . . . 122
6.2 Structure of 128-EIA3 . . . . . . . . . . . . . . . . . . . . . . . 125
6.3 Structure of MAC generation using Grain-128a . . . . . . . . . . 127
6.4 Structure of MAC generation using Sfinks . . . . . . . . . . . . . 128
6.5 Type of forgery attacks on our model . . . . . . . . . . . . . . . 130
6.6 Modified MAC generation model using indirect message injection 138
List of Tables

2.1 Minimum data required to run repetition test $S$ . . . . . . . . . . 24

3.1 Summary of the three methods of ordering encryption and authen-
tication . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35

3.2 Summary of the results for 128-EIA3 algorithm . . . . . . . . . . 47

3.3 Summary of the results for UIA2 algorithm . . . . . . . . . . . . . 64

3.4 Summary of the classification for several AE ciphers . . . . . . . . 66

4.1 Algebraic attack on modified version of ZUC . . . . . . . . . . . . 90

5.1 Summary of the analysis for MAC generation using direct message
injection. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 115
List of Algorithms

3.1 Sending and receiving processes in Encrypt-and-MAC scheme 33
3.2 Sending and receiving processes in MAC-then-Encrypt scheme 34
3.3 Sending and receiving processes in Encrypt-then-MAC scheme 35
Notation

The following notation will be used consistently throughout this thesis:

- $\oplus$: XOR operation.
- $+$: Addition modulo the word size.
- $x||y$: The concatenation of two binary strings $x$ and $y$.
- $W^t$: Word at time $t$.
- $W \ll i$ and $W \gg i$: The rotation of word $W$ by $i$ bits to the left and right, respectively.
- $W \ll i$ and $W \gg i$: The shift of word $W$ by $i$ bits to the left and right, respectively.
- $W^t_i$: The $i$-th bit of word $W$ at time $t$.
- $R[i]^t$: The $i$-th stage of register $R$ at time $t$.
- $R[i]^t_j$: The $j$-th bit position of stage $i$ in register $R$ at time $t$.
- $R[i]^t_{j\rightarrow e}$: A segment of consecutive bits in stage $i$ of register $R$, from the $j$-th to $e$-th bits of stage $i$ at time $t$.
- $P^t$: The plaintext word at time $t$.
- $Z^t$: The output keystream word at time $t$.
- $K$: The key.
- $IV$: Initialisation Vector.
- $C^t$: The ciphertext word at time $t$. 
- LFSR: Linear Feedback Shift Register.
- $X_H$: The Most Significant Byte (MSB) of word $X$.
- $X_L$: The Least Significant Byte (LSB) of word $X$. 
Declaration

The work contained in this thesis has not been previously submitted for a degree or diploma at any higher education institution. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made.

Signed: .................................................. Date: .........................
Previously Published Material

The following papers have been published or presented, and contain material based on the content of this thesis.


Acknowledgements

I would like to acknowledge with gratitude the many people whose support enabled me to complete the work presented in this thesis.

My grateful thanks to my Principal Supervisor Dr. Leonie Simpson, for the guidance, encouragement and enthusiastic support over my PhD journey.

Many thanks and grateful to my Associate Supervisors Prof. Ed Dawson, Dr. Harry Bartlett and Dr. Kenneth Wong, for giving constructive comments and support to my work. I really enjoy working with them during my research period.

Many thanks to Omani government that provided the scholarship and gave me this opportunity to complete my PhD degree.

Thanks also for the willing help, entertaining encouragement and intelligent friendship of many others, who are or have been within the ISI, including Muhammad Reza Z’aba, Huseyin Hisil, Sui Guan Teo, Ali Alhamdan, Farzad Salem, Mark ,Kenneth Radke, Esa AlSulemi, Christine Orme, Lakshmi Kuppusamy, Jothi Rangasamy and Chai Wen Chuah.

Many thanks to all members of my family and special thank to my father and my brothers Ali, Mahmoud, Muhammed, Said and Salim and my sisters. Also many thanks to all members of my family in law. Thank to my friend in Omani society in Queensland who have helped me to maintain a life outside the university and have been invaluable in keeping things in perspective.

Finally, my deepest gratitude to my wife, for continuous support, encouragement and tolerance, and for having the confidence in my ability to complete this thesis that I lacked. And to our lovely children, Mahnad, ALZahara, Mohammed and Majid for keeping me active to complete this thesis.
Chapter 1

Introduction

Information technology is widely used for storage and transmission of information, sometimes over insecure networks such as the internet, or via satellites. Some examples of applications where sensitive information is sent over insecure channels include smart cards to protect information for financial transactions, web browsing, mobile phones, RFID transponders, devices in sensor networks and electronic identity cards. To protect the information, these applications require algorithms with small implementation cost and minimal storage requirement which also operate at maximum speed and provide a high level of security.

Cryptographic mechanisms are used to provide information security. Aspects of information security include confidentiality and data integrity [88]. Confidentiality means keeping the message information secret from all people except the intended receiver. This is provided by using encryption algorithms. Integrity means that the received message is sent by the expected sender and no changes have been made to the message during the transmission over an insecure channel. Assurances of integrity can be provided by authentication algorithms such as keyed hash function, block ciphers in certain mode of operations, and more recently by stream ciphers with built in authentication mechanisms [72, p4].

Previously, to provide both confidentiality and integrity using symmetric key encryption required the application of two layers [11, 12]. In one layer, encryption is provided for the input message to provide confidentiality. In another layer, the message is processed to generate a Message Authentication Code (MAC) tag to provide integrity. The same cryptographic mechanism could be used
to perform both operations, but not simultaneously. For example, the Cipher Block Chaining (CBC) mode \cite{72} of operation for block ciphers provides an \( n \)-block message with confidentiality at cost \( n \), and with authentication with cost \( n \) also. Thus the two-pass process has a computational cost of \( 2n \), which may be excessive for high throughput infrastructure or constrained environments with low processing power. Consequently in these systems the provision of either integrity or confidentiality may be abandoned, greatly increasing the security vulnerability.

Since some of the operations performed in each layer are similar, to provide a high level of security with increased speed and reduced storage requirements it may be possible to perform the calculation for both confidentiality and integrity with the same cipher in the same operation. This is referred to as authenticated encryption (AE). The goal of AE is to provide confidentiality and integrity for messages simultaneously, with much less effort than a two-pass process. Symmetric ciphers providing authenticated encryption could be either stream ciphers with built in authentication mechanisms or block ciphers using appropriate modes of operation.

In 2000, Bellare and Namprempre \cite{11,12} presented a security analysis for authenticated encryption schemes based on generic compositions of a symmetric cipher and a MAC algorithm, and provided security proofs for certain compositions. In 2001, a mode of operation for block ciphers called Integrity Aware Cipher Block Chaining (IACBC) was introduced \cite{65}, providing authenticated encryption for an \( n \) block of plaintext message with \( n + \log n \) block encryptions, much less than the \( 2n \) block encryptions required for the traditional two-pass system. Other proposals for block cipher modes providing authenticated encryption with a similar reduction in complexity include Offset Code Book (OCB) mode \cite{82} and Galois/Counter mode (GCM) \cite{35}.

To provide confidentiality, most of the AE primitives based on block ciphers use the block cipher in modes which effectively turn the block cipher into a keystream generator for a stream cipher \cite{84}. Dedicated stream ciphers have the potential for higher performance and smaller footprint (in hardware and software) than block ciphers, making them ideal for use in resource constrained environments where space and power are limited. They may be possible to provide both confidentiality and integrity with the cost of \( n \) operations for an \( n \) block of plaintext message. So, there is significant interest in the development
of stream ciphers which provide authenticated encryption.

Between 2004 and 2008, Europe launched a project, called the ECRYPT Stream Cipher Project eSTREAM [78], to identify new stream ciphers suitable for widespread adoption. The eSTREAM project included a call for AE stream ciphers. Seven of the 34 stream ciphers submitted the project claimed to offer authenticated encryption [1, 18, 43, 52, 54–56, 74, 79, 85, 99]. However, all of these were flawed [14, 22, 23, 25, 32, 64, 60, 70, 75, 87, 98, 101].

Recently, the sequences produced using stream cipher keystream generators have been incorporated other mechanisms and used to provide both confidentiality and integrity in a single pass, but with two different keys and IVs, in proposals intended for application in mobile phone networks. Examples of this approach are SNOW 3G [93] which uses UEA2 and UIA2 [92] algorithms to provide confidentiality and integrity respectively, and ZUC [38] which uses 128-EEA3 and 128-EIA3 [40] algorithms to provide confidentiality and integrity respectively.

### 1.1 Aims and objectives of thesis

The main aim of this thesis is to analyse proposed methods for providing authenticated encryption using symmetric stream ciphers. The plan will be to analyse both the confidentiality and integrity components. In particular the main direction of research will be to analyse integrity component since the use of stream ciphers is relatively new in this context. The motivation behind this analysis is that there are many real world applications for which AE stream ciphers could be useful. However, there are few design criteria for AE stream ciphers; most of the designs are ad hoc. So it is essential to develop a framework for the design and analysis of AE stream ciphers.

Furthermore, the thesis aims to establish a set of design guidelines for ciphers providing AE in a secure manner using stream ciphers.

### 1.2 Contributions and achievements

This thesis has four major contributions:

1. **Classifications of existing AE stream ciphers.** We classify seven existing AE stream ciphers based on three characteristics. The first characteristic is whether the confidentiality and integrity are performed using a single
Chapter 1. Introduction

key and $IV$, or two distinct keys and $IV$s. The second characteristic is the order in which the encryption and authentication processes are performed. The third characteristic is the way in which the message is accumulated into the internal states of the cipher in order to generate a MAC tag. Generally, there are two methods for message injection into the internal states of the cipher: direct or indirect. This classification schemes can be used as a framework to assist in the analysis of AE stream ciphers.

2. Algebraic analysis of two AE stream ciphers. We analyse both the SSS and ZUC stream ciphers using algebraic techniques. The technique used in an existing successful attack on SOBER-t32, which has similar structure to SSS, was applied. This method uses indirect relationships to construct equations for the nonlinear filter (NLF) of the ciphers, by considering the NLF as a combiner and trying to find relations between the initial states of the register and the keystream outputs. For both SSS and ZUC the attack was not successful, in that the results obtained were worse than exhaustive key search. However, the algebraic attacks on ZUC were more efficient than the previous algebraic analysis presented in [24] and [67]. For both ciphers the way of SBoxes are used in the cipher designs provided resistance to algebraic attack. In the case of SSS the use of a key dependent SBox prevents this attack. For ZUC the use of two different SBoxes in the NLF prevents the attack. This analysis is focussed on the confidentiality mechanism for the AE stream ciphers, to identify the weakness when it combined with the integrity mechanism.

3. General model for MAC generation using direct injection of the input message. We develop a novel general matrix-based model for the accumulation phase of MAC generation, where the MAC is formed using a nonlinear filter keystream generators and the message is injected directly into the internal state of the generator. Nakano et al. [76] described two methods for injecting the input message directly into internal states of a nonlinear filter generator to generate a hash value. We extend their model to generate a MAC tag instead of a hash value, and further extend our model from bit-based to word-based registers. The model is applied to existing AE stream cipher proposals. The matrix formulation permits extensive analysis of our model with respect to forgery attacks based on collisions. Based on the analyses, design recommendations are given to enhance the
security of proposals which follow this model to generate a MAC tag. The analysis is focused on the integrity mechanism for the AE stream cipher to identify the weakness when the integrity and confidentiality mechanisms are combined in order to provide AE.

4. General model for MAC generation using indirect injection of the input message. We develop a novel general matrix based model for message injection indirectly into the internal states of the cipher to generate a MAC tag using stream ciphers. In this model the input message is not accumulate into the internal state of the cipher. The role of the message is to control which segments of keystream sequence have to be accumulated into the internal states. Instead, the model is applied to existing AE stream cipher proposals. This model is analysed with respect to forgery and side-channel attacks. Based on the analyses, design recommendations are given to enhance the security of proposals following this model to generate a MAC tag. This analysis also is focused on the integrity mechanism for the AE stream cipher to identify the weakness when the integrity and confidentiality mechanisms are combined in order to provide AE.

1.3 Outline of thesis

This thesis is organised as follows:

- Chapter 2 describes the background information necessary to provide the context for the subsequent chapters. The information includes methods for providing confidentiality using stream ciphers, cryptanalytic techniques for stream ciphers, methods for providing integrity using message authentication codes (MACs) and cryptanalysis of MACs. This chapter gives the necessary background of confidentiality and integrity mechanisms using symmetric keys separately.

- Chapter 3 describes methods for providing AE using stream ciphers. This chapter also includes three methods for classifying ciphers providing AE based on different characteristics. Seven AE stream ciphers are described and we classify each of these ciphers using each of the three classification methods. Some of the results in this chapter are published in [97]. This
Chapter 1. Introduction

Chapter combines the two mechanisms within only one mechanism and gives some examples of these ciphers that provide AE with single mechanism.

- Chapter 4 presents algebraic analyses of the SSS and the ZUC stream ciphers. This chapter provides the necessary tools to apply algebraic attacks to the keystream generators. In each case, the analyses consider the non-linear filter of the keystream generator as a combiner with memory. The results of this chapter appear in two published papers [2, 4]. This chapter is focused on the confidentiality mechanism for the AE stream ciphers.

- Chapter 5 introduces a general matrix-based model for MAC generation using stream ciphers where the input message is injected directly into the internal state of the keystream generators. This chapter is focused on the integrity mechanism for the AE stream ciphers.

- Chapter 6 presents a general matrix based model for MAC generation in stream ciphers where the input message is used indirectly. The main results of this chapter are included in [3]. This chapter also is focused on the integrity mechanism for the AE stream ciphers using different way of injected the input message into the internal states.

- Chapter 7 presents conclusions about the work presented in this thesis. In addition, some future directions for exploration are proposed.
Chapter 2

Background

This chapter presents the background information necessary to support the research described in this thesis. This includes an overview of methods to provide confidentiality using stream ciphers, integrity using MAC generation algorithms and techniques for the cryptanalysis of stream ciphers and MAC generation algorithms. The chapter is organised as follows. Section 2.1 describes methods for providing confidentiality using stream ciphers and the related analytical techniques. Methods for providing integrity using message authentication codes are described in Section 2.2 along with related analytical techniques. A summary of the work in this chapter and conclusions are given in Section 2.3.

2.1 Confidentiality using stream ciphers

Encryption is used to provide confidentiality for data known as plaintext \( (P) \), by changing it into an unreadable format known as ciphertext \( (C) \), using an algorithm, a secret key \( (K) \), and some public information called an initialisation vector \( (IV) \) [80, 88]. This transformation is performed using a mathematical operation known as an encryption algorithm. Let the encryption using \( K \) and \( IV \) be denoted \( E_{K,IV} \). The plaintext can be recovered from the ciphertext by using a reverse transformation process known as the decryption algorithm. Let decryption using key \( K \) and \( IV \) be denoted \( D_{K,IV} \).

For symmetric ciphers, decryption is performed using the same secret key and public \( IV \) that were used to perform the encryption. Encryption and decryption
processes in symmetric ciphers can be described as follows:

\[ E_{K,IV}(P) = C, \]

\[ D_{K,IV}(C) = P. \]

Figure 2.1 shows a transmission model for providing confidentiality using symmetric key algorithms. Note that the key and IV used to encrypt the plaintext message must also be used to decrypt the ciphertext. The dashed line represents transmission of data over an insecure channel.

![Figure 2.1: Model for providing confidentiality](image)

The security of a message encrypted using a symmetric cryptographic algorithm relies on the key. This means that, in order to keep the communication secret, the key that is used for both encryption and decryption must also be kept secret.

Symmetric key algorithms that provide confidentiality can be divided into two categories: block ciphers and stream ciphers. Block ciphers divide a message into successive blocks and encrypt each block using the same key. Stream ciphers divide a message into successive characters and encrypt each character under some time-varying function of the key. The research presented in this thesis related to providing confidentiality focuses on stream ciphers.

### 2.1.1 Stream ciphers

Prior to 2000, most proposed stream ciphers used a character size of one bit only. However, modern stream cipher proposals are frequently based on character sizes
which are more than one bit. This group of bits is called a word, and these stream ciphers are referred to as word-based stream ciphers. Common word sizes are 8, 16, 32, and 64 bits. Word-based stream ciphers are intended to achieve high speed and efficiency in software implementation as well as in hardware [72,91].

The most important component of stream ciphers is the keystream generator. The purpose of the keystream generator is to use the secret key $K$ and a public $IV$ as inputs, to produce a pseudorandom binary keystream sequence. The two main phases of this process are the initialisation phase and the keystream generation phase. In the initialisation phase, the key and $IV$ are used to form the initial contents of the internal state components of the keystream generator. Sometimes this initial state is referred to as a session key. In the keystream generation phase, the internal state is updated using a state update function, and then the output function uses some of the internal state values to produce an output bit or word, depending on the design specifications of the keystream generator. Repeated application of the state update and output function results in the production of a keystream sequence.

The keystream is used to encrypt or decrypt the message. This is done by combining the keystream with the plaintext to produce ciphertext, or with the ciphertext to recover plaintext. The most common combining function used in stream ciphers is the bitwise exclusive OR, or modulo two addition (XOR). The XOR operation has several advantages including speed and ease of implementation in both software and hardware. It is an efficient approach, as the same device can be used for both encryption and decryption. The stream ciphers which use this combining function are called binary-additive stream ciphers. The security of the binary-additive stream cipher depends entirely on the keystream generator [88].

There are two classes of stream ciphers, with classification based on the type of state update function: synchronous stream ciphers and self-synchronous stream ciphers [72].

**Synchronous stream ciphers**

For synchronous stream ciphers, the sequence produced by the keystream generator is independent of the plaintext and the ciphertext, as shown in Figure 2.2. That is, the keystream is generated from the internal state (which is formed from the secret key and $IV$) and an update function. Both the sender and receiver
must be synchronised in order for the decryption process to recover the plaintext message. That is, if a ciphertext segment is formed by combining plaintext and keystream segment, the same keystream segment must be combined with the ciphertext in order to recover the plaintext.

The advantage of using synchronous stream ciphers is that there is no error propagation during the transmission of the message for this type of symmetric encryption. That is, if a bit/word error occurs (only for bit flip error, not deletion or insertion) in the ciphertext during transmission, it affects only the corresponding bit/word of the plaintext.

The disadvantage of synchronous stream ciphers is that if synchronisation is lost due to ciphertext digits being inserted or deleted during transmission, then decryption fails and can be restored only through additional techniques for regaining synchronisation between the transmitter and the receiver. One approach to resynchronisation is to trial offsets of the keystream. Another approach is to use a new key and $IV$ to generate a new keystream to establish a new link between the transmitter and the receiver. An alternative approach to minimise the overheads of maintaining synchronisations which is commonly applied in communications is to use frame-based communication, where the messages are usually split into a number of frames or packets. These frames are transmitted to the receiver using network protocols such as TCP/IP. The lost or distorted frames during the transmission are discarded. The frame number attached to the encrypted data payload is generally used as the $IV$. The same secret key is used for the entire communication. Although the same key is used for the resynchronisation process should generate a completely different sequence of keystream output. This is a practical method for establishing a new communication link especially for applications where the length of the frame that uses the same key with each different $IV$ is small.

**Self-synchronous stream ciphers**

For self-synchronous stream ciphers, the output sequence of a keystream generator depends on both the internal state, and some of the previously generated ciphertext characters. The ciphertext is fed back to the keystream generator, as shown in Figure 2.3. That is, the self-synchronous stream cipher encryption/decryption function can be described as a function of the secret key, public $IV$ and the feedback of the ciphertext/plaintext.
2.1. Confidentiality using stream ciphers

Figure 2.2: Structure of synchronous stream cipher

The advantage of self-synchronous ciphers is that if the synchronisation is lost due to ciphertext digits being inserted or deleted during transmission, then such a cipher is capable of resynchronising automatically at the receiving side, with only a fixed number of plaintext characters not being recovered. Self-synchronising stream ciphers are preferred in situations which require the transmission of very long streams of encrypted messages without resynchronisation.

The disadvantage of self-synchronous ciphers is that there is a limited error propagation in this type of stream ciphers. Suppose that this cipher depends on \( t \) previous ciphertext characters, then if an error occurs, decryption of up to \( t \) ciphertext characters is incorrect. As reported in [63], there appears to be little demand for self-synchronisation.

Figure 2.3: Structure of self-synchronous stream cipher
2.1.2 Cryptanalysis of stream ciphers

Cryptanalysis is the study of mathematical techniques for attempting to defeat cryptographic techniques and, more generally, information security services [88]. It is used by both algorithm designers to evaluate the security of their ciphers, and by attackers attempting to break the ciphers. Direct way to break a cipher with a $k$-bit key is by brute force, that is, by trying a maximum of $2^k$ possible values in order to find the correct key. This approach is also called exhaustive key search. Designers make such an attack infeasible by choosing a large $k$, so that the time it takes to recover the correct key is greater than the time for which the information must remain confidential.

Stream cipher cryptanalysis is usually based on Kerckhoffs’s assumption; it is commonly assumed that the attacker knows the structural details of the cipher used, but not the secret key [66]. Also, it is assumed the attacker has access to the ciphertext, since the transmission channel is insecure. Attacks making use of ciphertext can be classified into the following:

- **Ciphertext-only attack**. (or known-ciphertext) An attacker possesses a set of ciphertexts produced by a stream cipher using the same key with different IVs. If a stream cipher can be attacked using a ciphertext-only attack, this means that the stream cipher’s keystream generator is very weak.

- **Chosen ciphertext attack**. This attack allows the attacker to select the ciphertext they desire to be decrypted. Some attacks rely on the ciphertext being in a certain format.

- **Adaptive chosen ciphertext attack**. This attack allows the attacker to select which ciphertexts are to be decrypted and modify some ciphertext choices based on previously decrypted plaintext. This model is not relevant for synchronous stream ciphers, since the keystream generated is not ciphertext-dependent, but can be applied to self-synchronous stream ciphers.

A cipher which resists ciphertext-only attacks may be susceptible to attacks when some corresponding plaintext is known. In some cases, an attacker may have access to additional information, such as knowledge of some plaintext corresponding to known ciphertext. Under these conditions, three attack types can be considered as follows:
• **Known plaintext attack.** The attacker knows some plaintext-ciphertext pairs. For binary additive stream ciphers, the attacker can then gain access to the same amount of keystream simply by XORing the plaintext and its corresponding ciphertext.

• **Chosen plaintext attack.** The attacker chooses the plaintext she or he wants encrypted and obtains the corresponding ciphertext. For binary-additive synchronous stream ciphers, the outcome of known plaintext and chosen plaintext attacks is the same: a segment of the keystream is revealed. However, if the keystream is plaintext-dependent, as is the case of self-synchronous stream cipher, a chosen-plaintext attack may be an effective approach.

• **Adaptive chosen plaintext attack.** This attack allows the attacker to select which plaintexts are to be encrypted and modify some plaintext choices based on previously obtained ciphertext.

**Attack measurement**

The success and power of the attack can be measured using three parameters as follows:

- **Data:** The data complexity measures the amount of data (plaintext or ciphertext) required to execute the attack.

- **Time:** The time complexity represents the number of encryption/decryption operations needed to perform the attack.

- **Memory:** The memory complexity indicates the amount of memory needed to store all data during the attack.

The parameters of successful attacks are used to compare different attacks on the same cipher, or the level of security provided by different ciphers.

**Main goals of attacks**

There are three main goals of attacks on stream ciphers:

- **Key recovery:** Key recovery attacks are also known as master key attacks. This type of attack recovers the secret key. For some attacks, it
may be possible to recover the plaintext without knowledge of any key, for example, if the same keystream is used more than once \[34\]. A key recovery attack can be performed regardless of whether or not the initialisation function of the stream cipher is one-way. A successful key recovery attack means that data from multiple encryption sessions will be rendered insecure. Once the secret key has been recovered, the attacker will initialise the keystream generator with the recovered key and any known IVs, generate the keystreams and recover the plaintexts from the ciphertexts.

- **State recovery**: State recovery attacks are also known as session key attacks. Modern stream cipher proposals use the secret key and the IV as inputs to the initialisation function and produce an initial state. Following this, the state is updated as the keystream is produced. Recovery of the internal state of a keystream generator means that data from that encryption session would be rendered insecure since the attacker who knows the internal state of the cipher would be able to generate keystream to recover the plaintext message from that point on. Encrypted data from other sessions, assuming that a different IV was used, may still be secure.

- **Distinguishing attacks**: Distinguishing attacks are a form of attack that allows an attacker to determine if a keystream has been generated using a particular stream cipher. Since the keystream output from a stream cipher is supposed to look random, a successful distinguishing attack on a stream cipher implies that there exists some relationship between the keystream bits that can be used to identify the cipher. Although distinguishing attacks do not directly result in the recovery of either the key or internal state, or reveal the plaintext, they indicate a potential weakness. There is also a possibility that the observation may later be extended to a key or state recovery attacks. In order to curb distinguishing attacks, stream cipher designers often impose limits on how much keystream can be generated using a single key-IV pair before rekeying is required.

### 2.1.3 Cryptanalytic techniques for stream ciphers

This section describes existing tools or techniques used to achieve any of the main goals of the attacks, or to evaluate proposed designs against any weakness that can be exploited by an attacker to break the ciphers. The most successful
techniques used to identify weaknesses in stream cipher proposals will be outlined in the following sections. An attack is considered successful if it is faster than exhaustive key search.

**Time-Memory-Data trade off attacks**

This attack approach on stream ciphers was developed independently by Babbage [7] and Golić [47] in 1995 and 1997 respectively. The parameters used in this attack are time $T$, memory $M$, internal state space of the cipher $S$ and amount of data $D$. There are two phases in this attack. The first phase is precomputation which involves computing a table of keys or internal states and the corresponding keystream prefixes. In this table, a set of $M$ random keys or internal states $\{x_i\}_{i=1}^{M}$ are chosen, and for each $x_i$, the corresponding keystream $y_i$ is computed. The pairs $(x_i, y_i)$ are sorted into increasing order of the $y_i$. The second phase is the real time attack phase. This involves observing the sequences of keystream bits with length of $D + \log(N) - 1$, where $N$ is the number of possible secret keys (in $T$ attacking time for $M$ memory). Then use a sliding window to produce all $D$ possible keystream segments $y_1, y_2, \ldots, y_D$ of length $\log(N)$. An attempt to find each of the keystream segments $y_i$ to stored in the table is made. Once a match is found, then the corresponding entry in the table, $x_i$, is the state or key that generates the observed keystream.

The tradeoff in this attack is between the precomputation effort, the length of keystream, and the time spent in the real time phase. The length of the keystream required can be reduced by increasing the amount of precomputation performed. However, this increases the memory required to store the calculated keystream prefixes. The basic tradeoffs satisfy the following equations $T \times M = S$, $P = M$ and $D = T$, where $P$ is the precomputation time to produce the table.

A reduction of the memory requirement of this attack was developed by Biryukov and Shamir in 2000 [16], by using the idea of Hellman’s time-memory tradeoff attack on block cipher [61]. The more general TMD tradeoff equations obtained are: $T \times M^2 \times D^2 = S^2$, $P = S \div D$ and $D^2 \leq T$ [16]. If the internal state of the stream ciphers is small, then the collision of the states can be exploited to recover the secret state.

To avoid time-memory-data tradeoff attacks, the state size of the cipher must be at least twice the key size. Also $IV$ of length at least equal to the key size is needed [62].
Divide and conquer attacks

A divide and conquer attack requires knowledge of the structure of the keystream generator, and applies particularly to keystream generators with multiple components. Originally, it applied to nonlinear combiner generators based on multiple regularly clocked LFSRs. Divide and conquer attacks on stream ciphers work on each component of the keystream generator separately and sequentially solve for individual subkeys and/or the initial states. The attacks start by finding and attacking the most vulnerable component first, then the cipher can be reduced, and the reduced cipher further attacked until the entire state is deduced.

An example of the divide and conquer method is the attack on the nonlinear combiner generator performed by Siegenthaler [90]. Suppose that there is a nonlinear combiner generator consisting of \( n \) different LFSRs, and that these LFSRs have lengths \( L_1, L_2, \ldots, L_n \). Then the total number of different possible initial values of these LFSRs is \( \prod_{i=1}^{n}(2^{L_i} - 1) \). However, if there is some weakness in the generation process so that the properties of some of the individual component registers leak into the observed keystream, then one can potentially break the keystream generator one component at a time. Then the total number of different possible initial values of these LFSRs that must be trialled is reduced using a divide and conquer attack from \( \prod_{i=1}^{n}(2^{L_i} - 1) \) to \( \sum_{i=1}^{n}(2^{L_i} - 1) \).

Divide and conquer is a strategy for attacking that is often combined with other methods, such as correlation attacks or algebraic attacks. Resistance to divide and conquer attacks can be provided by linking the components rather than combining autonomous units, for example, using mutual feedback in registers [20], and in LFSRs using mutual clocking [48]. Increasing the size of components such that the size of any component is greater than the total key size also ensures that standard divide and conquer attacks will be worse than exhaustive key search. However, increasing the size of components requires a secure initialisation process to expand the key and IV in order to form the initial state.

Correlation attacks

Correlation attacks on stream ciphers are based on statistical dependencies that exist between the observed keystream sequence and the sequence generated by individual component parts of the keystream generator [46, 90]. Generally, the correlation between any two binary sequences is a measure of the extent to which
they approximate each other. For sequences of the same length, the usual measure of correlation is based on Hamming distance between the two sequences. For sequences of different lengths, the measure of correlation is based on the operations required to transform one sequence into another and the usual distance measure is the Levenshtein distance [49, 68].

This attack method was first applied to a nonlinear combiner generator by Siegenthaler in 1984 [89, 90], and can be considered as a type of divide and conquer attack. Correlation between the generated keystream sequence and the sequences produced by the underlying internal state of a component LFSR can be used to identify internal state values. This can be done by trying all the initial states of the first LFSR and, for each initial state, producing its LFSR sequence. Then, for each generated LFSR sequence, calculate the correlation between the entire LFSR output sequence and the keystream output using a measure of distance. If the correlation between any sequence generated by initial state and the keystream sequence is high, preserve that initial state as a possible candidate. Then the remaining LFSRs are analysed by the same method. If the key is the initial state of the components, this approach can be used directly for key recovery.

To avoid correlation attacks, any combining functions should be carefully chosen such that they are highly nonlinear and have a high order of correlation immunity [46].

**Fast correlation attacks**

If a correlation attack outperforms exhaustive search over the initial contents of the individual shift register, then it is called a fast correlation attack [50]. This attack approach was developed by Meier and Staffelbach in 1989 [71], as an attack on the nonlinear combiner generator. The basic idea of it is to view the keystream sequence as a noisy version of the underlying LFSR sequence, where the noise is additive to the keystream sequence and independent of the underlying LFSR sequence. Techniques to remove the noise and recover the underlying LFSR sequences are applied.

To perform error correction, parity checks are used to determine which keystream bits are, with high probability, the same as the corresponding bits in the underlying LFSR sequence. A parity check is any linear relationship satisfied by a sequence of the underlying LFSR. The attack aim is to obtain sufficiently many
low weight parity check polynomials of low degree. The parity check polynomials
are polynomial multiples $h(x)$, of the LFSR feedback polynomial $f(x)$, such that
$h(0) = 1$. A simple method to obtain parity check polynomials is by repeated
squaring of the LFSR feedback polynomial, if the weight of $f(x)$ is low \[71\]. An-
other method to generate more parity checks is by applying a polynomial residue
method \[50\].

The nonlinear filter generator is a keystream generator based on a single
LFSR. Hence, for a nonlinear filter generator, the basic correlation attack is
not faster than exhaustive search over the LFSR states. Mihaljević and Golić
proposed an alternative method to reconstruct the initial state of the under-
lying LFSR by a fast iterative algorithm using a probabilistic error correction
method \[73\]. They viewed the keystream as noisy version of a linear trans-
form of underlying LFSR sequence. The application of this algorithm consists
of three main stages; precomputation stage, iterative error correction stage and
reconstruction the initial state of LFSR stage \[50\]. In the precomputation stage,
correlation coefficients between the nonlinear filter function $f(x)$ and all linear
functions are obtained by computing the Walsh transform of function $f(x)$. In
an iterative error correction stage, the keystream sequence is input to this stage.
In each iteration, the parity check values for each keystream bit are calculated,
then the error probabilities for each keystream bit are computed and bit-by-bit
error correction using decision rule is performed. This process is continued until
all parity checks satisfied, or a maximum number of rounds have been reached,
which is set before the running of the attack. The output of each round is a
modified keystream sequence based on the decision rule used. At the end of
each round, a sliding window of $L$ successive bits, where $L$ is the length of the
underlying LFSR, from the modified output keystream is loaded into LFSR and
the whole sequence is reconstructed. Then every sequence is tested for correla-
tion to the observed keystream sequence. If the computed correlation coefficient
estimate is significantly different from zero, then the attack stops as successful,
otherwise the attack proceeds to the next round. In the last stage, the initial
state of the LFSR is recovered from the first $L$ consecutive bits of the modified
sequence of the keystream.

To avoid basic and fast correlation attacks, combining functions must be
selected carefully, so that they have a high order of correlation immunity and are
not correlated to a single input. For keystream generators where components
are LFSRs, the LFSR feedback polynomials must be selected carefully, such that they are not low weight functions, or functions with low weight polynomial multiples \[71\].

**Algebraic attacks**

Algebraic attacks were first applied to block ciphers in 2002 \[31\] and then to stream ciphers in 2003 \[30\]. On stream ciphers, the attacks involve two main stages \[29\]. In the first stage, the analyst establishes multivariate relations of sufficiently low degree between either the key or the internal state values and the output keystream. In the second stage, the system of equations is solved to recover the key or internal state values. The feasibility of the algebraic attack mainly depends on the complexity of solving the system of equations, which can be determined from the maximum degree of the equations, and on the number of equations.

For ease of analysis, algebraic attacks are often performed by describing the cipher state and keystream in terms of bit values, regardless of whether the cipher is bit-based or word-based. The cipher functions, at the bit level, are described in terms of the XOR and AND bit operations (addition and multiplication over GF(2) respectively).

The relationship between either key bits or internal state bits and the output bits can be obtained directly or indirectly for the nonlinear components. Direct relationships are obtained by building mathematical models of the nonlinear components, where the output bits are presented as functions of the input bits. Indirect relationships are obtained by considering the input and output bits of the nonlinear components together and finding valid relationships among them. In either case, the output of first stage is a system of multivariate equations. The resulting system can be post-processed to reduce the maximum degree of the equations in order to increase the efficiency of finding a solution. Reductions in degree of equations can be achieved using techniques such as low degree multiples \[28\].

Once the system of equations is prepared, an attempt is made to find a solution. Several methods exist to solve polynomial equations over GF(2). Linearisation is a method which involves treating each unique product of variables in equations as a single variable. The introduction of these new variables enables the equation system to be translated to a linear system with a larger set
Chapter 2. Background

of variables [30]. More equations and keystream would be required to solve the linearised system. Gröbner basis methods do not require linearising variables, but the time complexity of these methods is generally hard to determine. Another method that has been specially designed for algebraic attacks is eXtended Linearisation (XL) [26]. The XL method will be explained in more detail in Chapter 4.

Other techniques can be combined with computing the solution to a system of equations. Guessing some key or state bits that occur in many high degree terms in the equations may lead to more efficient attacks. Reducing the degrees of equations using approximations can lead to probabilistic attacks [6].

To resist algebraic attacks, the state update functions should be highly non-linear. Also, properties such as high algebraic immunity should be satisfied for nonlinear components within the cipher [27,28]. Algebraic attack techniques are applied to several stream ciphers in Chapter 4 to evaluate their security against these type of attacks on the mechanisms for providing confidentiality.

2.2 Integrity and Message Authentication Codes (MACs)

A Message Authentication Code (MAC) is a symmetric cryptographic primitive for providing message integrity protection [88]. This mechanism can be used to detect any modifications to the message occurring during transmission via a communication channel. Figure 2.4 shows a model for providing integrity using symmetric key algorithms. The dashed line in the Figure represents transmission of data over an insecure channel. The security of message authentication using a MAC function depends on the security of the key, and also on the properties of the function used. A block cipher using certain modes of operation such as Integrity Aware Parallelizable Mode (IAPM) [65] or Offset CodeBook Mode (OCB) [83] can be used to form a MAC tag for any input messages. Keyed hash function also can be used to generate a MAC tag for the input message [9]. The inputs to the message authentication functions are a message of arbitrary length, a secret key $K$, and a public $IV$. The output of these functions is a MAC tag with fixed length. The common lengths of MAC tag are 32, 64 and 128 bits.

Integrity verification can be performed using symmetric ciphers. MACs based on block ciphers using certain modes of operation have been suggested [51,65].
2.2. Integrity and Message Authentication Codes (MACs)

Suppose a MAC tag $\tau$ is generated for the message $M$ as shown in Figure 2.4. Note that this message could be ciphertext or plaintext. Then $\tau$ is appended to $M$ at the sending site and the augmented message $(M, \tau)$ is transmitted over an insecure channel. At the receiving side, a message $M'$ and MAC tag $\tau'$ are received. A MAC tag $\tau'$ is calculated for the received message $M'$ using the same MAC function, the same secret key and the same non-secret IV. Then $\tau$ and $\tau'$ are compared. If they are different, then it is considered that the received message has been modified during the transmission. Otherwise, it is assumed that the integrity of the received message is maintained.

The use of the secret key means that even if an attacker has intercepted the message and knows the MAC algorithm that was used, they should be unable to make changes to the message and calculate the corresponding MAC tag for this modified message without knowledge of the key. MACs therefore provide assurance of data integrity and some degree of data-origin protection. MACs based on symmetric cryptology techniques however, do not provide non-repudiation protection. That is, having verified the MAC tag of a message, we cannot be assured that the claimed sender actually sent the message. To provide this non-
repudiation, digital signatures \cite{82} are required. These make use of asymmetric cryptographic techniques which are beyond the scope of this thesis.

Properties of MAC functions

A secure MAC function should have the following properties:

- **One-way function**: For a given message $M$ and for a known key and $IV$, it is easy to calculate a MAC tag $\tau$; but if you know the MAC tag $\tau$, key and $IV$, then it is difficult to obtain a message $M$ corresponding to that MAC tag $\tau$ \cite{72}.

- **Collision resistance**: For a given message-MAC pair $(M, \tau)$, it is computationally infeasible to find a new message-MAC pair $(M', \tau')$, such that $M \neq M'$ but $\tau = \tau'$ \cite{72}.

- **Forgery resistance**: For a given public information such as sets of input messages and the corresponding MACs formed using the secret key (and possibly other public information such as $IV$), it is computationally infeasible to find a valid MAC tag $\tau$ for any new message $M$, without knowing the secret key \cite{56,72}.

- **Uniform distribution**: For a given MAC algorithm, the probability of obtaining any particular value of MAC tag for a message using either the same key with different $IV$s, or different keys with the same $IV$, should be equally likely \cite{53}.

2.2.2 Statistical analysis

Statistical tests can be used to check whether the distribution of the MAC tags generated from a particular MAC function is uniform. That is, the probability of obtaining any particular value of MAC tag for any input message is equally likely over the set of all possible key - $IV$ pairs. This can be tested using a test known as the repetition test \cite{53}.

**Repetition test**

This test relates the actual number of different patterns that appear in a set of b-tuples. It is based on the birthday paradox that for a set of randomly generated b-tuples, there is an approximately 50% chance of having at least two
b-tuples match after the generation of a set of size $2^{b/2}$ \[^{102}\]. In order to apply the repetition test for uniformity, it is necessary to generate a sample of sufficient size $S$ such that a limited number of repetitions is expected with a high probability, under the hypothesis that the sample was selected at random from a uniform distribution of b-tuples. Under this hypothesis, the number of repetitions in a sample of $S$ b-tuples follows a Poisson distribution \[^{53}\] and the mean $\lambda$ for this distribution can be calculated using the following equation:

$$
\lambda = S - N (1 - e^{-S/N})
$$

where $N = 2^b$. If we choose $S = 2^{b/2+3}$, then the value of $\lambda$ is approximately 32, for $b > 20$.

The test is applied as a two-tailed test of significance, where too many repetitions may indicate that the patterns generated are non-random and as well too few repetitions may also indicate that patterns are non-random. As indicated above, the null hypothesis for the test is that the number of repetitions follows a Poisson distribution with mean $\lambda$. For MAC applications, the test is performed by accumulating many generated MAC tags by fixing one input and varying the other inputs to the MAC generation algorithms. For example, collecting many MAC tags for a message with different keys and IVs used for the generation, or collecting many MAC tags for different messages using the same key and IV for the same MAC generation algorithm. The process for running the repetition test is as follows:

1. Generate a sample $S = 2^{b/2+3}$ of b-tuples.
2. Sort $S$ and calculate the number of repetition.
3. Compare step 2, the number of repetitions from the test with the expected number $\lambda$ to derive the test statistic for this test.
4. Compare the test statistic to the relevant statistical tables to find the probability of obtaining the observed data under the assumption of the null hypothesis. (This probability is known as the p-value for the test.)

To interpret the p-value for each test, we follow the standard practice for statistical tests, which is to consider that low p-value (typically, p-value below 0.05) provide evidence against the null hypothesis, with lower p-values providing stronger evidence against this assumption.
The amount of data required to run this test, for MAC tag sizes of 32, 64 and 128 bits are given in Table 2.1

Table 2.1: Minimum data required to run repetition test $S$

<table>
<thead>
<tr>
<th>MAC size</th>
<th>Number of tags</th>
<th>Number of bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>32 bit</td>
<td>$2^{19}$</td>
<td>$2^{21}$</td>
</tr>
<tr>
<td>64 bit</td>
<td>$2^{35}$</td>
<td>$2^{38}$</td>
</tr>
<tr>
<td>128 bit</td>
<td>$2^{67}$</td>
<td>$2^{71}$</td>
</tr>
</tbody>
</table>

From Table 2.1 it can be observed that it is possible to examine MAC sizes of length 32 and 64 for uniformity by applying the repetition test while it is not feasible to do this for a MAC size of 128 bits due to the very large sample size required. In Sections 3.3.2 and 3.3.7, we apply this test to the MAC tags for the algorithms 128-EIA3 and UIA2 respectively which both have a MAC length of 32 bits. We used the repetition test algorithm from the crypt-X package [53] to carry out steps 2-4 of these tests.

2.2.3 Cryptanalysis of MACs

For MAC cryptanalysis, the attacker is assumed to have complete knowledge of the MAC algorithm and the format of the message. As well, it is assumed that the attacker has knowledge of the MAC tag and the corresponding transmitted message which is ciphertext in case of AE. The attacker can mount either forgery attacks or key recovery attacks. Resistance to MAC attacks is measured using the smallest number of operations with regards to two variables: the size of the key $K$ ($k$ bits), and the size of the MAC tag ($d$ bits).

Attack models on MACs

There are three attack models against which MACs have to provide resistance. These are the known-message attack, the chosen-message attack and the adaptive chosen-message attack [45].

- **Known-message attack**: The attacker looks at a sequence of messages $m_1, m_2, \cdots, m_n$ and the corresponding authentication tags communicated between the legitimate parties in a communication channel. The attacker
might observe these messages by intercepting the communication channel in a manner uninfluenced by the communicating parties.

- **Chosen-message attack:** The attacker chooses a sequence of messages and obtains the corresponding tags from the party possessing the secret key \( k \) to the MAC function. Chosen messages are also referred to as queries and the corresponding tags as answers. The attacker then attempts to produce a valid MAC tag for that chosen message.

- **Adaptive-chosen message attack:** This is a chosen message attack in which the attacker can choose which message to request MAC tags for, based on the previously seen messages and their tags.

**Attack goals on MACs**

When considering the effort involved for an attacker to produce a valid MAC, we compare with the effort required for naive attacks such as brute force attacks on the key and birthday attacks on the MAC. This provides bounds for comparisons with the computational effort required for other potential attacks.

A MAC algorithm with security parameters \((k, d)\) should provide resistance to the following MAC attacks \([9, 33, 72]\):

- **Brute force attacks on the key:** An attacker wishing to provide a valid MAC for an altered message could try all possible keys to determine the key used to construct a MAC tag for a given message, and then use that same key to generate a MAC tag for the altered message. This approach requires at most \(2^k\) trials to determine the key used.

- **Birthday attacks:** Consider a particular MAC algorithm with a secret key as a function that maps input messages of any length to MAC tags of length \(d\). Using a brute force approach, and calculating the MAC tag for many different messages, the expected number of MAC tag calculations necessary in order to find two distinct messages with the same MAC tag is \(2^{d/2}\) \([102]\).

- **Collision attacks:** A collision with respect to a MAC means that there are two distinct messages, \(M\) and \(M'\) that, for a given MAC algorithm and secret key \(K\), and IV both produce the same MAC tag. That is, \(MAC_{K, IV}(M) = MAC_{K, IV}(M')\). In considering security against forgery
attacks, we consider the possibility of collisions of the MAC tag itself, as well as collisions at points corresponding to phases in the generation of the MAC tag.

- **Forgery attacks:** An attacker who does not know the secret key but knows other information such as sets of input messages and the corresponding MACs formed using the secret key (and possibly other public information such as IV) should not be able to produce a valid MAC tag for any other messages with any probability better than guessing \[56, 72\]. There are two types of forgery attacks, selective forgeries and existential forgeries:
  
  - Selective forgery: An attacker is able to produce a new message-MAC pair of his or her own choosing.
  
  - Existential forgery: An attacker is able to produce a new message-MAC pair, but has no control over the contents of that message.

### 2.3 Summary and conclusion

This chapter reviewed the current literature that is relevant to the research topic examined in this thesis. It is divided into two main sections reflecting the two major security services: confidentiality and integrity, respectively, and also discusses the different attack methods that can be applied to stream ciphers.

Firstly, the basic method for providing confidentiality with stream ciphers was examined. Then the classifications of the stream ciphers, and the review of the attack scenarios and the attack methods for cryptanalysis of stream ciphers were given. Attacks on stream ciphers are then described. Note that Chapter 4 of this thesis will discuss the application of algebraic attacks to two stream ciphers.

Secondly, the basic method for providing assurances of integrity with MAC algorithms was examined. Properties of MAC algorithms and type of attacks on MACs are given in this section. Finally, analysis of MACs is explained in this section. Chapters 5 and 6 of this thesis describe two different models of message accumulation processes to generate MAC tag using stream ciphers, and analyse the security of these models against forgery attacks.
Chapter 3

Authenticated encryption using stream ciphers

Methods for providing AE using stream ciphers are discussed in this chapter. We present three methods for classifying stream ciphers which claim to provide AE, and use these classification methods to identify those ciphers with structural similarities. This is useful in determining analysis techniques which can be applied to particular ciphers, to determine the security of either confidentiality or integrity components.

The first method of classification is described by Bellare and Namprempre \cite{11,12} and is based on the order in which the encryption and authentication occur. The second method of classification is based on the way the input message is injected into the internal states of the cipher to generate a MAC tag. The third method is based on whether the sequences that are used for confidentiality and integrity are generated using a single key and IV, or each of the two sequences uses distinct keys and IVs. The second and the third methods of classifications are outcomes of this research. We describe several existing AE proposals, giving descriptions of the keystream generators used to produce the keystream to provide confidentiality, and the MAC tag generation process used to provide the integrity assurance for the input message. Then, we classify these proposals according to these three methods.

The security properties for a particular authenticated encryption scheme vary depending on the order in which authentication and encryption are performed,
assuming that the underlying mechanisms for providing these services are themselves secure. However, for authenticated encryption using stream ciphers, the security of the authentication mechanism may depend on the way in which the input message is injected into the internal state. In some cases forgeries are possible, violating the assumptions of Bellare and Namprempre [11] and thus the expected security properties would not hold. Identifying whether the authenticated encryption scheme makes use of one key-\(IV\) pair or two is also important. If only one key-\(IV\) pair is used to provide authenticated encryption, an attack which compromises either the confidentiality or the integrity component also results in compromise of the other component. Where two different key-\(IV\) pairs are used, compromise of one security service may not compromise the other. This may be a more robust, if less efficient, approach.

This chapter is organised as follows. Authenticated encryption using stream ciphers is presented in Section 3.1. Classifications for AE are given in Section 3.2. Section 3.3 contains descriptions of specific AE stream ciphers and the classification for each cipher based on the three methods for providing AE. A summary of the work in this chapter and conclusions are given in Section 3.4.

### 3.1 Authenticated encryption using stream ciphers

Authenticated encryption proposals using stream ciphers combine the mechanisms for providing both confidentiality and integrity protection into a single cryptographic primitive [11][13][17]. Figure 3.1 shows a model for providing authenticated encryption using symmetric encryption. In this thesis, the symmetric cipher examined are stream cipher algorithms. The process is as follows: a plaintext \(P\) is encrypted (for confidentiality) to form a ciphertext \(C\), and the MAC tag \(\tau\) of the message is also calculated (integrity). This is performed using an authenticated encryption algorithm, which takes as input the plaintext message, the secret key \(K\) and optionally, some public information \(IV\). The outputs of the AE encryption are the ciphertext \(C\) and MAC tag \(\tau\). These are sent across the insecure channel to the receiver side.

Upon receiving the ciphertext \(C\) and the MAC tag \(\tau\), the receiver uses the authenticated decryption algorithm to recover the message and checks whether it has been modified. This decryption algorithm takes as input the same key
3.1. Authenticated encryption using stream ciphers

$K$ and $IV$ used in the authenticated encryption algorithm and the received ciphertext $C'$. The output of the authenticated decryption algorithm is the recovered plaintext $P'$ and the MAC tag $\tau'$. The receiver checks if the value of $\tau'$ is equal to $\tau$. If the values of the two MAC tags are not the same, the receiver will know that the message has been modified. That is $C' \neq C$. Note that Figure 3.1 is similar to Figure 2.4, although in this case the transmitted message is the ciphertext $C$ (the encrypted form of the message).

![Figure 3.1: Model for providing authenticated encryption](image)

Usually the pseudorandom binary sequences that are generated using the keystream generators of stream ciphers are used as keystreams for binary additive stream ciphers to provide confidentiality for plaintext messages. However, they can also be used for integrity applications only. Where authenticated encryption is required, the keystream sequence used for the integrity application may be produced by the same generator as the sequence used for the confidentiality application, but a different keystream generator could also be used. Figure 3.2(a) shows a single keystream generator using a single ($K$, $IV$) pair to produce two different binary sequences, $z_t$ and $y_t$, used for confidentiality and integrity applications, respectively. If one generator is used to produce both sequences, then we assume that the two sequences are distinct. Examples of such algorithms include SSS [55], Sfinks [18], Grain-128a [1], SOBER-128 [54] and NLSv2 [56]. Figure 3.2(b) shows an alternative case where the sequences $z_t$ and $y_t$ are generated by separate keystream generators, with distinct keys and IVs for each: $K_C$, $IV_C$, $K_I$ and $IV_I$, respectively. Examples of such algorithms include SNOW 3G [93], used in UEA2 and UIA2 [92], and ZUC [38], used in 128-EEA3 and 128-EIA3 [40].
Chapters 5 and 6 of this thesis will focus on the integrity component of the designs; the part inside the dashed lines in Figure 3.2 (for both (a) and (b)). In this Figure, we use dashed lines from the output ciphertext to the integrity component of the algorithm to indicate that either plaintext or ciphertext could be accumulated into the integrity component.

![Diagram](image)

Figure 3.2: General structure for AE algorithms

Encryption and authentication using an AE cipher occurs in three phases as follows [72]:

- **Initialisation phase**: The secret key $K$ and public $IV$ are used to initialise the confidentiality and integrity components using the defined initialisation function.

- **Keystream generation and message accumulation phase**: The plaintext message is encrypted and ciphertext is generated. Simultaneously, the message (either the plaintext or ciphertext) is used to update the integrity
component in a message dependent way. The message injection may be direct or indirect.

- **MAC finalisation phase:** Once the plaintext has been encrypted, the integrity component undergoes some additional post-encryption operations. At the end of this phase, the MAC tag of the message (either the plaintext or ciphertext) is produced.

The AE decryption and MAC verification process occurs in four phases as follows [72]:

- **Initialisation phase:** The secret key $K$ and public $IV$ are used to initialise the confidentiality and integrity components using the initialisation function.

- **Keystream generation and message accumulation phase:** The ciphertext message is decrypted and the plaintext is recovered. Simultaneously, the message (either the plaintext or ciphertext) is used to update the integrity component.

- **MAC finalisation phase:** The integrity component undergoes some additional post-decryption operations, then the MAC tag of the message is produced.

- **MAC verification phase:** The generated MAC tag is compared with the received MAC tag. If the two tags are identical then it is assumed that the received message has not been modified during the transmission. Otherwise the received message is rejected as it has been modified during transmission over the insecure channel.

### 3.2 Classification of methods for providing AE

As discussed previously, the inputs to the encryption process using the AE stream cipher are plaintext $P$, secret key $K$ and public $IV$. The outputs are ciphertext $C$ and the MAC tag $\tau$. There are currently no design guidelines to design secure methods for AE, all of the existing design proposals are ad hoc designs. In this section we present three function methods of classification for providing AE using stream ciphers. The first method of classification is based on the order
in which the two processes involved in AE, encryption and authentication, are performed. This method was described by Bellare and Namprempre [11, 12]. The second method of classification is based on the way the input message is injected into the internal states of the cipher in order to generate a MAC tag, either directly or indirectly. Various methods for direct injection are described in [76, 77]. We generalise these methods and form a general model in Chapter 5, while in Chapter 6 we discuss various options associated with indirect message injection. The third method is based on the number of key-IV pairs used. Some AE ciphers are used single key-IV pair for both confidentiality and integrity mechanisms. However, two different key-IV pairs are used by some of AE ciphers for providing confidentiality and integrity mechanisms.

3.2.1 The ordering of encryption and authentication

Bellare and Namprempre [11, 12] investigated the security properties of three methods for providing AE. These methods refer to the order in which encryption and MAC generation occur. These are Encrypt-and-MAC (E&M), MAC-then-Encrypt (MTE) and Encrypt-then-MAC (ETM). These methods can be described as follows:

- **Encrypt-and-MAC (E&M):** The sender first forms a MAC tag $\tau$ of the plaintext $P$ using a MAC algorithm $MAC_{K,IV}(P)$, encrypts the plaintext using a symmetric encryption algorithm $E_{K,IV}(P)$ to obtain the ciphertext $C'$, and then appends the MAC tag of the plaintext to the ciphertext $C'$ to form an augmented message $C$ that will be transmitted. At the receiving end, the receiver first parses the received message $C$, to obtain $C'$ and $\tau$. Then the ciphertext $C'$ is decrypted using a decryption algorithm $D_{K,IV}(C')$. Following this, a MAC tag $\tau'$ is generated using the decrypted ciphertext. Finally $\tau'$ is compared to the MAC tag $\tau$ that was received (as a part of $C$). The processes required at the sending and receiving ends of the transmission for this scheme are shown in Algorithm 3.1.

- **MAC-then-Encrypt (MTE):** The sender first appends a MAC tag $\tau$ of the plaintext $P$ to the message and then encrypts the augmented plaintext-MAC message using a symmetric encryption algorithm $E_{K,IV}(P||\tau)$. At the receiving end, the receiver checks the message by first decrypting the message using a decryption algorithm $D_{K,IV}(C)$ to obtain the plaintext $P$
3.2. Classification of methods for providing AE

and the MAC tag $\tau$. Then the receiver computes the MAC tag $\tau'$ of the received plaintext, and compares it with the MAC tag $\tau$. The processes required sending and receiving ends for this scheme are shown in Algorithm 3.2.

- **Encrypt-then-MAC (ETM):** The sender encrypts the plaintext $P$ to obtain the ciphertext $C$ using a symmetric encryption algorithm $E_{K,IV}(P)$. A MAC tag $\tau$ of the ciphertext $C$ is calculated using a MAC algorithm $MAC_{K,IV}(C)$ and then appended to the message. At the receiving end, the receiver will calculate $\tau'$ and then check the MAC tag. If the verification check passes (which means $\tau = \tau'$) the ciphertext is decrypted to obtain the plaintext using a decryption algorithm $D_{K,IV}(C)$. The processes required sending and receiving ends for this scheme are shown in Algorithm 3.3.

Bellare and Namprempre [11][12] analysed the security of these three generic AE methods in terms of the following security properties:

1. Indistinguishability of encryptions under the chosen plaintext attack.
2. Indistinguishability of encryptions under the chosen ciphertext attack.
3. Non-malleability under the chosen ciphertext attack.
4. Integrity of plaintexts.
Algorithm 3.2: Sending and receiving processes in MAC-then-Encrypt scheme

5. Integrity of ciphertexts.

Indistinguishability of encryption is the inability of an attacker to learn any information about the plaintext or the ciphertext from a challenge ciphertext or plaintext respectively [10]. Non-malleability is the inability of the attacker, given a challenge ciphertext $C$, to produce a different ciphertext $C'$ such that there is no meaningful relationship between the decrypted messages $P$ and $P'$ corresponding to $C$ and $C'$ respectively [10]. Integrity of plaintext refers to the computational infeasibility of producing a ciphertext decrypting to a message that the sender never actually sent, while integrity of ciphertext refers to the computational infeasibility of producing a ciphertext regardless of whether the underlying plaintext is “new” [11,12].

In their analysis, Bellare and Namprempre prove that if the integrity component of an AE algorithm is strongly unforgeable, the ETM scheme provides all five of the security properties listed above. However, the MTE method provides only two security properties: indistinguishability of encryptions under the chosen plaintext attack, and integrity of plaintext. The E&M method provides only one security property, integrity of plaintext. A summary of the three methods is given in Table 3.1. In Chapters 5 and 6 we will consider conditions for the integrity component to meet this assumption of unforgeability.
Algorithm 3.3: Sending and receiving processes in Encrypt-then-MAC scheme

Table 3.1: Summary of the three methods of ordering encryption and authentication

<table>
<thead>
<tr>
<th>Method</th>
<th>Form of output</th>
<th>Security services provides</th>
</tr>
</thead>
<tbody>
<tr>
<td>E&amp;M</td>
<td>$C = E_{k,IV}(P)</td>
<td></td>
</tr>
<tr>
<td>MTE</td>
<td>$C = E_{k,IV}(P)</td>
<td></td>
</tr>
<tr>
<td>ETM</td>
<td>$C = E_{k,IV}(P)</td>
<td></td>
</tr>
</tbody>
</table>

3.2.2 Message injection

Based on our research (see Chapters 5 and 6) we describe a new classification for MAC generation using stream ciphers. This sort of classification is not based on the considerations above but is instead based on the way that the input message is accumulated into the internal states of the cipher to generate the MAC tag. As discussed in Section 3.1, the input message is accumulated during the second phase of the MAC generation process for an AE cipher. The aim of this process is to accumulate specific message dependent inputs into the internal states of the cipher that contribute to providing the integrity service for the input message. Based on the way the input message is injected into the internal state of the cipher, we identify two categories as follows:

- **Direct injection**: The input message is injected directly into the internal states of the cipher. That is, the input message itself is accumulated into
the internal states. The internal states for the integrity component using this method of injection could be the same as the internal states of the confidentiality component, or could be dedicated internal states used for message accumulation only. Figure 3.3 shows a nonlinear filter keystream generator used in this way to generate a MAC tag. Several existing AE proposals using this model will be discussed in Section 3.3. Using this method of injection, an attacker may be able to manipulate the input message during the message accumulation to force a collision for two distinct messages resulting in the MAC tags for both messages being the same. So, the accumulation process for the input message in this case plays a critical role on the security of this method of injection. A security investigation and more details about this way of injection will be given in Chapter 5.

Figure 3.3: MAC generation using the input message directly

- **Indirect injection:** The input message is injected indirectly into the internal states of the cipher. That is, the input message is not loaded directly into the internal states. Instead, it is used as a controller to determine which segments of the keystream are accumulated into the internal states. In this case, the internal states of the integrity component are dedicated states used only for MAC generation. Figure 3.4 shows a model for generating a MAC tag using this method of injection. In this case, an attacker may be able to modify an input message such that a valid MAC tag can be generated for a modified message, unless good initialisation and finalisation phases are selected to avoid these sorts of attacks. So, the options in the initialisation and finalisation phases in this case play a critical role on the
security of this method of injection. The security investigation and more details about this way of injection will be given in Chapter 6.

![MAC generation process](image)

Figure 3.4: MAC generation using indirect message injection

### 3.2.3 Number of key-IV pairs used

During this research, we noted that there are two different approaches that used for the key-IV pair. The first approach is to use a single key-IV pair for both mechanisms confidentiality and integrity. The second approach is to use two different key-IV pairs for confidentiality and integrity mechanisms respectively. Note that there is a possible weakness in using a single key-IV pair for both integrity and confidentiality.

### 3.3 Description of specific AE stream ciphers

This section describes several specific AE stream ciphers. All of the ciphers have been proposed for widespread use (either as eSTREAM submissions or for applications for mobile telephony), and hence warrant careful examination. These include SSS [55], ZUC [38], Sfinks [18], NLSv2 [56], SOBER-128 [54], Grain-128a [1] and SNOW 3G [93]. For each cipher, the description includes the method for keystream generation, MAC tag generation and a summary of previous analysis. We also discuss the classification of the cipher using Bellare and Namprempre [11, 12] method and give our classification based on the way
Chapter 3. Authenticated encryption using stream ciphers

the input message is used to generate a MAC tag. We also perform a statistical examination of the randomness for MAC tags generated using the ciphers: ZUC and SNOW 3G. This is possible as they both generate MAC tags of size only 32 bits. The repetition test discussed in Section 2.2.2. was applied using five random files of messages with same key and using four random files of keys for the same message. Note that these ciphers are used later in chapters 4, 5 and 6 of this thesis.

3.3.1 SSS

The stream cipher Self Synchronous SOBER (SSS) [55] is a member of the SOBER family [86] designed to provide AE of messages. It was submitted to eSTREAM [78], the ECRYPT stream cipher project, as a cipher suitable for software and hardware applications (Profiles I and II). The cipher takes as input a 128-bit key $K$ and optional 128-bit $IV$. The keystream is produced as a series of 16-bit words. The cipher also generates a 128-bit MAC tag.

**Keystream generation**

The SSS keystream generator consists of a 17-stage shift register and a nonlinear filter (NLF) as shown in Figure 3.5. Each register stage $R[0], \ldots, R[16]$, contains a 16-bit word. Thus the total internal state size is 272 bits.

![Figure 3.5: SSS keystream generator and MAC generation](image)
3.3. Description of specific AE stream ciphers

The state update function for the shift register is given as follows:

\[
R[i]^{t+1} = \begin{cases} 
C^t & \text{for } i = 16 \\
R[15]^t + f(C^t) \gg 8 & \text{for } i = 14 \\
f(R[13]^t) & \text{for } i = 12 \\
f(R[12]^t) \gg 8 & \text{for } i = 1 \\
R[i + 1]^t & \text{for } i = 0, 2, \ldots, 11, 13, 15.
\end{cases}
\]

The keystream word \(Z^t\) is obtained as a nonlinear combination of the contents of five stages of the shift register, as shown in Figure 3.5, and given by following equation:

\[
\]

An important component of SSS is the nonlinear function \(f\). The function \(f\) makes use of a key-dependent SBox which has an 8-bit input and a 16-bit output. We consider the 16 bit intermediate word \(X = R[0] + R[16]\) as the concatenation of the two eight bit words, \(X_H\) and \(X_L\). In [55] the role of the SBox in \(f\) is given by

\[
f(X) = \text{SBox}(X_H) \oplus X \quad (3.2)
\]

If we expand the function \(f\) using the expression in Equation 3.2 we obtain an equivalent structure for the NLF, as shown in Figure 3.6.

The ciphertext word \(C^t\) at time \(t\) is formed by XORing the generated keystream word \(Z^t\) with the plaintext word \(P^t\). As SSS is a self-synchronous stream cipher, the ciphertext word \(C^t\) is also fed back to the stages of shift register, as shown in Figure 3.5.

MAC tag generation

The SSS MAC generator consists of a 17-stage LFSR, denoted register \(A\) in Figure 3.5. Each register stage \(a[0], \ldots, a[16]\), contains a 16-bit word. Thus the total internal state size of this component is 272 bits.

Register \(A\) is initialised with keystream words produced by the confidentiality component during the initialisation phase of the cipher. The state update
Chapter 3. Authenticated encryption using stream ciphers

Figure 3.6: Non-Linear Filter (NLF) of SSS

function for the shift register is given as follows:

\[
\begin{align*}
    a[i+1] &= \begin{cases} 
        a[i + 1]^t & \text{for } i = 0, \ldots, 15, \\
        P^t \oplus (\delta \otimes a[0]) \oplus a[4] \oplus a[15] & \text{for } i = 16.
    \end{cases}
\end{align*}
\]

where \( \delta \) is a non-trivial element in \( \text{GF}(2^{16}) \). The MAC generation process begins by preparing the input message by padding with zero bits such that the resulting length in bits is a multiple of 16. Then the message is accumulated into register \( A \). The input message is accumulated using the state update function for register \( A \) described by Equation 3.3. When all of the message has been processed, the finalisation phase is carried out to form the MAC tag. This process is as follows:

- A padding word is formed of value \( b \gg 3 \) (where \( b \) is the number of bits added into the message) and this word is encrypted using a keystream word from the confidentiality component, but this word is not input into register \( A \).

- The content of register \( A \) at time \( t \), where \( t \) is the size of the input message in words, is encrypted word by word using the keystream words, starting with word \( a[16] \). The output ciphertext word at each time is fed back into register \( R \) in the confidentiality component.

- Finally, to generate a MAC tag of length \( d \) bits, we then use the confi-
dentiality component to generate the required number of keystream words, using all-zero plaintext words as input. These generated keystream words form the MAC tag and at the same time are also fed back into register $R$, until the number of words required to form the MAC tag is generated.

**Previous analysis**
The previous attack on SSS was a chosen ciphertext attack [32] on SSS’s confidentiality component. During the initialisation phase, the key of SSS is used with two fixed tables to construct a key dependent SBox that forms part of the nonlinear function $f$. The chosen ciphertext attack aims to recover all entries of the SBox by selecting ciphertexts of particular patterns and obtaining the corresponding plaintext and hence the keystream. The attack forms equations based on the decryption process of the ciphertext. It is necessary to guess 24 bits in these equations to reconstruct the SBox. To verify a guess, the ciphertext is decrypted using the constructed SBox to recover the corresponding plaintext. If the plaintext is correct then the SBox is assumed to be correct. Otherwise the process is repeated for a different 24-bit guess. This attack recovers the SBox in, on average, 10 seconds using 9468 bytes of chosen ciphertext. As stated in the specifications of SSS [55], the SBox is effectively the key, since it is the only key-dependent component in the cipher. Therefore, the chosen ciphertext attack is equivalent to a full key recovery attack. Despite the attack being very successful, chosen ciphertext attacks were not considered to be a valid scenario by the designers of SSS [55]. The argument is that in a self-synchronising cipher, one can always control the state of the cipher by choosing the ciphertext. Therefore, it should be assumed that decrypting altered ciphertexts should not be allowed by an attacker.

We apply an algebraic attack on SSS using an indirect method to obtain equations by considering the NLF as a combiner with memory. Details of this analyses are given in Chapter 4.

**Classifying SSS**
The plaintext is encrypted to form the ciphertext. At the same time, the plaintext message is accumulated in register $A$. Then the final value of register $A$ is encrypted using the keystream words to form the MAC tag. The tag is appended to the ciphertext to form a message that will be sent to other side. If we compare the process of forming the sent message with the above methods of Bellare and Namprempre [11][12], we can classify SSS under the MAC-then-Encrypt (MTE)
scheme. From Table 3.1 we can see that this scheme may not provide integrity of ciphertext security service since the MAC tag is accumulated plaintext. We note that non-malleability of ciphertext is not guaranteed for MAC’s generated using the MTE scheme; however, in the case of SSS, this property is provided by using a self synchronous mechanism to generate the keystream. Thus, any changes in the ciphertext will change the keystream output causing the plaintext to change in a manner which is not controlled by the attacker.

SSS can be classified based on the message injection method, since the input message is loaded into the dedicated internal states of the integrity component, and the input message words are loaded into these internal states. Thus, SSS can be classified under the direct input message injection. A security analysis for the MAC formed using a directly injected input message in SSS is presented in Chapter 5.

For the third classification of SSS, SSS uses single key-IV pair for both confidentiality and integrity mechanisms.

### 3.3.2 ZUC and 128-EIA3

The ZUC stream cipher [38] is designed for use in China as a standard for fourth generation mobile phones. It is a word based stream cipher with a word size of 32-bits. The inputs to the cipher are a key \( K \) and an IV, each of size 128 bits (4 words), and the output is a 32 bit keystream word. Besides generating keystream to be used for encryption, the keystream of ZUC can also be used to generate a 32-bit MAC. The algorithms making use of the ZUC keystreams for encryption and for authentication are referred to as 128-EEA3 and 128-EIA3, respectively [39,40].

**Keystream generation**

The internal state of ZUC [38] consists of a 16-stage LFSR \((S[0], S[1], \ldots, S[15])\) and two internal memory states \(R[1] \) and \(R[2] \), with each register stage containing 31 bits and each memory state containing 32 bits. Thus, the total internal state size is 560 bits. The structure of ZUC can be considered as three layers: the LFSR, a bit reorganisation layer which takes 16 bits from each of 8 different stages of the LFSR to form four 32-bit words, and a nonlinear function \(f\), which uses two 32-bit words from the bit reorganisation layer and the contents of the two memory stages \(R'[1] \) and \(R'[2] \) to generate two new values \(R'^{+1}[1]\) and \(R'^{+1}[2]\). The second and third layer can be considered together as a NonLinear Filter
(NLF) applied to the LFSR. The structure of the ZUC stream cipher is shown in Figure 3.7. The two memory words are combined with a word from the bit reorganisation layer, $X_0$, to generate a word $W$ used only during the initialisation phase to update the shift register. In the keystream generation phase these two memory words are combined with two words from the bit reorganisation layer, $X_0$ and $X_3$ to generate the keystream word $Z$.

The ZUC keystream generator uses a 16-stage shift register $S$, where each stage contains a 31 bit value. The set of possible values of each stage are restricted to $\{1, 2, \ldots, 2^{31} - 1\}$. The state update function for the LFSR differs depending on whether the cipher is in initialisation or keystream generation phase. In the initialisation phase, the update function of the LFSR is not autonomous but uses the output word $W$ as shown in Figure 3.7 (including the dashed line). The update process in the initialisation phase is as follows:

2. $S[16] = (u + (W \gg 1)) \mod (2^{31} - 1)$
3. If $S[16] = 0$, set $S[16] = 2^{31} - 1$
4. $(S[1], S[2], \ldots, S[15], S[16]) \rightarrow (S[0], S[1], \ldots, S[14], S[15])$

In the keystream generation phase, the LFSR is autonomous with state update function as follows:
Chapter 3. Authenticated encryption using stream ciphers


2. If \( S[16] = 0 \), set \( S[16] = 2^{31} - 1 \)

3. \((S[1], S[2], \ldots, S[15], S[16]) \rightarrow (S[0], S[1], \ldots, S[14], S[15])\)

The NLF takes as input four 32-bit words from the LFSR denoted \((X_0, X_1, X_2, X_3)\) as shown in Figure 3.7 and two words from memory states \(R[1]\) and \(R[2]\). There are two layers in the NLF: bit reorganisation and the nonlinear function \(f\).

The bit reorganisation layer extracts 128-bits from the state of the LFSR to form four 32-bit words as follows:

1. \( X_0 = S[15H] || S[14L] \)
2. \( X_1 = S[11L] || S[9H] \)
3. \( X_2 = S[7L] || S[5H] \)
4. \( X_3 = S[2L] || S[0H] \)

where \( S[iH] \) and \( S[iL] \) are the 16 most significant bits and 16 least significant bits of stage \( i \) respectively. The two words \( X_1^t \) and \( X_2^t \) are used by the nonlinear function \( f \) to update the two 32-bit words \( R^t[1] \) and \( R^t[2] \). The other two words \( X_0^t \) and \( X_3^t \) are used by the NLF to generate the output word \( Z \).

The nonlinear function \( f \) uses four words as input: \( X_1^t, X_2^t, R^t[1] \) and \( R^t[2] \). Two output words \( R^{t+1}[1] \) and \( R^{t+1}[2] \) are produced as shown in Figure 3.7. Two intermediate words \( W_1 \) and \( W_2 \) are used in the nonlinear function \( f \) as follows:

1. \( W_1 = R[1] + X_1 \)
2. \( W_2 = R[2] \oplus X_2 \)
3. \( R_1 = SB(L_1(W_{1L} || W_{2H})) \)
4. \( R_2 = SB(L_2(W_{2L} || W_{1H})) \)

where \( SB \) is a \((32 \times 32)\) SBox and \( L_1 \) and \( L_2 \) are linear transformations.

The \((32 \times 32)\) SBox used in ZUC is composed of four \((8 \times 8)\) Sboxes \( SB = (S_0, S_1, S_2, S_3) \), where \( S_0 = S_2 \) and \( S_1 = S_3 \). In order to use this construction the input word to the SBox is divided into four 8 bit subwords. Each of these is input to one of the four Sboxes.
The two Sboxes $S_0$ and $S_1$ have the following properties. The algebraic degree for $S_0$ and $S_1$ are 5 and 7 respectively. The highest probability of differential characteristic for $S_0$ and $S_1$ are $2^{-5}$ and $2^{-6}$ respectively. The highest bias of linear characteristic for $S_0$ and $S_1$ are $2^{-3}$ and $2^{-4}$ respectively. The number of linear independent quadratic equations that can be established by $S_0$ and $S_1$ are 11 and 39 respectively [24, 67].

The linear transformations $L_1$ and $L_2$ act on a 32-bit word $X$ in the following way:

$$L_1(X) = X \oplus (X \ll 2) \oplus (X \ll 10) \oplus (X \ll 18) \oplus (X \ll 24)$$
$$L_2(X) = X \oplus (X \ll 8) \oplus (X \ll 14) \oplus (X \ll 22) \oplus (X \ll 30)$$

The initialisation phase takes as input the 128-bit secret key $k$, a 128-bit IV $v$ and a 240-bit constant value $D$. Consider both $k$ and $v$ as the concatenation of 16 bytes, so $k = k_0||k_1||\ldots||k_{15}$ and $v = v_0||v_1||\ldots||v_{15}$, respectively. Similarly, consider $D$ as the concatenation of 16 15-bit values as $D = d_0||d_1||\ldots||d_{15}$. To begin, the register and memories are loaded as follows:

1. Set $S[i] = k_i||d_i||v_i$ for $0 \leq i \leq 15$
2. Set the memory cells $R[1]$ and $R[2]$ to 0.
3. Following this, 32 iterations of the initialisation state update function are performed.

Once the initialisation phase is complete, one word of keystream is generated at each time $t$, as follows:

$$Z^t = ((X^t_0 \oplus R[1]^t) + R[2]^t) \oplus X^t_3 \quad (3.4)$$

The LFSR state update function during keystream generation is as outlined above. The first word of the keystream generated is discarded and the remaining words are used to encrypt the plaintext or decrypt the ciphertext using 128-EEA3 [39, 40], as shown in Figure 3.8.

**MAC tag generation**

A second ZUC keystream generator using a different key $K_I$ and $IV_I$ is used to generate a 32-bit MAC by using the 128-EIA3 [39, 40] algorithm as shown in Figure 3.8.
Chapter 3. Authenticated encryption using stream ciphers

The 128-EIA3 algorithm uses the ZUC keystream generator to generate keystream words for length two words $y_t$ more than the length of the input message. These words of keystream and the input message (which is the ciphertext) are converted into bits.

During the preparation phase, the input message is padded with a single bit of value 1 at the end of the message, and also register $R$ is initialised with a word of keystream $y_t$.

Then the accumulation phase is started, in which the keystream bits are accumulated into the internal state of the algorithm based on the value of the input message. If the input message bit value is 1, then register $A$ is updated using the contents of register $R$, and if the value of message bit is 0, then there is no change in register $A$, while register $R$ in all clocks is updated from keystream sequence $y_t$.

Then after the whole message is processed, the final contents of register $A$ are XORed with the last word of the keystream sequence $y_t$ which not has been used during the accumulation phase. The output result word is considered as a MAC tag value of the input message. More details of MAC generation using 128-EIA3 are in Section 6.2.1.

**Statistical analysis for MAC tag generated using 128-EIA3**

We applied the repetition test as described in Section 2.2.2 to MAC tags generated from 128-EIA3 algorithm to examine the hypothesis that a uniform distribution of 32-bit tuples is beings generated. Seven different experiments were conducted each forming a sample size of $2^{19}$ MACs. Three different formats for the input data for the formation of a sample of MAC tags were used namely:

1. random message and fixed key,
2. random key and fixed message and
3. random key and random message

The results of the tests are presented in Table 3.2. From observation of this table, the p-value for all the samples support the hypothesis that the MAC tags generated from 128-EIA3 are equally likely.

Table 3.2: Summary of the results for 128-EIA3 algorithm

<table>
<thead>
<tr>
<th>Message</th>
<th>Key</th>
<th>Number of repetitions observed</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random 1</td>
<td>same</td>
<td>38</td>
<td>0.2888</td>
</tr>
<tr>
<td>Random 2</td>
<td>same</td>
<td>27</td>
<td>0.3768</td>
</tr>
<tr>
<td>Random 3</td>
<td>same</td>
<td>24</td>
<td>0.1573</td>
</tr>
<tr>
<td>Random 4</td>
<td>same</td>
<td>35</td>
<td>0.5959</td>
</tr>
<tr>
<td>Same</td>
<td>Random 1</td>
<td>25</td>
<td>0.2159</td>
</tr>
<tr>
<td>Same</td>
<td>Random 2</td>
<td>26</td>
<td>0.2888</td>
</tr>
<tr>
<td>Random 4</td>
<td>Random 3</td>
<td>31</td>
<td>0.8597</td>
</tr>
</tbody>
</table>

Previous analysis
A state convergence in ZUC V1.4 identified by Wu et al. [100] was presented at a rump session of Asiacrypt 2010, based on chosen IV attack. They made the following observation: if two distinct 31-bit values \(a\) and \(a'\), were XORed with the 31-bit value \(b\), then it is possible to find: \((a \oplus b) \mod (2^{31} - 1) = (a' \oplus b) \mod (2^{31} - 1)\). For example, if \(a = 1111111111000000000000000000000\), \(a' = 0000000000011111111111111111111\), \(b = 1111111111111100000000000000000000\), then \((a \oplus b) \mod (2^{31} - 1) = (a' \oplus b) \mod (2^{31} - 1) = 0 = (2^{31} - 1) \mod (2^{31} - 1) = 0\). Based on this observation, it is possible for an attacker to introduce a difference at first word of IV for two different IVs. These differences will each give a different feedback bit at \(S[15]\). Then, these differences may be cancelled out after XORing the respective \(W\) in Figure 3.7 with the \(S[15]\). (Note that Figure 3.7 shows the initialisation process for the two versions (V1.4 and V1.5).) Since the difference was introduced in the first word of the IV which is at state \(S[0]\), state convergence can occur after one iteration of ZUC’s mixing process. The attack estimates that identical keystreams can occur with probability of \(2^{-16}\) and also estimates that the effective key size of ZUC can be reduced to a value in the range of 66 -100 bits, depending on where the difference in IV is located during the initialisation phase.
Chapter 3. Authenticated encryption using stream ciphers

ZUC version 1.5 [38] was introduced to overcome this weakness. The modification suggested by Wu et al. [100] that the addition of \( S[15] \) to \( W \) is addition modulo \( (2^{31} - 1) \) instead of modulo 2 was adopted.

There was also a forgery attack on 128-EIA3 version 1.4 [44]. Given the original message and the associated MAC value, the attacker was able to predict the MAC value of a related message. The two messages are different in length, but the attack assumes that the two messages are using the same key and \( IV \). The attack exploits the sliding property between the keystream sequence used in the accumulation process and final mask. This property relates the length of the message with the word that is used as a final mask. Then, version 1.5 was introduced to overcome this problem also. The final mask in version 1.5 is selected such that it breaks this sliding property between the two sequences. More details about this attack, and our analysis are presented in Chapter 6.

Algebraic analysis were conducted on the ZUC using direct method to establish equations for the NLF [24,67]. We apply a different algebraic attack on the ZUC using an indirect method to obtain equations by considering the NLF as a combiner with memory. Further details of the analysis from [24, 67] together with our indirect analysis is presented in Chapter 4.

Classifications of ZUC for providing AE

The plaintext is first encrypted to get the ciphertext using 128-EEA3 algorithm. At the same time a MAC tag is generated for the ciphertext. Then final MAC tag is appended with the ciphertext to form a message that will be sent to other side. So if we compare the process of forming the sent message with the above methods of Bellare and Namprempre [11,12], we can classify that ZUC is under the Encrypt-then-MAC (ETM) scheme. Based on the assumption that the integrity mechanism is strongly unforgeable, this scheme according to Table 3.1 provides all the five security services that have been mentioned in Section 3.2.1.

For the second classification of ZUC, the input message is not loaded into the internal states of the integrity component. It is only used to control which segment of keystream bits have to be accumulated into the internal states. Thus, ZUC can be classified under the indirect input message injection. A security analysis for MAC generation using indirect injection of the input message in ZUC is discussed in Chapter 6.

There are two key-\( IV \) pairs are used in ZUC stream cipher. One pair is used
3.3. Description of specific AE stream ciphers

for confidentiality mechanism and the other pair is used for integrity mechanism.

3.3.3 Sfinks

Sfinks [18] stream cipher is one of the proposals submitted to the eSTREAM project [78] that claims to provide AE. It is a hardware-oriented stream cipher (Profile II in ECRYPT project) with key and IV of 80 bits each. The cipher also generates a 64-bit MAC tag.

Keystream generation

Sfinks keystream generator uses a single binary LFSR $S$ of size 256 stages, and a nonlinear inversion function $INV$, as shown in Figure 3.9. The $INV$ function may considered as a $16 \times 16$ bit SBox, with input bits $(x[16], \ldots, x[1])$ and output bits $(y[15], \ldots, y[0])$. Each register stage $S[0], \ldots, S[255]$, contains a single bit. There are also two registers ($R$ and $A$) of size 64 bits each, which are used to generate a MAC tag, for message authentication. Thus the total internal state size is 384 bits.

![Figure 3.9: Sfinks keystream generator and MAC generation](image)

There are two state update functions for the LFSR; one during the initialisation phase and the second one during the keystream generation phase. During the initialisation phase, the secret key is loaded into stages $S[96]$ to $S[175]$, the $IV$ is loaded into stages $S[176]$ to $S[255]$, stage $S[95] = 1$, and the rest of the stages
are loaded with zero. Then, the cipher works using the initialisation update function for 128 iteration without any keystream bits generated, as illustrated with dashed lines in Figure 3.9. At each step, the LFSR contents are updated using the update function 3.5. In addition, the output bits of the INV function are delayed and combined with the shift register contents to update the LFSR. At the same time, bit number \( y[1] \) is used to update register \( R \) by shifting all contents of register \( R \) one step left and the update bit \( y[1] \) is used in the last stage of register \( R \). Also register \( A \) is updated during the initialisation phase by XORing the whole contents of it with the contents of register \( R \) for all the times of initialisation phase. By the end of this phase, the LFSR, register \( R \) and register \( A \) are loaded with the initial states.

During keystream generation, the LFSR is updated using the register update function only, without the feedback from the INV function:

\[
\]

Then, at each clock, a keystream bit \( Z_t \) is generated using the following equation:

\[
Z_t = S^{t-7}[0] \oplus y^{t-7}[0]
\]

where \( y^{t-7}[0] \) is bit position number 0 of the output of the INV function at time \( t - 7 \).

**MAC tag generation**

After the initialisation phase of the cipher, both registers \( R \) and \( A \) of the integrity component are initialised. Then the plaintext is input to both the confidentiality component and the integrity component. For the integrity component, the plaintext bit \( P_t \) controls which keystream segment is accumulated into register \( A \). Note that the input plaintext is not accumulated into the internal state of the integrity component. It is only used as a switch to control when the contents of register \( R \) is XORed with register \( A \). If the value of input plaintext bit is 1, then register \( A \) is updated by XORing with the contents of register \( R \). Otherwise, if the value of input message is 0, then there is no change in register \( A \). For register \( R \), at each clock a new bit value which is bit number \( y[1] \) of the output of the INV function is used to update register \( R \) by simply shifting one bit and
discarding the last bit of the register, in the same manner as was used during the initialisation phase. Then, after the input message has been processed, the final contents of register $A$ are encrypted using the keystream output. More details of MAC generation using Sfinks stream cipher are in Section 6.2.3.

**Previous analysis**

A previous attack applied was algebraic attacks \cite{25} on Sfinks’s confidentiality component. The attack basically constructs equations that relate the initial state of the LFSR with the keystream output bits. Firstly, these equation are constructed using simple straightforward algebraic attack to get equations with a degree of 15. To solve these equations requires $2^{79.2}$ keystream bits with complexity of $2^{222}$ operations.

In the same paper, the author used computer simulation to find equations with lower degree that can multiply with the equations of the Boolean function of Sfinks. This enabled a fast algebraic attack to be performed. The results were that the fastest attack on Sfinks needs $2^{49}$ keystream bits with complexity of $2^{70}$ operations. As the Sfinks key size is 80 bits, it was concluded that Sfinks stream cipher is insecure.

**Classifications of Sfinks for providing AE**

The plaintext is first encrypted to get the ciphertext. At the same time a MAC tag is generated for the plaintext. Then final MAC tag is encrypted using the keystream bits and appended with the ciphertext to form a message that will be sent to other side. So if we compare the process of forming the sent message with the above methods of Bellare and Namprempre \cite{11,12}, we can classify Sfinks under the MAC-then-Encrypt (MTE) scheme. Based on their results (Table 3.1), this scheme may not provide integrity of ciphertext security and also may not provide non-malleability for ciphertext.

For the second classification of Sfinks, the input message is not loaded into the internal states of the integrity component. It is only used to control which segment of keystream bits have to be accumulated into the internal states. Thus, Sfinks can be classified under the leading of indirect input message injection.

For the third classification, Sfinks stream cipher uses single key-IV pair for both mechanisms confidentiality and integrity.
3.3.4 NLSv2

NLSv2 (Non-Linear Sober Version 2) \[56\] is an updated version of NLS \[85\]. Both are synchronous stream ciphers and use a secret key and optional IV, each 128 bits long. These were submitted to eSTREAM \[78\], the ECRYPT stream cipher project, as ciphers suitable for software applications (Profile I). NLSv2 generates a 128-bit MAC tag.

**Keystream generation**

The NLSv2 keystream generator consists of a 17-stage nonlinear shift register $R$ and a NLF as shown in Figure 3.10. Each register stage $R[0], \ldots, R[16]$, contains a 32-bit word. Thus, the total internal state size is 544 bits.

![Keystream generator diagram](image)

Figure 3.10: NLSv2 keystream generator and MAC generation

During the initialisation phase, the secret key and the optional IV are loaded into the stages of the shift register using the key and IV loading function. By the end of this loading, the register is loaded with the initial states, a key dependent word ($Konst$) is generated and the two registers ($SHA$ and $A$) of the integrity component are initialised with initial states.

During keystream generation, the shift register is updated using the following function:
3.3. Description of specific AE stream ciphers

\[
R[i]^{t+1} = \begin{cases} 
R[i + 1]^t & \text{for } i = 0, \ldots, 15, \\
\left(f((R[0] \ll 19) + R[15] \ll 9) + Konst\right) \oplus R[4] & \text{for } i = 16.
\end{cases}
\]

where + is addition modulo $2^{32}$, and $f$ is defined by $f(a) = SBox(a_H) \oplus a$, $a_H$ is the most significant 8 bits of 32-bit word $a$, and $SBox$ is an $8 \times 32$-bit substitution box.

Then, at each clock, a keystream word $Z^t$ is generated using the following equation:

\[
Z^t = (R[0] + R[16]) \oplus (R[1] + R[13]) \oplus (R[6] + Konst)
\]

If the time during the keystream generation $t \equiv 0$ modulo $(2^{16} + 1)$, then stage $R[2]$ is modified by adding $t$ modulo $2^{32}$, and $Konst$ is changed to the resulting value of $Z^t$. Then, the other keystream words $Z^t$ are used to encrypt the plaintext words $P^t$ by XOR operation to form ciphertext words $C^t$. Note that there is no output word at the clock when the $Konst$ is updated.

MAC tag generation

The NLSv2 MAC generator consists of two registers as shown in Figure 3.10. The first register is a SHA-256 register that uses nonlinear feedback combined with the input message. The second register is an 8-stage LFSR denoted register $A$ in Figure 3.10. Each stage of register $A$ ($a[0], \ldots, a[7]$), contains a 32-bit word. This register also uses the input message with the feedback to update the state of the register.

The state update function for register $A$ is given as follows:

\[
a[i]^{t+1} = \begin{cases} 
a[i + 1]^t & \text{for } i = 0, \ldots, 6, \\
P^t \oplus (\delta \otimes a[0]) \oplus a[5] & \text{for } i = 7.
\end{cases}
\]  

(3.6)

where $\delta$ is a non-trivial element in GF($2^{32}$). The MAC generation process begins by preparing the input message by padding with zero bits such that the resulting length is a multiple of 32 bits. Then the message is accumulated into both registers $A$ and SHA-256 which are initialised with keystream words at the initialisation phase of the cipher, as discussed above. When the message has been processed, the finalisation phase is started to form a MAC tag. This process is as follows:
• A padding word is formed of value $b \gg 3$ (where $b$ is the number of bits added into the message) and this word is added to register SHA-256 but this word is not input into register $A$.

• The content of register $A$ at time $l$, where $l$ is the size of the input message in words, is input to register SHA-256 and treated like the input message.

• To generate a MAC of length 128, 4 words are then generated from SHA-256 and these words are considered to be the MAC tag.

**Previous analysis**

Cho and Pieprzyk [25] used a technique based on the known plaintext attack called a distinguishing attack on NLSv2’s confidentiality component. This attack investigates high correlation between two neighbouring bits of the cipher. In their analysis, they noted that bit number 29 and 30 of the output of SBox had a high correlation. So the attack exploits this correlation to make a linear approximation of the NLF. As a result of that, a distinguisher of NLSv2 has a bias of $2^{-37}$ and so the attacker required only $2^{79}$ words of keystream to distinguish the keystream of NLSv2 from random. Cho and Pieprzyk claim that this falls within the amount of keystream generated ($2^{80}$ words) before rekeying needs to be done. However, from the NLSv2 specifications [56] a new session key should be generated after at most $2^{53}$ words of plaintext are encrypted. This means that the distinguishing attack of Cho and Pieprzyk has no practical significance.

**Classifications of NLSv2 for providing AE**

The plaintext is first encrypted to get the ciphertext. At the same time a MAC tag is generated for the plaintext. Then this final MAC tag is encrypted using the keystream words and appended with the ciphertext to form a message that will be sent to other side. So if we compare the process of forming the sent message with the above methods of Bellare and Namprempre [11][12], we can classify that NLSv2 is under the MAC-then-Encrypt (MTE) scheme. Based on their results (Table 3.1), this scheme may not provide integrity of ciphertext security and also may not provide non-malleability for ciphertext.

For the second classification of NLSv2, we can see that the input message is loaded into the dedicated internal states of the integrity component. Thus, NLSv2 can be classified under direct input message injection.

In the third classification, NLSv2 uses single key-IV pair for both mechanisms confidentiality and integrity.
3.3. Description of specific AE stream ciphers

### 3.3.5 SOBER-128

SOBER-128 \(^{[54]}\) is a synchronous stream cipher, and a member of the SOBER family \(^{[86]}\) that claims to offer optional AE of messages. It was designed as an improved version of the SOBER-t32 \(^{[58]}\) and SOBER-t16 \(^{[57]}\) ciphers that were submitted to the NESSIE project. Both were rejected due to security flaws reported in \(^{[8, 21]}\). SOBER-128 takes as input a 128-bit key \(K\) and an optional 128-bit \(IV\). The keystream is produced as a series of 32-bit words. The cipher also generates a 128-bit MAC tag.

#### Keystream generation

The SOBER-128 keystream generator consists of a 17-stage LFSR, denoted \(A\), and a NLF, as shown in Figure 3.11. Each register stage \(a[0], \ldots, a[16]\), contains a 32-bit word. Thus, the total internal state size is 544 bits. Note that if the cipher is used for generating a MAC tag, then the LFSR will be updated in a nonlinear manner.

![Figure 3.11: SOBER-128 keystream generator and MAC generation](image)

During the initialisation phase, the secret key and the optional \(IV\) are loaded into the stages of the LFSR using the key and \(IV\) loading function. By the end of this loading process, the register \(A\) is initialised with the initial states and a key dependent word (\(Konst\)) has been defined.

If the cipher is used only for keystream generation, then the LFSR is updated using the following function:

\[
a[i]_{t+1} = \begin{cases} 
a[i+1]_t & \text{for } i = 0, \ldots, 15. \\
 a[15]_t \oplus a[4]_t \oplus a.a[0]_t & \text{for } i = 16.
\end{cases}
\]
where $+$ is addition modulo $2^{32}$, and $\alpha$ is a non-trivial element in GF($2^{32}$).

Then, at each clock, a keystream word $Z^t$ is generated using the following equation:


where $f$ is defined by $f(a) = SBox1(a_H)||SBox2(a_H \oplus a_L)$, $a_H$ and $a_L$ are the most significant 8 bits and the least significant 24 bits of 32-bit word $a$ respectively. $SBox1$ and $SBox2$ are two substitution boxes with sizes $8 \times 8$ bits and $8 \times 24$ bits respectively.

Then, the output 32-bit keystream words $Z^t$ are used to encrypt the 32-bit plaintext words $P^t$ by XOR operation to form ciphertext words $C^t$.

**MAC tag generation**

The SOBER-128 MAC generator consists of the same LFSR that is used for keystream generation and an additional nonlinear filter function, labeled with integrity component in Figure 3.11. Note that the integrity component in Figure 3.11 is only used if the ciphers used for MAC generation, and in this case, the state update function for register $A$ is given as follows:

$$a[i]^{t+1} = \begin{cases} a[i + 1]^t & \text{for } i = 0, \ldots, 15. \\ f((f(P^t + a[4]^t) \gg 8) + Konst) & \text{for } i = 4. \\ a[15]^t \oplus a[4]^t \oplus \alpha.a[0]^t & \text{for } i = 16. \end{cases} \quad (3.7)$$

The MAC generation process begins by preparing the input message by padding with zero bits such that the resulting length is a multiple of 32 bits. Then the message is accumulated into register $A$ using the update function described by Equation (3.7). Once the message has been processed, the finalisation phase is applied to form a MAC tag. This process is as follows:

- A padding word is formed by (0x6996c53a $\oplus$ num_pad), where num_pad is the number of padding bits, and this word is loaded into $a[15]$ of register $A$ by addition modulo $2^{32}$.
- Register $A$ is clocked 18 times, and at each time $t$, the value of $a[4]^t$ is replace with $(a[4]^t \oplus Z^t)$, where $Z^t$ is the output keystream word at time $t$.
- Generate four words of keystream $Z$ to form a MAC tag for the input message.
3.3. Description of specific AE stream ciphers

Previous analysis
A MAC forgery attack \[98\] was applied to SOBER-128’s integrity component. The attack is based on finding two distinct messages \( P \) and \( P' \) such that they give the same internal state value of the register at a certain time \( t \), used with the same key and \( IV \). The attack starts by searching the differential propagation in the integrity component with high probability and then constructing the differential elimination in the LFSR. The attack exploits the fact that the substitution \( f \) is not uniformly nonlinear and the most significant byte of the input word is not influenced by \( f \). It was found that any pairs of messages with the related formats \((0, \delta_1, \delta_2, 0)\) and \((0, \delta'_1, \delta'_2, 0)\), give the same internal state, and after 17 clocks of the register, the contents of the register for the two messages will be same with probability of approximately \(2^{-6}\).

Classifications of SOBER-128 for providing AE
The plaintext is first encrypted to get the ciphertext. At the same time a MAC tag is generated for the plaintext. Then the final MAC tag is encrypted using the keystream words and appended with the ciphertext to form a message that will be sent to other side. So if we compare the process of forming the sent message with the above methods of Bellare and Namprempre \[11,12\], we can classify SOBER-128 use a MAC-then-Encrypt (MTE) scheme. Based on the results in (Table 3.1), this scheme may not provide integrity of ciphertext security, but the integrity component of SOBER-128 is accumulated plaintext instead of ciphertext, and also the scheme may not provide non-malleability for ciphertext.

For the second classification of SOBER-128, we can see that the input message is loaded into the dedicated internal states of the integrity component, and the input message words are loaded into these internal states. Thus, SOBER-128 can be classified under direct input message injection.

In the third classification, SOBER-128 uses single key-\( IV \) pair for both mechanisms confidentiality and integrity.

3.3.6 Grain-128a

Grain-128a \[1\] is a new version of Grain-128 \[59\]. Both are members of the Grain family \[60\] but Grain-128a has built-in support for authentication. The cipher takes as input a 128-bit key \( K \) and optional 96-bit \( IV \). One bit of keystream is produced at each clock. The cipher also generates a 32-bit MAC tag.

Keystream generation
Chapter 3. Authenticated encryption using stream ciphers

The Grain-128a keystream generator consists of two registers and a nonlinear filter and generates a keystream bit at each clock time. The first register is a LFSR, denoted by $S$ and the second register is a nonlinear feedback shift register (NFSR) denoted by $b$ as shown in Figure 3.12. Each stage of registers $S = S[0], \ldots, S[127]$, and $b = b[0], \ldots, b[127]$, contain a bit value. Thus, the total internal state size of the confidentiality component is 256 bits.

During the initialisation phase, the secret key $K = k[0], \ldots, k[127]$ and $IV = v[0], \ldots, v[95]$ are loaded into the internal state as follow: $b[i] = k[i], 0 \leq i \leq 127$, $S[i] = v[i], 0 \leq i \leq 95$, $S[i] = 1, 96 \leq i \leq 126$, and $S[127] = 0$. The state update function for register $S$ during initialisation is given as follows:

$$S[i]^{t+1} = \begin{cases} S[i+1]^t & \text{for } i = 0, \ldots, 126, \\ S[0]^t + S[7]^t + S[38]^t + S[70]^t + S[81]^t + S[96]^t + S[38]^t + y^t & \text{for } i = 127. \end{cases}$$

The state update function for register $b$ during this phase is given as follows:


During the initialisation phase no keystream bits are generated. The state
update function during initialisation is illustrated with dashed lines in Figure 3.12. At each time clock the output bit $y_t$ is used to update the two registers $S$ and $b$. The value $y_t$ is calculated using the following equation:

$$y_t = h(x) + S[93]_t + b[2]_t + b[15]_t + b[36]_t + b[45]_t + b[64]_t + b[73]_t + b[89]_t$$


By the end of this phase, the two registers $S$ and $b$ contain with the initial states.

During the keystream generation phase, the two registers $S$ and $b$ are updated using the same equations as above but without feeding back bit $y_t$, and the keystream bits $Z_t$ are generated using the following equation:

$$Z_t = y^{64+2t}$$

Note that the first 64 bits of $y$ after the initialisation phase are used to initialise the integrity component. This will be discussed below. Following this, the output keystream bits $Z_t$ are used to form ciphertext bits $C_t$ by XORing with plaintext bits $P_t$.

**MAC tag generation**

After the initialisation phase of the cipher, the first 64 bits of keystream $y$ are used to initialise the two 32-bit registers $R = r[0], \ldots, r[31]$ and $A = a[0], \ldots, a[31]$ of the integrity component as follows: $r[i] = y^i, 0 \leq i \leq 31$, and $a[i] = y^{32+i}, 0 \leq i \leq 31$. In the preparation phase of MAC generation, the message is padded with one bit of value 1 at the end of the message. Then the message is input to both the confidentiality component and the integrity component. In the integrity component, the ciphertext bit $c_t$ is used to control the keystream segments accumulated into register $A$, as follows. If the value of the input ciphertext bit is 1, then register $A$ is updated by XORing the contents of register $A$ with the contents of register $R$, otherwise there is no change in register $A$. For register $R$, at each clock a new bit value $y^{64+2t+1}$, which is the second bit of keystream, is used to update register $R$ by simply shifting one bit, discarding the last bit and loading the new value at $r[31]$ of the register. More details of MAC generation using Grain-128a stream cipher are provided in Section 6.2.2.

**Previous analysis**

Grain-128a is a very recent version of Grain-128, and there is currently no attack on it. However, there is an existing attack on Grain-128. The previous attack
applied to Grain-128 is a fault attack [15]. This attack assumes that the attacker knows the value of the IV and the keystream generated using this IV and tries to recover the initial states of the LFSR and the NFSR and then recover the secret key. The attack also assumes that it is possible to flip exactly one bit lying in one position of the LFSR, without choosing its location, but at chosen time. This attack can be summarised in five steps. In the first step, the attacker uses a test device to find the exact location in the LFSR that the attacker can flip the bit contained in the cell. It is possible to establish the position by analysing the keystream difference. In the second step, the attacker uses the target device that contains the wanted secret key. It is assumed that the attacker is able to induce a fault at the same position in the LFSR at different times. The bits of LFSR that are recovered depend on both the fault location and the number of times the cipher is clocked after the fault is injected. In third step, a system of linear equations is constructed with unknowns of the initial states of LFSR and with known keystream bits. Solving this system will recover the initial state of the LFSR. In the fourth step, several algorithms to recover the initial state of the NFSR are constructed based on the injecting a fault in the LFSR permitting knowledge of the initial state of the LFSR. The system of equations is developed with linear relation and solving this system reveals the initial state of the NFSR. They found that the faults used in step three to recover the LFSR state are sufficient to recover the NFSR state with a couple of minutes. Based on the previous steps, the last step is to recover the secret key. This is obtained from knowing the state of the NFSR at time $t$ and by computing the state of the NFSR at time $t - 1$.

They found that with an average number of 24 consecutive faults in the LFSR state, they can recover the secret key within a couple of minutes of offline computation of the target device.

**Classifications of Grain-128a for providing AE**

The plaintext is first encrypted to get the ciphertext. At the same time a MAC tag is generated for the message. It is not clear from the cipher specification whether the message used for MAC generation is plaintext or ciphertext. However, the designers claim that the authentication mechanism of Grain-128a is similar to ZUC so we will consider this message as a ciphertext. The final MAC tag is appended with the ciphertext to form a message that will be sent to other side. So if we compare the process of forming the sent message with the above
methods of Bellare and Namprempre \[11,12\], we can classify that Grain-128a is under the Encrypt-then MAC (ETM) scheme. Based on the assumption that the integrity mechanism is strongly unforgeable, this scheme according to Table 3.1 could provide all the five security services that has been mentioned above.

For the second classification of Grain-128a, the input message is not loaded into the internal states of the integrity component. It is only used to control the segments of keystream that are accumulated into the internal states. Thus, Grain-128a can be classified as indirect input message injection.

For the third classification of Grain-128a, Grain-128a uses single key-IV pair for both mechanisms confidentiality and integrity.

### 3.3.7 SNOW 3G

SNOW 3G \[93\] is an updated version of the keystream generator SNOW 2.0 \[36\] to also provide integrity assurance. SNOW 2.0 provided only confidentiality, using 128 bits or 256 bits of key size and 128 bits of IV. SNOW 2.0 and SNOW 3G are word-based keystream generators for stream ciphers. Both generate keystream in 32-bit words. SNOW 3G uses a secret key and IV, each 128 bits long (4 words each).

**Keystream generation**

The SNOW 3G keystream generator consists of a LFSR and a Finite State Machine (FSM) as shown in Figure 3.13. The LFSR consists of 16 stages, $R[0], \ldots, R[15]$, and each stage contains a 32 bit word. The FSM is constructed from three registers $R1, R2, R3$, with each register containing a 32-bit word. Thus, the total internal state size is 608 bits.

![Figure 3.13: SNOW 3G keystream generator](image)
As shown in Figure 3.13 SNOW 3G uses two SBoxes $S_1, S_2$, multiplication on the GF($2^{32}$) (denoted by $\otimes$), XOR and addition modulo $2^{32}$.

During the initialisation phase, the key $K = k[0], k[1], k[2], k[3]$ and $IV = v[0], v[1], v[2], v[3]$ words are loaded into the stages of the register as follows:

- $R[15] = k[3] \oplus v[0]$,
- $R[13] = k[1]$,
- $R[12] = k[0] \oplus v[1]$,
- $R[11] = k[0] \oplus 1$,
- $R[9] = k[1] \oplus v[3] \oplus 1$,
- $R[8] = k[0] \oplus 1$,
- $R[4] = k[0]$,
- $R[3] = k[3] \oplus 1$,
- $R[2] = k[2] \oplus 1$,
- $R[1] = k[1] \oplus 1$,
- $R[0] = k[0] \oplus 1$,

where 1 is a word of all ones (0xffffffff) and registers $R1, R2, R3$ in the FSM are initialised with zeroes. Then the state update function for the initialisation phase for the LFSR and FSM is applied 32 times. There is no keystream output during this phase. The output is fed back to the LFSR, as indicated by dashed line in Figure 3.13. This function works as follows:

$$x^t = (R[15]^t + R1^t) \oplus R2^t$$

$$R[15]^{i+1} = x^t \oplus (\alpha \times R[0]^t) \oplus R[2]^t \oplus (\alpha^{-1} \times R[11]^t)$$

$$R[i]^{i+1} = R[i + 1]^t, \quad \text{for } i = 0, 1, \ldots, 14$$

$$R1^{i+1} = R[5]^t \oplus R3^t + R2^t$$

$$R2^{i+1} = S_1(R1)^t$$

$$R3^{i+1} = S_2(R2)^t.$$ 

where $S_1$ and $S_2$ are each made of four parallel 8-bit SBoxes followed by a mul-
3.3. Description of specific AE stream ciphers

Multiplication by a $4 \times 4$ matrix over $\text{GF}(2^8)$.

The keystream word $Z^t$ is obtained as a nonlinear combination of the contents of FSM with $R[15]$ XORed with $R[0]$, as shown in Figure 3.13 and given by following equation:

$$Z^t = (R[15]^t + R1^t) \oplus R2^t \oplus R[0]^t$$

(3.8)

For confidentiality, the first word of the keystream generated $Z^0$ is discarded. The other words $Z^1, Z^2, \ldots$, are used to encrypt the plaintext or decrypt the ciphertext using UEA2 [92,94].

A second SNOW 3G keystream generated using a different key and IV is used to generate a 32-bit MAC using the UIA2 [92,93] algorithm as follows:

**MAC tag generation**

The UIA2 algorithm [92,93] uses the SNOW 3G keystream generator to generate five words of keystream $Z^0, \ldots, Z^4$ which are then used to provide integrity for an input message of length up to 2500 bytes. The first two words $Z^0, Z^1$ are used with a multiplication function to accumulate the input ciphertext. Then, the output of this accumulation process are 64 bits which are XORed with the length of the input message. The resulting output 64 bits are then multiplied with the following two words of the keystream $Z^2, Z^3$. Then, the first 32 bits of this multiplication are XORed with the last word of the keystream $Z^4$ and the output word is considered to be the MAC tag for the input message.

**Statistical analysis for UIA2 based on SNOW 3G**

We applied the repetition test as described in Section 2.2.2 to MAC tags generated from UIA2 algorithm to examine the hypothesis that a uniform distribution of 32-bit tuples is being generated. Seven different experiments were conducted each forming a sample size of $2^{19}$ MACs. Three different formats for the input data for the formation of a sample of MAC tags were used namely:

1. random message and fixed key,
2. random key and fixed message and
3. random key and random message

The results of the tests are presented in Table 3.3. From observation of this table, p-value for all the samples support the hypothesis that the MAC tags generated from UIA2 are uniformly distributed.
Table 3.3: Summary of the results for UIA2 algorithm

<table>
<thead>
<tr>
<th>Message</th>
<th>Key</th>
<th>Number of repetitions observed</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random 1</td>
<td>same</td>
<td>28</td>
<td>0.4795</td>
</tr>
<tr>
<td>Random 2</td>
<td>same</td>
<td>40</td>
<td>0.1573</td>
</tr>
<tr>
<td>Random 3</td>
<td>same</td>
<td>34</td>
<td>0.7237</td>
</tr>
<tr>
<td>Random 4</td>
<td>same</td>
<td>31</td>
<td>0.8597</td>
</tr>
<tr>
<td>Same</td>
<td>Random 1</td>
<td>25</td>
<td>0.2159</td>
</tr>
<tr>
<td>Same</td>
<td>Random 2</td>
<td>28</td>
<td>0.4795</td>
</tr>
<tr>
<td>Random 4</td>
<td>Random 3</td>
<td>32</td>
<td>1.0</td>
</tr>
</tbody>
</table>

**Previous analysis**

A fault attack on SNOW 3G is presented in [5]. This attack assumed that the attacker is able to modify a 32-bit value of $S[i]^t$, where $0 \leq i \leq 15$, during the keystream generation and the stage $i$ is chosen by the attacker but he does not have full control on the time $t$. To run the attack, the attacker injects a fault at the RAM location where the value $S[i]$ is stored. The attacker assumes that it is possible to inject faults at the same location and time $t$ during several executions with the same IV. During the fault setup stage, the attacker has to find the right location in the LFSR stages on which the fault must be injected. Stages $S[2]$, $S[3]$ or $S[4]$ are the right location to inject the fault, and modify the values of these stages during the execution of the cipher software. The attacker then analysed the output keystream difference induced by the injection on each stage. Then a system of equations from the observed keystream words is obtained. Solving this system reveals the secret key $K$.

The attack requires the injection of 22 faults on the selected stages of the LFSR, on average, and it is possible to recover the secret key in less than one minute of off-line computations. The attack also requires 200 output keystream words using the same key and IV. However, from the SNOW 3G specifications [92] a new session key should be generated after at most 2500 bytes (approximately 78 words) of plaintext are encrypted. This means that the fault attack of [5] has no practical significance.

**Classifications of SNOW 3G for providing AE**

The plaintext is first encrypted to get the ciphertext using the UEA2 algorithm. At the same time a MAC tag is generated for the ciphertext. Then the final
MAC tag is appended with the ciphertext to form a message that will be sent to the other side. So if we compare the process of forming the sent message with the above methods of Bellare and Namprempre \[11,12\], we can classify that SNOW 3G under the Encrypt-then-MAC (ETM) scheme. Based on the assumption that the integrity mechanism is strongly unforgeable, this scheme according to Table \[3.1\] could provides all the five security services.

For the second classification, SNOW 3G does not qualify for either method of message injection, since the input message is accumulated using the multiplication function with the keystream words. It could be considered as a modified form of indirect injection method.

For the third classification of SNOW 3G, SNOW 3G uses two different key-IV pairs for both mechanisms confidentiality and integrity.

### 3.4 Summary and Conclusion

This chapter provided a summary of the current literature on authenticated encryption using stream ciphers that is relevant to the work presented in this thesis. It has also discussed three methods for classification for AE stream ciphers. The first method was introduced by Bellare and Namprempre and is based on the order of occurrence for the encryption and authentication in the AE algorithm. The second method is based on how the input message is injected during the accumulation phase in the MAC generation algorithms. The third method of classification is discussed in greater detail in Chapters 5 and 6 of this thesis. The third method is based on whether a single keystream is used for both the encryption and integrity mechanisms or whether two different keystream generators are used. Specific AE stream cipher proposals described in Section 3.3 have been classified according to the methods of classification. Table \[3.4\] provides a summary of the classification for these examples of AE in relation to all three different classification methods. From this classification, it can be seen that the more recent AE stream cipher proposals since eSTREAM project namely ZUC, Grain128a and SNOW 3G are all ETM and indirect injection of the input message. As shown in Table 3.1, schemes in the ETM category have the benefit that they provide all five security services under assumption that the integrity component is unforgeable. We examine in Chapters 5 and 6 under which conditions this assumption is correct.
Table 3.4: Summary of the classification for several AE ciphers

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Date</th>
<th>Bellare and Namprempre</th>
<th>Message Injection</th>
<th>Key used</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOBER-128</td>
<td>2003</td>
<td>MTE</td>
<td>direct</td>
<td>single</td>
</tr>
<tr>
<td>SSS</td>
<td>2005</td>
<td>MTE</td>
<td>direct</td>
<td>single</td>
</tr>
<tr>
<td>Sfinks</td>
<td>2005</td>
<td>MTE</td>
<td>indirect</td>
<td>single</td>
</tr>
<tr>
<td>NLSv2</td>
<td>2005</td>
<td>MTE</td>
<td>direct</td>
<td>single</td>
</tr>
<tr>
<td>SNOW 3G</td>
<td>2006</td>
<td>ETM</td>
<td>indirect</td>
<td>two</td>
</tr>
<tr>
<td>Grain-128a</td>
<td>2011</td>
<td>ETM</td>
<td>indirect</td>
<td>single</td>
</tr>
<tr>
<td>ZUC</td>
<td>2011</td>
<td>ETM</td>
<td>indirect</td>
<td>two</td>
</tr>
</tbody>
</table>

Practical experiments were performed for two specific authentication algorithms, 128-EIA3 and UIA3, where the MAC size is 32 bits, to confirm that the generated MAC tags are equally likely to occur for different messages using either the same or different keys.
Chapter 4

Algebraic analysis of SSS and ZUC stream ciphers

If a single key-IV pair is used for both confidentiality and integrity mechanisms, a key recovery against one mechanism compromises the other service as well. So it is essential to investigate each mechanism individually. Any weakness in either one of these mechanisms results in weakness for the whole AE cipher. This chapter focuses on the confidentiality mechanism for AE stream ciphers only by applying the attacks that were applied to stream ciphers which is algebraic attack.

Algebraic attacks on stream ciphers work by finding equations describing relations between the secret key or the initial content of the internal state and the output keystream, as outlined in Chapter 2. These equations are presented as multivariate polynomials of defined degree. The relations can be obtained directly or indirectly. Direct relations mean straightforward mathematical representations between the secret key bits or the internal state bits and the output keystream bits are obtained. Indirect relations use other methods to obtain the relation, such as considering some functions of the cipher as a combiner. In either case, once the relations are obtained, they are applied at multiple time steps $t = 0, 1,\ldots$ and the variables representing the keystream outputs $z_0, z_1,\ldots$ are replaced by their observed values, to obtain a system of multivariate equations with low degree. Finally, the system is solved to recover the unknowns in the equations which are either the initial state bits of the cipher or the secret key.
Chapter 4. Algebraic analysis of SSS and ZUC stream ciphers

In this chapter the SSS and ZUC stream ciphers are analysed algebraically using an indirect method to establish valid relationships between the initial state bits and the output keystream bits. These analyses follow the attack methodology presented in [21], where it was applied in an attack on SOBER-t32 [58] to recover the initial state of the cipher. This approach treats the nonlinear function (NLF) of the cipher as a combiner with memory to build indirect relationships among various bits of the ciphers.

The chapter is organised as follows. Section 4.1 describes the approach used to model a combiner with memory. An algebraic analysis of SSS is presented in Section 4.2, while Section 4.3 provides an algebraic analysis of ZUC stream cipher. A summary of the work in this chapter and conclusions are given in Section 4.4.

4.1 Modeling a combiner with memory

A nonlinear function in a stream cipher can be considered as a combiner, either with or without internal memory. A combiner has \( b \) input bits from \( n \) stages of the shift register, \( m \) output bits of keystream and \( c \) bits of internal memory (\( c \) could be zero for a memoryless combiner).

The aim of the first step in an algebraic attack is to find relations among the output bits with the input bits, of the nonlinear function to be considered as a combiner. These relations are used later to construct a system of equations. To apply an algebraic attack to the combiner, it is shown in [6] that if \( b \) and \( c \) are fixed then there is a polynomial of degree at most \( \lceil b(c + 1)/2 \rceil \), relating the output bits to the input bits. The output bits of the combiner \( z_0, z_1, \ldots, z_{m-1} \) at each time step \( t = 0, 1, 2, \ldots, m - 1 \) depend on both the \( b \) input bits from the stages of the shift register and on the internal memory bits \( a_0, a_1, \ldots, a_{c-1} \) before and at time \( t \).

The goal of the first step of the attack is to eliminate all the monomials in \( a_i \) and establish relationships between the input bits from the shift register stages and the output keystream bits without the memory bits [29]. To eliminate these monomials in memory bits, a matrix representing the combiner is constructed. The rows of the matrix represent all \( 2^{b+c} \) possible values of the input bits and the memory bits. The columns represent all the monomials in the variables of
input bits and output bits, up to a certain degree. Gaussian elimination [95] is applied to the constructed matrix to find linear dependencies and derive valid relations for the combiner. From these relations, a system of equations is built using keystream observations.

After the system of equations is developed, the second stage of the attack is to replace keystream variables by their values, and to solve this system. Solving the system may be simple if the equations are linear, or easily linearised. The XL algorithm [26] described below may be applied to achieve this in a preprocessing step before solving the equations. This approach is used in this thesis. Another approach is to use Gröbner basis algorithms such as $F_5$ [19, 41]; essentially transforming a set $F$ of equations into another set $G$ such that $F$ and $G$ have the same solution, but $G$ can be used to directly compute this solution.

**XL Algorithm**

The XL algorithm was proposed in 2000 as a means for solving overdefined quadratic systems [26], and adapted for equations of higher degree in 2002 [27]. The idea of the XL algorithm is to multiply the initial system of equations in $n$ variables with maximum degree $d$ by all possible monomials of degree up to $d$. If the degree of a resulting equation is equal to or less than $d$, then this new equation is added to the system. After this the linearisation approach outlined in Section 2.1.3 is applied to the resulting system of equations. Let $T$ denote the number of monomials in a polynomial system with maximum degree $d$. Then $T$ is given by

$$T = \sum_{i=0}^{d} \binom{n}{i}$$  \hspace{1cm} (4.1)

The complexity of the XL algorithm depends on the elimination technique used. Strassen’s algorithm [95] is widely applied for this purpose, requiring about $7 \cdot T \log_2 7$ operations.

### 4.2 Algebraic analysis of SSS

In this section, the algebraic attack applied to SOBER-t32 [21] is modified, and an attempt is made to apply it to SSS, since SSS and SOBER-t32 have similar designs and both are from the SOBER [86] family of stream ciphers. Each cipher design is based on a single 17-stage word-based shift register, and uses a nonlinear filter function (NLF) to generate the keystream outputs. In both cases the word-
based NLF makes use of a Substitution Box (SBox), and both modular and binary addition (XOR) are performed over the word size. Although the structures are similar, there are some important differences between these two designs. SOBER-t32 is a synchronous stream cipher that uses 32-bit words, a 256-bit key (8 words) and a linear feedback function for the shift register. SSS is a self-synchronous stream cipher that uses 16-bit words, a 128-bit key (also 8 words) and a nonlinear feedback function for the shift register. Although both designs apply a nonlinear filter to stages of the shift register in producing keystream, there are differences in the components of these functions. For SOBER-t32, the SBox used is fixed, and it is applied once in the NLF, whereas the SBox used for SSS is key dependent, and it is used twice in the NLF. A final difference is that for SOBER-t32, the output of the NLF is stuttered before it is used as an output keystream. For SSS the output of NLF is used directly as the keystream. Besides keystream generation, SSS is also used to generate a 128-bit MAC tag. This functionality is explored in more detail in Chapter 5.

The SOBER-t32 cipher, with the stuttering function removed, has been attacked using an algebraic attack to recover the initial states of the shift register [21]. The basic idea of this attack is to develop a set of equations for the NLF of SOBER-t32 that relate the inputs to the NLF (taken from the shift register stages) with the keystream outputs by considering part of the NLF as a combiner. A matrix is constructed to obtain valid equations relating the inputs to the combiner with the outputs, and then by linking these equations with the keystream output equation, a system of equations that relate the keystream bits with the initial state bits is obtained. For a known keystream segment, solving the set of equations permits the recovery of the initial state bits of the shift register.

In Section 4.2.1, the SOBER-t32 keystream generator is described. Then in Section 4.2.2, the previous successful algebraic attack on a modified version of SOBER-t32 is explained in more detail. The description of the SSS keystream generator was given in Section 3.3.1. In Section 4.2.3, the existing attack on SOBER-t32 [21] is adapted in various ways and applied to SSS. Finally, algebraic analysis on a modified version of SSS is discussed in Section 4.2.4.
4.2.1 Description of SOBER-t32 keystream generator

The SOBER-t32 keystream generator consists of a 17-stage Linear Feedback Shift Register (LFSR) and a nonlinear filter function (NLF) as shown in Figure 4.1. Each register stage $S[0], \ldots, S[16]$ contains a 32-bit word. Thus, the total internal state size is 544 bits.

The state update function for the LFSR is as follows:

$$S[i]^{t+1} = \begin{cases} S[i + 1]^t & \text{for } i = 0, \ldots, 15 \\ S[15]^t \oplus S[4]^t \oplus \delta \cdot S[0]^t & \text{for } i = 16. \end{cases}$$

where $\delta = 0xc2db2aa3$.

The output function for SOBER-t32 is calculated in two stages. Firstly, the intermediate value $V^t$ is obtained as a nonlinear combination of the contents of five stages of the LFSR and the key dependent constant $K$. The value of the constant $K$ is determined during the initialisation phase of the LFSR and is retained for the entire session. As shown in Figure 4.1, $V^t$ is given by the
the following equation:

\[ V^t = \left( (g(S[0]^t + S[16]^t) + S[1]^t + S[6]^t) \oplus K \right) + S[13]^t \]

The function \( g \) uses a fixed SBox with an 8-bit input and 32-bit output. Let \( D \) denote the 32-bit word input to the function \( g \), \( D_H \) denote the 8 most significant bits and \( D_L \) denote the 24 least significant bits. In [58] the function \( g \) is given by:

\[ g(D) = SBox(D_H) \oplus (0||D_L) \quad (4.2) \]

If the function \( g \) in Figure 4.1 is expanded using the expression in Equation 4.2, then a more detailed diagram of the structure for the NLF is obtained, as shown in Figure 4.2.

![Figure 4.2: Non-Linear Filter (NLF) of SOBER-t32](image)

The second stage of the output function is the irregular decimation of the output of the NLF (in [58], this is referred to as stuttering). The sequence of keystream words \( Z^t \) is the irregular decimation of the sequence of words \( V^t \). This is intended to make correlation attacks infeasible. The ciphertext word \( C^t \) at time...
4.2. Algebraic analysis of SSS

$t$ is generated by XORing the generated keystream word $Z^t$ with the plaintext word $P^t$ as shown in Figure 4.1. The algebraic attack on SOBER-t32 discussed in Section 4.2.2 is applied to SOBER-t32 with the stuttering mechanism removed. Therefore, in this chapter the specific details of the stuttering mechanism are not described. For stuttering details, the reader is referred to the SOBER-t32 specification [58].

4.2.2 Algebraic attack on modified SOBER-t32

An algebraic attack on a modified version of SOBER-t32 is presented in [21]. The modification of SOBER-t32 removed the stuttering stage from the keystream generator. This attack assumes that an attacker can get the output of the NLF before stuttering. The approach taken is to regard the nonlinear function of the cipher as a combiner with memory, and to build indirect relationships among various bits of the cipher. The attack recovers the initial state of the LFSR. A summary of the results from [21] is presented below.

The first step of the attack is to construct equations for the NLF. These equations relate the input bits to the NLF with the output keystream bits indirectly, through a number of intermediate variables. Let $D_0^t$, $D_1^t$ represent the first two bits of the intermediate value $D$ which is the input word to the SBox and similarly let $y_0^t$, $y_1^t$ represent the first two bits of the intermediate value $y$ which is the output word of the SBox, as shown in Figure 4.2. These equations can be described as follows:

\[
\begin{align*}
D_0^t &= S[0]_0^t \oplus S[16]_0^t \\
D_1^t &= S[0]_1^t \oplus S[16]_1^t \oplus S[0]_0^tS[16]_0^t \\
y_0^t &= D_0^t \oplus S[1]_0^t \oplus S[6]_0^t \oplus S[13]_0^t \oplus K^t_0 \oplus V^t_0 \\
y_1^t &= D_1^t \oplus S[1]_1^t \oplus S[6]_1^t \oplus S[13]_1^t \oplus K^t_1 \oplus V^t_1 \oplus K^t_0S[13]_0^t \oplus (y_0^t \oplus D_0^t)(S[1]_0^t \oplus S[6]_0^t \oplus S[13]_0^t) \oplus S[1]_0^t(S[6]_0^t \oplus S[13]_0^t) \oplus S[6]_0^tS[13]_0^t \\
\end{align*}
\]

Then to find a valid relationship among the input bits and the output bits, two parts of the NLF, namely the SBox and the initial addition modulo $2^{32}$, are treated as a combiner with memory. The inputs to this combiner are the most significant bytes of the states $S[0]$ and $S[16]$ of the LFSR, since the most
significant byte of intermediate value \( D \) is input to the SBox as shown in Figure 4.2. The output is the output of SBox (labeled \( \mathcal{y} \) in Figure 4.3).

![Figure 4.3: Combiner structure of SOBER-t32](image)

The carry bit \( r \) is considered as an internal memory state of the combiner, storing the carry bit from the addition of the previous bits of the two states \( S[0]_{8\rightarrow15}, S[16]_{8\rightarrow15} \). Then, according to [6, 29], there is a multivariate relation that relates the inputs \( S[0]_{8\rightarrow15}, S[16]_{8\rightarrow15} \) and the outputs \( y_0, y_1 \) of the combiner without the carry bit \( r \). To find this relation, a matrix \( M \) is constructed for this combiner. There are 8 bits from \( S[0]_{8\rightarrow15} \), 8 bits from \( S[16]_{8\rightarrow15} \) and the carry bit \( r \). The rows of \( M \) represent all \( 2^{17} \) possibilities for the inputs and memory state bits. The columns represent all monomials up to degree \( k \) in the 16 input variables and first two bits of the output variables, \( y_0 \) and \( y_1 \). For a particular row, the entries are the specific values of these monomials given the values of the inputs of that row. The degree \( k \) is chosen as the lowest degree that satisfies the following condition:

\[
\sum_{i=0}^{k} \binom{18}{i} > 2^{17}
\]

The minimum degree that satisfies this condition is 9. Therefore, all product monomials that can be constructed for all degrees from 0 to 9 have to be con-
sidered. The number of columns of this matrix $M$ will be:

$$
\sum_{i=0}^{9} \binom{18}{i} \approx 2^{17} + 2^{14.5}
$$

Since the number of columns is greater than the number of rows, at least one column must be a linear combination of other columns. The linear dependencies can be efficiently found by Gaussian elimination. So, equations of up to degree 9 that relate the input and the output to the combiner without including the carry bit can be constructed. These equations are expressed in terms of the contents of the stages of the shift register and $y_0, y_1$. If $y_0$ and $y_1$ are replaced by their values in Equation 4.3, the equations will have degree at most 10 and contain only state bits of the LFSR, the constant K and the keystream bits.

This attack requires $2^{69}$ keystream bits and the complexity is around $2^{196.5}$ CPU clocks. This is significantly less than the complexity of exhaustive key search for SOBER-t32, as the key size is 256 bits.

### 4.2.3 Applying algebraic attack on SSS

The algebraic analysis on SSS is performed over GF(2), so each variable represents a bit. For SSS, a word is considered as 16 individual bits. The first stage of the algebraic attack is to build multivariate equations for the NLF. These equations relate the contents of the shift register stages with the keystream output bits.

**Constructing multivariate equations for NLF**

The NLF of SSS at time $t$ takes input words from five stages of the shift register to generate one word of keystream $Z^t$, as shown in Figure 3.6. The relation between the words of the shift register stages $R[0], R[16]$ and the intermediate word $X$ can be described at time $t$ by the following equation:

$$
X^t = R[0]^t + R[16]^t
$$

The first two bits of $X$, namely $x_0$ and $x_1$, can be obtained from the following equations which use binary addition modulo 2 (exclusive OR) instead of integer addition:
\[ x_0^t = R^t[0]_0 \oplus R^t[16]_0 \]
\[ x_1^t = R^t[0]_1 \oplus R^t[16]_1 \oplus R^t[0]_0 R^t[16]_0 \]

For the other bits \( x_i \) of \( X \), where \( 2 \leq i \leq 15 \), the following equation can be applied to convert the operation from modular addition to exclusive OR:

\[ x_i^t = R^t_i[0] \oplus R^t_i[16] \oplus R^t_{i-1}[0] R^t_{i-1}[16] \oplus \sum_{j=0}^{i-2} R^t_j[0] R^t_j[16] \left\{ \prod_{r=j+1}^{i-1} (R^t_r[0] \oplus R^t_r[16]) \right\} \]

From these equations, it is obvious that the degree of the terms \( x_i \) increases when \( i \) increases, and that the degree for each term will be \( i + 1 \). This amounts to equations of maximum degree at least 17 for SSS with 16-bit words, which may be too high for efficient solution of the resulting system.

Another approach is taken to determine relations between the inputs and the outputs of the SBox. Referring to Figure 3.6, we consider the relations between the outputs of the first and the second SBox, which are \( \alpha \) and \( \beta \) respectively, the keystream output \( Z \), and several register states, as follows:

\[ Z_8^t = x_0^t \oplus R[1]_0^t \oplus R[6]_0^t \oplus R[13]_0^t \oplus \beta_8 \oplus R[0]_8^t \oplus \alpha_0^t \]
\[ Z_9^t = x_1^t \oplus R[1]_1^t \oplus R[6]_1^t \oplus R[13]_1^t \oplus \beta_9 \oplus R[0]_9^t \oplus \alpha_1^t \oplus \]
\[ (\alpha_0^t \oplus x_0^t)(R[1]_0^t \oplus R[6]_0^t \oplus R[13]_0^t) \oplus R[1]_0^t (R[6]_0^t \oplus R[13]_0^t) \oplus R[6]_0^t R[13]_0^t \]

Other relations can be built as follows. The modulo 2^{16} addition of two 16-bit words \( X = R[0] + R[16] \) can be considered as two separate additions of 8-bit words. The first addition is the addition of the least significant bytes (the bits 0 \( \rightarrow \) 7) of stages \( R[0] \) and \( R[16] \). The second addition are the most significant bytes of stages \( R[0] \) and \( R[16] \). This is used as an input to the SBox. In the second addition there will be a carry \( s \) at the 8-th position from the least significant bytes addition which should be considered in this addition process. So the two additions can be described as follows:
4.2. Algebraic analysis of SSS

\[
x_{0\rightarrow7} = (R[0]_{0\rightarrow7} + R[16]_{0\rightarrow7}) \mod 2^8
\]
\[
x_{8\rightarrow15} = (R[0]_{8\rightarrow15} + R[16]_{8\rightarrow15} + s) \mod 2^8
\]

Constructing the SSS combiner

The NLF of SSS is too complex for the output keystream to be modeled directly as functions of register states. However, indirect relationships may be computed in a similar manner to those formed in the algebraic attack on SOBER-t32 by treating the NLF as a combiner with memory. The combiner for SSS is shown in Figure 4.4. It can be observed that it is not possible to determine bits \( \alpha_0, \alpha_1 \) given the inputs, since the SBox is key-dependent. Therefore, this combiner cannot be used to find valid relations for SSS.

\[
\begin{align*}
\text{s}(1\text{-bit}) \\
R[0]_{8\rightarrow15} \quad R[16]_{8\rightarrow15} \\
\quad \quad SBox \\
\quad \quad \alpha (16\text{ bits}) | \alpha_1 | \alpha_0
\end{align*}
\]

Figure 4.4: Combiner structure of SSS

Following the discussion from [29] for key-dependent combiners, the unknown output bits obtained from the use of the key-dependent SBox can be treated as memory bits in the combiner. This means that these unknown bits would not appear in the relations obtained from the combiner. Recall that the SBox is used twice in the NLF of SSS.

The combiner is extended to cover the entire NLF, where the output is the keystream bit \( Z_8 \), described in Equation 4.4. Let \( \alpha \) and \( \beta \) represent the intermediate values obtained as the output of the first SBox and second SBox, respectively. The \( Z_8 \) keystream bit depends on \( \beta_8, R[0]_{8}, R[13]_{0}, R[6]_{0}, R[1]_{0}, \)
\( \alpha_0, R[0]_0 \) and \( R[16]_0 \). As mentioned before, bits of \( \alpha \) and \( \beta \) can be treated as memory bits. In total there would be 6 bits of input, one bit output, and 2 memory bits. This combiner is shown in Figure 4.5.

![Figure 4.5: Possible valid combiner structure for NLF of SSS](image)

If the matrix \( M \) is constructed for this combiner, it is necessary to have \( 2^6+2 \) rows in \( M \) to cover all possibilities for the input bits and memory bits. On the other hand, there are only 7 variables from the input bits and the output bit for the columns. To guarantee a valid relation between the input bits and the output bits, the following inequality must be satisfied [6,29]:

\[
\sum_{i=0}^{k} \binom{k + m}{i} > 2^{k+c}
\]

In this case the inequality is as follows:

\[
\sum_{i=0}^{k} \binom{7}{i} > 2^8
\]

This inequality cannot be satisfied, since the maximum value for the binomial sum is \( 2^7 \). This means that valid relations cannot be guaranteed, since there will always be more rows than columns in the matrix \( M \). Nevertheless, it may still be possible to find valid relations if the rank of \( M \) happens to be lower than the number of columns, but this occurs with a very low probability.
Since the matrix $M$ is of a practical size, it can be constructed using Magma 2.12. All possible monomials up to the maximum degree 7 are used, giving $2^7$ columns, which is the highest possible for the 7 bits of chosen inputs and outputs in the combiner. The resulting matrix $M$ has rank $2^7$, which means that it has trivial nullspace, and so no linear dependencies exist among the columns. Therefore, it is not possible to obtain a valid relation for this combiner.

**Algebraic analysis assuming valid relations exist**

It is clear that a valid relation cannot be obtained by applying the combiner method. However, suppose that another method could be used to find these relations for this specific NLF. Such a method may not be systematic and may be difficult to discover, and is left an open problem for future research. However, we continue the analysis of SSS under the assumption that a method for forming these relations exists.

Assume that a valid relation can actually be found for the NLF among the register states and the output keystream using an alternative method, despite the difficulties discussed previously. This means there is an equation

$$F(R[0], R[16], \ldots, Z_8) = 0$$

of maximum degree $k$ in 7 variables that is valid for all possible register input bits and keystream output bits. In the worst case, the value of $k$ is 7. The relation can then be used to generate a system of equations by evaluating the relation for a series of time steps. Under a known plaintext scenario, the keystream bits $Z_t^j$ are assumed to be known at all times $t$, and these values can be substituted into the relevant equations.

If the underlying register is linear, as it is in the case of SOBER-t32, the register state bits $R[i]$ would be linear combinations of the initial state bits of the shift register. These linear combinations of initial state bits can simply be substituted into the equations relating the register contents to the keystream. Clocking the register does not increase the degree of the equations relating the contents of the register stages to the initial values, they remain linear. The resulting equation system would have variables representing all bits of the initial states of the shift register, and it would be of maximum degree $k \leq 7$, since all substitutions made have been linear. Solving the system means recovering these
initial states.

In the case of SSS, some stages of the shift register are updated nonlinearly, so it is not possible to use only linear combinations of the initial values of the register states to express the contents of the register stages at later times. Furthermore, the updated register contents contain values obtained from the key-dependent SBox, so it is not possible to determine the register states for successive clocks without knowledge of either the key or the SBox or without introducing an excessive number of new variables. This state update process prevents the generation of a set of equations from the relations found for the NLF, and the algebraic attack fails.

### 4.2.4 Algebraic analysis on modified SSS

This section analyses a modified version of SSS by considering the application of the algebraic attack outlined in Section 4.2.3. In this section, modifications to the design of SSS are considered as a theoretical exercise to determine how big a design change is necessary in order for this type of attack to succeed. This provides some indication of the robustness of the design to this form of attack. The modifications involve altering the state update function, so that the shift register is updated linearly, using a slightly modified version of the feedback function taken from the shift register in SOBER-t16 [57]. Also, the modification consists of XORing the ciphertext word with the feedback word from the shift register in updating the register contents, as shown in Figure 4.6. Note that the cipher remains self-synchronous.

For this modified version of SSS, we attempt to develop a system of equations for use in an algebraic attack. Since the register states are updated linearly, each state at time \( t \) can be represented as linear expressions in the initial state bits. Therefore, the system has maximum degree \( k \leq 7 \). Solving this system of equations permits the recovery of the initial states of the shift register. In the worst case, the equations are of degree up to 7 as mentioned in Section 4.2.3. There are 272 unknowns in this system representing the initial state of the 16 bits in each of the 17 stages of the shift register. This system can be solved by the XL algorithm [27].
4.2. Algebraic analysis of SSS

Figure 4.6: Modified structure of SSS keystream generator

The number of monomials $T$ in this system is given by:

$$T = \sum_{i=0}^{7} \binom{272}{i} \approx 2^{44}$$

Therefore, we need to generate at least $T$ equations for all initial states to be recovered. We also need to observe $2^{44}$ observations of the bit $Z_8$ from a single key, which would be obtained from at least $2^{44}$ keystream words generated from the key. This keystream requirement falls within the maximum of $2^{80}$ words of keystream output allowed for a single key IV pair in the SSS specification [55].

The complexity of this attack using the XL algorithm would be:

$$(7/64) \cdot T^{\log_2 T} \approx 2^{121}$$

Note that the complexity of this algebraic attack on the modified SSS is less than $2^{128}$ the complexity for exhaustive key search of SSS. This shows that modifying the register so that the feedback function is linear does allow the algebraic attack to be launched, under the condition that a method for forming valid relations for the NLF can be found. Although the attack can be successful on this modified version of SSS under these conditions, we note that the modification is substantial and the attack can not be successfully applied to the unmodified version. Hence the SSS design is reasonably robust against this form of attack.
Chapter 4. Algebraic analysis of SSS and ZUC stream ciphers

4.3 Algebraic analysis of ZUC

The ZUC [39] stream cipher was evaluated by two different teams [24, 67] before it was released for public evaluation. The two teams evaluated ZUC algebraically, using the direct method to obtain a system of equations. In this section, a different approach is applied on ZUC to establish relations between initial state bits and the output keystream bits. This approach is similar to that applied to SSS, using the combiner method and considering some parts of the NLF as a combiner. Note that the description of ZUC keystream generator was given in Section 3.3.2.

4.3.1 Existing algebraic analysis of ZUC

In this section the work in [24, 67] to determine the algebraic relations between the initial state bits and the output keystream bits directly is reviewed. From the description of the ZUC stream cipher, the main operations in the structure of ZUC are: addition and multiplication in the finite field $F_{p}$, where $p = 2^{31} - 1$, addition in the ring $Z_{2^{32}}$, and word XORing. Besides these, the cipher also makes use of the SBox and linear transformations on the bit level.

The total internal state of ZUC is 560 bits, and the cipher generates at each clock 32 bits of output. Each output bit raises an equation between the input internal state and the output keystream bits. To recover the secret internal state bits using algebraic attacks, a system of equations should be developed. The minimum keystream requirement to construct this system will be 18 keystream words ($\lceil 560/32 \rceil = 18$) [24]. The analysis starts by constructing equations over $F_{2}$, (then relates modular addition to the XOR operation to establish a linear relation in the NLF of ZUC).

Constructing equations over $F_{2}$

From the specification of the ZUC stream cipher and the details shown in Figure 3.7, we can see that the update function of the LFSR involves five stages and raises algebraic equations of degree 6 to update the state $S[16]$, since the state $S[0]$ appears twice. This state ($S[16]$) is input to the nonlinear function $f$ through the variable $X_{1}$ after four clocks. There are two nonlinear operations in the nonlinear function: addition modulo $2^{32}$ and the application of the SBox.

If $S[16]$ is eliminated from the system of equation, then the degree of equations
of the whole system will increase to 8. The number of variables $T$ in the system of equations to recover the secret states under the assumption of observing only 18 words of keystream will be $[24]$: 

$$T = 16 \times 31 + 18 \times 2 \times 32 = 1648$$

This system of equations can be solved using the linearisation method with complexity of:

$$\binom{T}{8}^{2.37} \approx 2^{166}$$

If the output of the LFSR feedback computation is considered as a new set of variables, and the algebraic equations for the LFSR states are of degree 6, then the total number of variables $T$ under the assumption that only 18 words of keystream will be used in this system will be $[67]$: 

$$T = 16 \times 31 + 17 \times 31 + 18 \times 2 \times 32 = 2175$$

The complexity of solving this system using linearisation method will be:

$$\binom{T}{6}^{2.37} \approx 2^{135}$$

New variables can be introduced and a system of equation of algebraic degree 2 can be developed for the states $[24]$. Then the computation of LFSR feedback will be for $p = 2^{31} - 1$ as follows:

$$y_1 = (1 + 2^8) s_0 \mod p$$
$$y_2 = 2^{20} s_4 + y_1 \mod p$$
$$y_3 = 2^{21} s_{10} + y_2 \mod p$$
$$y_4 = 2^{17} s_{13} + y_3 \mod p$$
$$S[16] = 2^{15} s_{15} + y_4 \mod p$$

From the above equations, 93 independent algebraic equations can be generated from each equation of degree 2. Thus, the total number of independent algebraic equations of degree 2 that can be generated from the LFSR feedback computa-
tion will be $93 \times 5 = 465$. For the nonlinear function $f$, it is possible to generate algebraic equations of degree 2. It can be established that 93 independent algebraic equations from the modular addition operation and the SBoxes $S_0$ and $S_1$ can generate 11 and 39 independent equation of degree 2 respectively. Thus, the ZUC stream cipher can form a system of algebraic equation of degree 2 and the number of variables that can be established under the assumption of observing only 18 words of keystream will be:

$$T = 16 \times 31 + 2 \times 32 - 1 + 17(5 \times 31 + 3 \times 32 - 2) = 4792$$

The number of equations $m$ of degree 2 will be:

$$m = 93 + 17(93 \times 5 + 2 \times 93 + 39 + 11) = 12010$$

This system can be solved using XL algorithm, since the linearisation method is not effective, and the estimated complexity using XL algorithm will be:

$$\left( \frac{T}{T^{\frac{2.37}{m}}} \right) \approx 2^{830}$$

Practically, the number of independent equations is much smaller than the number of variables if the linearisation method is used, and hence the keystream word requirement to solve the system is more than 18. So this will increase the number of intermediate variables and also will increase the complexity of solving the system. Thus solving the equations in this way is far worse than exhaustive key search which costs $2^{128}$ operations.

### 4.3.2 New algebraic analysis of ZUC

The previous algebraic approach presented in Section 4.3.1 is straightforward and involves finding direct relationships between the internal state bits of ZUC and the observed keystream bits. In this section, a different approach is taken to establish relationships between the internal state bits and the output keystream bits indirectly. This analysis involves considering the nonlinear filter as a combiner with four input words and two output words. The input words are two input words from the LFSR, and two inputs words from the internal memory states. It is possible to find these indirect relations through the use of a com-
Valid relations can be found by using a matrix to represent all possible states of the combiner. The rows of the matrix represent the input bits of the combiner and the memory bits if any. The columns represent all the monomials in the variables of the inputs and outputs of the combiner, up to certain degree without the memory bits. Row reduction is then performed on the matrix to find linear dependencies, which corresponds to valid relations among the inputs, outputs and memory bits. Then, these relations are evaluated for $T$ clocks to give a system of multivariate equations with the initial state bits as part of the variables.

The analysis starts by constructing multivariate equations for the Non-Linear Function NLF of ZUC. Then a combiner for the NLF is built. Finally, this analysis is applied to a modified version of ZUC.

**Constructing multivariate equations for NLF**

During the keystream generation, the NLF of ZUC at time $t$ takes four 32-bit words from different stages of the shift register to generate one 32-bit word of keystream $Z^t$, as shown in Figure 3.7. Two words $X_1$ and $X_2$ are inputs to the nonlinear function $f$. The output of $f$ is the two words $R_1$ and $R_2$ which are used with the other two words $X_0$ and $X_3$ from the input stages to generate the output keystream word $Z^t$ as shown later in Equation 4.6.

The analysis of the NLF begins by considering it as a combiner enclosed with dash line shown in Figure 4.7. The aim of this combiner is to find relationships between the input bits from the states of the shift register and the output keystream bits. There are four input words to the combiner which are from the states of shift register and one output word which is the keystream word.

To find a valid relation, a matrix for the combiner is constructed. The rows of the matrix represent the input bits of the words $X_0^t$, $X_1^t$, $X_2^t$ and $X_3^t$. The columns represent all monomials up to degree $d$ in the 128 bits of state input and the 32 bits of the output word $Z^t$. The number of columns must be larger than the number of rows to ensure linear dependencies in the matrix. Let $M$ represent the number of consecutive steps of forming the combiner, $k$, $m$, and $c$ represent number of the input bits, the output bits and the memory bits, respectively.
Chapter 4. Algebraic analysis of SSS and ZUC stream ciphers

Equation 4.5 gives the minimum value of degree $d$.

$$2^{Mm} \cdot \sum_{i=0}^{d} \left( \frac{Mk}{i} \right) > 2^{Mk+c} \quad (4.5)$$

The minimum degree $d$ that satisfies Equation 4.5 is 30, for $M = 1$. The value of the degree is very high and therefore this approach is not practical. An alternate combiner has to be investigated to minimize the degree of the equations.

**Constructing the ZUC combiner**

The nonlinear function $f$ is considered to be a combiner with memory. This combiner has two words $X_1^i$ and $X_2^i$ of inputs, two words $R_1^{t-1}$ and $R_2^{t-1}$ of memory and two words $R_1^t$ and $R_2^t$ of output, as shown in Figure 4.8.

To find a valid relation, a matrix for the combiner is constructed. The rows of the matrix represent the input bits of the words $X_1^i$ and $X_2^i$ and the memory bits of the words $R_1^{t-1}$ and $R_2^{t-1}$. The columns represent all the monomials in the variables of the inputs and outputs, up to certain degree without the memory bits. The problem here is that the memory bits are the same as the output bits.
4.3. Algebraic analysis of ZUC

If the memory bits are eliminated by using the output of the matrix relations then in this case the output bits are also eliminated, which are required to find a valid relation between the input bits and the output bits. This type of combiner also does not work due to the use of the memory bits as the output bits.

To deal with this problem, the memory words will be considered as input to the combiner. In this case there are four input words \( X_1^t, X_2^t, R_1^{t-1}, R_2^{t-1} \) to the combiner, and two output words \( R_1^t, R_2^t \). This combiner is as shown in Figure 4.9.

To find a valid relation between the input bits and the output bits of the combiner, a matrix \( A \) is constructed. The \( 2^{128} \) rows represent the set of all possible inputs to the combiner. The columns represent all monomials up to degree \( d \) in the 128-bits of register inputs and the 64 bits of the output. To find the minimum possible value of \( d \), Equation 4.5 is used.

The minimum degree \( d \) that satisfies Equation 4.5 is 16 for \( M = 1 \). So all product monomials that can be constructed for all degrees from 0 to 16 have to be considered. The number of columns of this matrix will be:

\[
2^{64} \cdot \sum_{i=0}^{16} \binom{128}{i} \approx 2^{131} > 2^{128}
\]

Since the number of columns is greater than the number of rows, at least one column must be a linear combination of other columns. The linear dependencies can be found by Gaussian elimination [95]. Therefore, equations of up to degree
16 can be constructed that relate the input bits and the output bits to the combiner. The time complexity of this precomputation can be as high as

\[(2^{128})^{\log_2 7} \approx 2^{357}.
\]

In reality, the effort may be much less due to the sparsity of the matrix.

The output equations relate the input words \(X_1^t, X_2^t, R_1^{t-1}, R_2^{t-1}\) to the output words \(R_1^t, R_2^t\) of the combiner, expressed in terms of the initial states. Then, Equation 4.6 is used to obtain the direct relationship between the input words and the output keystream words.

\[Z^t = ((X_0^t \oplus R[1]^t) + R[2]^t) \oplus X_3^t \tag{4.6}\]

There are 560 unknowns in this system representing the initial states of the 31
4.3. Algebraic analysis of ZUC

bits in each of the 16-stages of the shift register and the two memory words \(R_1\) and \(R_2\). An attack is considered significant if it require less than \(2^{560}\) operations. The number of monomials \(T\) in this system is given by:

\[
T = \sum_{i=0}^{16} \binom{560}{i} \approx 2^{102}
\]

Therefore, at least \(T\) equations need to be generated for all initial states to be recovered. This means that \(2^{102}\) keystream bits, or \(2^{97}\) word observations from a single key is required for a 64 bit processor. The complexity of this attack using the XL algorithm \cite{27} is.

\[
C = \left(\frac{7}{64}\right) \cdot T^{\log_2 7} \approx 2^{282}
\]

The complexity of this approach is much less than the complexity of the existing result described in Section 4.3.1 of \(2^{830}\). However, both results are far worse than the time complexity of exhaustive key search which cost \(2^{128}\) operations. This result combined with the previous investigations into algebraic attacks in \cite{24,67} indicates that the ZUC stream cipher seems secure from these forms of algebraic attacks.

4.3.3 Algebraic analysis of modified ZUC

In this section a modified version of ZUC is analysed. The modifications to the design of ZUC are considered as a theoretical exercise to determine how big a design change is necessary in order for this type of attack to succeed. This provides some indication of the robustness of the design to this form of attack. This modified version uses the same inputs and components, but the amount of output is increased. Let \(m\) denote the number of output bits produced at each clock.

The modified version can be realised in different ways, for example, by considering the output of each SBox as an output word after XORing it by one of the free words of the bit reorganisation word. The analysis is repeated for different values of \(m\) and then the minimum degree \(d\) that satisfies Equation 4.5, the keystream requirement and the complexity of the attack are calculated.

Table 4.1 shows the time complexities of attacks on the modified version of the ZUC stream cipher. In Table 4.1, Time (Step 1) represents the time required
to find valid relations. Time (Step 2) represents the time required to solve these equations using the XL algorithm.

<table>
<thead>
<tr>
<th>$m$</th>
<th>32</th>
<th>64</th>
<th>65</th>
<th>80</th>
<th>100</th>
<th>120</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d$</td>
<td>30</td>
<td>16</td>
<td>15</td>
<td>11</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>Keystream bits</td>
<td>$2^{162}$</td>
<td>$2^{95}$</td>
<td>$2^{74}$</td>
<td>$2^{38}$</td>
<td>$2^{17}$</td>
<td></td>
</tr>
<tr>
<td>Time (Step 1)</td>
<td>$2^{457}$</td>
<td>$2^{357}$</td>
<td>$2^{357}$</td>
<td>$2^{357}$</td>
<td>$2^{357}$</td>
<td></td>
</tr>
<tr>
<td>Time (Step 2)</td>
<td>$2^{653}$</td>
<td>$2^{278}$</td>
<td>$2^{206}$</td>
<td>$2^{106}$</td>
<td>$2^{47}$</td>
<td></td>
</tr>
</tbody>
</table>

It can be observed that the time required to find relations is the same for all numbers of output bits $m$. However, the number of keystream bits required and the time required to solve the equations both decrease when the number of output bits $m$ increased. This means that the security of the ZUC cipher would decrease when the number of output bits are increased, for reasons such as increasing the speed of the cipher.

If these results given in Table 4.1 are compared with the results of the attack on Snow 2.0 presented in [29], then it is clear that the two results are similar. This means that the ZUC and Snow cipher [36] designs are secure for the designed throughput rate, which is resistant to this type of algebraic attack. Higher throughput could put the security of these ciphers at risk. Although the attack can be successful on this modified version of ZUC under these conditions, we note that the modification is substantial and the attack can not be successfully applied to the unmodified version. Hence the ZUC design is reasonably robust against this form of attack.

4.4 Summary and conclusion

Investigations on the resistance of the confidentiality mechanism used in several AE ciphers to a form of attacks known as algebraic attacks were given in this chapter. Algebraic attacks are applied for two AE stream ciphers which are SSS and ZUC. Both of the ciphers examined have a similar structure, and both also have similarities to another stream cipher which was previously shown to be vulnerable to a form of algebraic attack.

The algebraic attack on SOBER-t32 [21] viewed the NLF as a combiner with two output bits ($\alpha_0$ and $\alpha_1$) and only one memory bit (the carry bit). The
matrix constructed for this combiner had a greater number of columns than
rows, so there exist multivariate equations that relate the input and the output
of the combiner without the memory bit.

As the SSS cipher has a similar structure, we reasoned that a similar approach
may permit an algebraic attack on SSS. We considered the NLF of SSS as a
combiner. However, the combiner constructed for the NLF of SSS has a greater
number of memory bits than output bits. This is due to the fact that the SBox is
key-dependent and is used twice in the NLF of SSS. As shown in Section [4.2.3] to
construct a combiner for one bit of keystream output from the NLF, two memory
bits are required. If more output bits in the combiner were be included, then
at the same time we would have to include more memory bits that correspond
to the output bits (the number of memory bits is twice the number of output
bits). Therefore, when constructing a matrix for this combiner, there will always
be more rows than columns. This means that linear dependencies cannot be
guaranteed, so finding a valid relation for the NLF using this matrix method is
unlikely. For a combiner with one bit of output, it is shown that it is not possible
to obtain a valid relation because the matrix is of full rank.

This result supports the claim by the SSS authors in their eSTREAM sub-
mission document [55] that SSS is secure from algebraic attack. The use of a
key-dependent SBox results in the relations describing the NLF being unknown,
which prevents valid relations being found among the register states and the
keystream outputs using the combiner method. The situation is further com-
pounded by the fact that the SBox is used twice in the NLF, which doubles
the number of memory bits needed for the combiner. This renders a successful
algebraic attack on SSS using this strategy very improbable, if not infeasible.

The conditional analysis shows that the use of a nonlinear shift register up-
date function also contributes to the resistance of SSS to algebraic attacks. Even
if valid relations of low degree could be found for the NLF, the degree of such
equations will sharply increase during the equation generation stage. Finally, a
modified version of SSS is considered such that the shift register state update
function is linear. It shows that this makes solving the resulting system of equa-
tions feasible. This algebraic method can be launched successfully on SSS if
both a valid equation can be formed for the NLF and the state update function
is changed to a linear function. In such a case the complexity of the attack is
estimated to be $2^{121}$ operations and to require $2^{44}$ keystream bits.
Using a key-dependent SBox in the NLF contributes to the resistance of SSS against algebraic attacks using the combiner method. The use of a nonlinear state update function (which also makes use of the key-dependent SBox for the shift register) also increases the resistance against this type of attack. This research indicates that the use of a key-dependent SBox may be a worthwhile strategy in designing secure keystream generators for stream ciphers.

The same method was applied in an algebraic analysis of the ZUC stream cipher. An existing method finds direct relations between the input and the output keystream, but has been shown to be infeasible. The method presented in this chapter uses an indirect method and considers the keystream generator as a combiner with memory in order to form the equations relating the input bits to the output bits. One of the possibilities considered, led to equations for the nonlinear function $f$ of the NLF. Although the complexities of both methods are worse than exhaustive key search, the complexity of the new method is much less than that of the previous method.

The algebraic analysis is also applied to a modified version of the ZUC stream cipher, where more output words are produced at each clocking. This increases the throughput of the cipher, but the analysis shows that the security of the cipher would be greatly reduced as a result. This result is in line with the result that was previously obtained for the modified version of Snow 2.0 in [29].

In summary, these results indicate that both the SSS and ZUC stream ciphers as described in the literature are secure from this type of algebraic attack since the complexity for attack is worse than exhaustive key search attack. However, modifications to either cipher could leave them vulnerable to algebraic attacks.
Chapter 5

General model for MAC generation using direct injection

In this chapter, the security properties of the integrity mechanism are investigated, for a specific method of message injection. As discussed in Chapter 3, there are two different ways in which a message is commonly incorporated into the integrity component of a cipher when generating a MAC tag, namely direct injection and indirect injection. This chapter describes a general matrix based model for MAC generation using the input message directly. We developed this model by extending the concept used by Nakano et al. [76] in constructing their hash function model. We consider the calculation of a MAC tag for a message $M$ of length $l$. That is $M_l = m_0m_1 \ldots m_{l-1}$. The model examines two different options for injecting the input message directly into the internal state of the cipher. Several current stream ciphers which can be considered as instances of this model will be presented and analysed in this chapter.

The chapter is organised as follows. Section 5.1 presents a matrix based representation of a general model for generating a MAC tag using stream ciphers with direct message injection. In Section 5.2, we extend the bit based model of Nakano et al. for generating a hash value to the generation of MAC tags, and further extend to the case where word based registers are used. Three specific stream cipher algorithms that generate a MAC tag using this method of injection (SSS [55], NLSv2 [56] and SOBER-128 [54]) are examined in Section 5.3. A security analysis of this model in relation to forgery is presented in Section 5.4.
Summary and concluding remarks are presented in Section 5.5.

5.1 General model for MAC generation using direct injection

We consider the case of injecting a message directly into the state of a keystream generator. For simplicity, we use a nonlinear filter generator which consists of one linear feedback shift register (LFSR) and a nonlinear filter function, as the state is contained in a single register. We assume the components of this nonlinear filter generator are selected for their security properties. That is, the feedback polynomial of the LFSR is a primitive polynomial and the filter function is nonlinear, balanced and with high correlation immunity. This provides resistance to various types of attacks on stream ciphers.

MAC generation is performed in three phases. In the first phase, the message is prepared by padding with any value which is predefined in the specification of the cipher. This process may be performed to avoid any insertion forgery attacks, or it could be performed in order to match the length of the message with the size of each stage in the internal state. Also in this phase, the internal state of the integrity component of the cipher is initialised using a keystream sequence which is generated using a secret key $K$ and optional public $IV$. In the second phase, the message is accumulated into the state of the LFSR using the accumulation function. In this chapter we will consider accumulation using direct message injection. After the whole message has been processed, the final phase occurs: the contents of the LFSR are processed using a finalisation function to form a MAC tag for the input message.

In this section, a general model for the second phase in MAC generation is proposed. This model describes methods for accumulating the input message directly into the internal state of the stream ciphers to generate a MAC tag. This model does not include the other two phases of the MAC generation. During the message accumulation phase the feedback of the LFSR (denoted by $y_t$) and the output of the nonlinear filter (denoted by $z_t$) may also be accumulated simultaneously with the input message (denoted by $m_t$) into the internal state of the LFSR, as shown in Figure 5.1.
5.1. General model for MAC generation using direct injection

5.1.1 State update for single step

We start by considering the case of a bit based (binary) shift register, where each register stage holds a single bit and the state update function processes a single message bit at each iteration. Let $A$ denote the LFSR register with $d$ stages, $A = (a[0], a[1], \ldots, a[d-1])$, as shown in Figure 5.1. All operations are performed over GF(2). At time $t$, there are three possible inputs to each stage in the register $A$: the input message bit $m_t$, the nonlinear filter output bit $z_t$ and either the feedback bit $y_t$ for stage $a[d-1]$ or the content of stage $a[i+1]$ for stage $a[i]$, where $0 \leq i < d - 2$. These three inputs can be fed into the register either by XORing $m_t$ and $z_t$ with either $y_t$ or $a_t[i+1]$, or by replacing the content of the relevant stage with the chosen combination of $m_t$ and $z_t$. Our model incorporates both these alternatives, as discussed below.

Let the feedback function $f_b$ of the accumulator register $A$ shown in Figure 5.1 be described by the polynomial, $f_b(a) = c_0a[0] \oplus c_1a[1] \oplus \cdots \oplus c_{d-1}a[d-1]$, where $c_i \in \{0, 1\}$, for $0 \leq i \leq d - 1$, indicates which stage of the register is present or absent in the feedback connection. If we represent the contents of

![Figure 5.1: General model for MAC generation using the input message directly](image-url)
register $A$ at time $t$ by the vector:

$$A_t = \begin{pmatrix} a_t[0] \\ a_t[1] \\ \vdots \\ a_t[d-1] \end{pmatrix},$$

then the contents of register $A$ at time $t + 1$ can be represented by the equation:

$$A_{t+1} = CA_t \oplus m_t \sigma_m \oplus z_t \sigma_z$$

(5.1)

where $C$ describes the feedback and shifting behavior of the LFSR and $\sigma_m$ and $\sigma_z$ are vectors of zeroes and ones indicating which stages of the register the message bit and the output filter bit, respectively, are injected into. More specifically,

$$\sigma_{m,i} = \begin{cases} 0, & m_t \text{ is not injected into stage } i, \\ 1, & m_t \text{ is injected into stage } i \end{cases}$$

and similarly

$$\sigma_{z,i} = \begin{cases} 0, & z_t \text{ is not injected into stage } i, \\ 1, & z_t \text{ is injected into stage } i. \end{cases}$$

As mentioned, $C$ is a $(d \times d)$ matrix describing the update behavior of the register, and its form depends on the method of insertion that is used.

**For insertion using XOR,** $C$ is the usual companion matrix of the register $A$, which can be described as follows:

$$C = \begin{pmatrix} 0 & 1 & 0 & \ldots & 0 & 0 \\ 0 & 0 & 1 & \ldots & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \ldots & \ldots & 0 & 1 & 0 \\ 0 & \ldots & \ldots & 0 & 0 & 1 \\ c_0 & c_1 & \ldots & \ldots & c_{d-2} & c_{d-1} \end{pmatrix}$$

(5.2)

**For insertion using replacement,** $C$ is similar to a normal companion matrix, but with some rows modified as follows: if the contents of stage $i$ are replaced by the inserted message bit and/or filter output bit (without combining
them with \( a[i+1] \), then all entries in row \( i \) of \( C \) are set to zero. Thus \( C \) takes the form:

\[
C = \begin{pmatrix}
0 & 1 & 0 & \ldots & \ldots & \ldots & \ldots & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
0 & \ldots & 0 & 1 & 0 & \ldots & \ldots & 0 \\
0 & \ldots & \ldots & 0 & 0 & \ldots & \ldots & 0 \\
0 & \ldots & \ldots & 0 & 1 & 0 & \ldots & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
0 & \ldots & \ldots & \ldots & \ldots & \ldots & 0 & 1 \\
c_0 & c_1 & \ldots & \ldots & \ldots & \ldots & \ldots & c_{d-2} & c_{d-1}
\end{pmatrix} \leftarrow (i) \tag{5.3}
\]

where the label \( i \) is indicated on the right of \( C \) to indicate that the row that has been modified. If the value of the input message and/or filter output replaces the contents of more than one stage of register \( A \), then the rows corresponding to these stages in matrix \( C \) are set to zero in the same manner as for the stage \( i \) in Equation 5.3.

### 5.1.2 Model for complete accumulation phase

Now let us consider Equation 5.1 at the end of the accumulation process. The contents of register \( A \) at time \( t = l \), where \( l \) is the length of the input message, can be represented in terms of the previous contents of register \( A \), previous bits of the input message and previous bits of the filter output. Our aim is to relate \( A_l \) to \( A_0 \) and to the full contents of the message and filter output. We build up the final equation in stages, considering successively the effects of the various components in Equation 5.1.

**Autonomous behavior of LFSR feedback**

If we ignore the input message and the filter output injection, and consider only the update feedback injection \( y_t \) into register \( A \) in Figure 5.1, then we have the following:

\[
A_l = CA_{l-1} = C(CA_{l-2}) = C(C(C(A_{l-3}))) \cdots .
\]

By induction, we have

\[
A_l = C^l A_0.
\]

**Message injection**
Now consider injecting the input message bit $m_t$ into the stages of register $A$ determined by $\sigma_m$ as shown by the dashed line in Figure 5.1, in addition to the feedback bit $y_t$ (as shown by the solid line in Figure 5.1). Then, the contents of register $A$ at time $t = l$ are

$$A_l = CA_{l-1} \oplus m_{l-1}\sigma_m$$
$$= C(CA_{l-2} \oplus m_{l-2}\sigma_m) \oplus m_{l-1}\sigma_m$$
$$= C^2A_{l-2} \oplus m_{l-2}C\sigma_m \oplus m_{l-1}\sigma_m$$

By induction,

$$A_l = C^l A_0 \oplus m_0 C^{l-1}\sigma_m \oplus m_1 C^{l-2}\sigma_m \oplus \cdots \oplus m_{l-2}C\sigma_m \oplus m_{l-1}\sigma_m$$
$$= C^l A_0 \oplus K_m M_{l-1}$$

where

$$K_m = [ C^{l-1}\sigma_m \ C^{l-2}\sigma_m \ C^{l-3}\sigma_m \ \cdots \ C\sigma_m \ \sigma_m ],$$

and

$$M_{l-1} = \begin{pmatrix} m_0 \\ m_1 \\ \vdots \\ m_{l-1} \end{pmatrix}.$$

**Filter output injection**

Similar to the message injection, if the filter output is also injected into the stages of register $A$ determined by $\sigma_z$, as shown by the dotted line in Figure 5.1, then the contents of register $A$ at time $t = l$ will be:

$$A_l = C^l A_0 \oplus K_m M_{l-1} \oplus K_z Z_{l-1}$$

(5.4)

where

$$K_z = [ C^{l-1}\sigma_z \ C^{l-2}\sigma_z \ C^{l-3}\sigma_z \ \cdots \ C\sigma_z \ \sigma_z ],$$

and

$$Z_{l-1} = \begin{pmatrix} z_0 \\ z_1 \\ \vdots \\ z_{l-1} \end{pmatrix}.$$
5.1.3 Form of matrices for alternative insertion methods

As noted previously, both the message bit and filter output bit can be injected into the stages of the register in one of two ways: XORing with the feedback to that stage or replacing the previous contents of the injected stages. We examine the model for each of these two options.

**Insertion with XOR**

For this method, the matrix $C$ is the companion matrix as shown in Equation 5.2. Then for $C^2$, the matrix will be:

$$
C^2 = \begin{pmatrix}
0 & 0 & 1 & 0 & 0 & \cdots & 0 \\
0 & 0 & 0 & 1 & 0 & \cdots & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\
0 & \cdots & \cdots & 0 & 0 & 0 & 1 \\
c_0 & c_1 & \ldots & \ldots & c_{d-3} & c_{d-2} & c_{d-1}
\end{pmatrix}
$$

where $c^T = \begin{pmatrix} c_0 & c_1 & \ldots & c_{d-1} \end{pmatrix}$, and we see in general that $C^{t+1}$ contains the last $d - 1$ rows of $C^t$ shifted up by one row followed by a new row $c^T C^t$, i.e. if

$$
C^t = \begin{pmatrix}
 r_1^T \\
r_2^T \\
\vdots \\
r_{d-1}^T \\
r_d^T
\end{pmatrix}, \text{ then } C^{t+1} = \begin{pmatrix}
r_2^T \\
r_3^T \\
\vdots \\
r_d^T \\
 c^T C^t
\end{pmatrix}.
$$

To find the form of matrix $K_m$, let $\sigma_m$ take the form:

$$
\sigma_m = \begin{pmatrix}
\sigma_{m,0} \\
\sigma_{m,1} \\
\vdots \\
\sigma_{m,d-1}
\end{pmatrix}
$$
Then $C\sigma_m$, will look like

$$
C\sigma_m = \begin{pmatrix}
0 & 1 & 0 & \ldots & 0 \\
0 & 0 & 1 & \ldots & 0 \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
0 & \ldots & 0 & 1 & 0 \\
0 & \ldots & 0 & 0 & 1 \\
c_0 & c_1 & \ldots & c_{d-2} & c_{d-1}
\end{pmatrix}
$$

and it follows that

$$
K_m = \begin{pmatrix}
\sigma_{m,0} & \sigma_{m,1} & \sigma_{m,2} \\
\sigma_{m,1} & \sigma_{m,2} & \sigma_{m,3} \\
\vdots & \vdots & \vdots \\
\sigma_{m,d-3} & \sigma_{m,d-2} & \sigma_{m,d-1} \\
\sigma_{m,d-1} & c^T \sigma_m
\end{pmatrix}
$$

Similarly, $K_z$ will look like $K_m$ but with $\sigma_m$ replaced by $\sigma_z$, where the vector $\sigma_z$ is given by

$$
\sigma_z = \begin{pmatrix}
\sigma_{z,0} \\
\sigma_{z,1} \\
\vdots \\
\sigma_{z,d-1}
\end{pmatrix}
$$

**Insertion with replacement**

In this case, $C'$ takes the form shown in Equation 5.3, so $C^2$ will be as follows:
5.1. General model for MAC generation using direct injection

\[ C^2 = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & \ldots & \ldots & \ldots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \ldots & 0 & 1 & 0 & \ldots & \ldots & \ldots & 0 \\ 0 & \ldots & \ldots & 0 & 0 & \ldots & \ldots & \ldots & 0 \\ 0 & \ldots & \ldots & \ldots & 0 & 0 & \ldots & \ldots & 0 \\ 0 & \ldots & \ldots & \ldots & 0 & 1 & \ldots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \ldots & \ldots & \ldots & \ldots & \ldots & 0 & 1 \\ c_0 & c_1 & \ldots & \ldots & \ldots & \ldots & c_{d-2} & c_{d-1} \end{pmatrix} \left\langle (i) \right. \]

Continuing similarity, once \( t \geq i \), \( C^t \) will have \( i \) empty rows at the top. If there are multiple zero rows, \( C^t \) will eventually be empty in and above the last of these (for \( t \geq \) the largest gap between empty rows).

The series of terms in \( K_m \) will also be affected as follows:

\[ C \sigma_m = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & \ldots & \ldots & \ldots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \ldots & 0 & 1 & 0 & \ldots & \ldots & \ldots & 0 \\ 0 & \ldots & \ldots & 0 & 0 & \ldots & \ldots & \ldots & 0 \\ 0 & \ldots & \ldots & \ldots & 0 & 1 & \ldots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \ldots & \ldots & \ldots & \ldots & \ldots & 0 & 1 \\ c_0 & c_1 & \ldots & \ldots & \ldots & \ldots & c_{d-2} & c_{d-1} \end{pmatrix} \begin{pmatrix} \sigma_{m,0} \\ \sigma_{m,1} \\ \vdots \\ \sigma_{m,i-1} \\ 0 \\ \vdots \\ \sigma_{m,i+1} \\ \vdots \\ \sigma_{m,d-1} \\ c^T \sigma_m \end{pmatrix} \left\langle (i) \right. \]
Chapter 5. General model for MAC generation using direct injection

and

\[
C^2\sigma_m = \begin{pmatrix}
0 & 0 & 1 & 0 & 0 & \ldots & \ldots & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ldots & \ldots & \vdots \\
0 & \ldots & 0 & 1 & 0 & \ldots & \ldots & 0 \\
0 & \ldots & \ldots & 0 & 0 & \ldots & \ldots & 0 \\
0 & \ldots & \ldots & 0 & 0 & \ldots & \ldots & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ldots & \ldots & \vdots \\
0 & \ldots & \ldots & 0 & 1 & \ldots & \ldots & 0 \\
0 & \ldots & \ldots & \ldots & 0 & 1 & \ldots & 0 \\
\end{pmatrix} \begin{pmatrix}
\sigma_{m,0} \\
\sigma_{m,1} \\
\vdots \\
\sigma_{m,i-1} \\
\cdots \\
\sigma_{m,d-1} \\
\end{pmatrix} = \begin{pmatrix}
\sigma_{m,2} \\
\sigma_{m,3} \\
\vdots \\
\sigma_{m,i-1} \\
0 \\
\sigma_{m,i+1} \\
\vdots \\
\sigma_{m,d-1} \\
\end{pmatrix} \leftarrow (i)
\]

So \( K_m \) will look like the following:

\[
K_m = \begin{pmatrix}
0 & \ldots & 0 & \sigma_{m,i-1} & \ldots & \sigma_{m,0} \\
\vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\
0 & \ldots & \ldots & 0 & \sigma_{m,i-1} \\
\cdots \\
\end{pmatrix} \begin{pmatrix}
\sigma_{m,0} \\
\sigma_{m,1} \\
\vdots \\
\sigma_{m,i-1} \\
\cdots \\
\sigma_{m,d-1} \\
\end{pmatrix}
\]

(5.5)

It is clear that states \( S_0^i, \ldots, S_{i-1}^i \) are only affected by the final \( i \) bits of the message.

Similarly, the series of terms in \( K_z \) will also be affected in the same manner as for \( K_m \), just we replace the value of \( \sigma_m \) by the value of \( \sigma_z \).

5.2 Reviewing the Nakano et al. model

Nakano et al. \[76\] consider using an LFSR with an associated nonlinear filter to generate a hash function for a given message, a process they describe as injection of the message into the internal state of the function. In considering this situation, they analysed two methods for injecting a message directly into
the LFSR. Briefly, these methods are injection (with XOR) into the last stage of the register, and injection (with XOR) into several regularly spaced stages of the register. In their analysis, they also assume that the attacker knows the plaintext and the corresponding hash value.

Recall that the general model presented in Section 5.1 takes the form

\[ A_l = C^t A_0 \oplus K_m M_{l-1} \oplus K_z Z_{l-1}. \]

In this section we first show how the Nakano et al. models can be represented in our general construction, and then explain the extension of the concept to the formation of MACs.

5.2.1 Previous model (plaintext hash function)

In both of the approaches described in [76], the message bit \( m_t \) and filter output \( z_t \) are combined using the XOR operation. This combination is then XORed with the register feedback \( y_t \) and inserted into either the final stage of the register, or (in the other approach) XORed into multiple stages of the register. Since Nakano et al. are considering using the keystream generator structure as an unkeyed hash functions, they do not use a key \( K \) or an IV and they assume that register \( A \) is initialised with zeroes, i.e. \( A_0 = (0 \ldots 0)^T \).

Since \( \sigma_m = \sigma_z \) in this scenario, the values of \( K_m \) and \( K_z \) are identical, so we write \( K = K_m = K_z \). Then both models can be represented in the form:

\[ A_l = K (M_{l-1} \oplus Z_{l-1}) \] (5.6)

For their first approach

In this case, \( \sigma_m = \sigma_z = (0 0 0 0 0 0 0 1)^T \). For example, if the feedback polynomial \( f(a) = a^8+a^7+a^2+a+1 \), and so \( c^T = (1 1 0 0 0 0 1 1)^T \), then \( K \) for a message length of \( l = 16 \) bits \( K \) will look like:
Chapter 5. General model for MAC generation using direct injection

\[
\mathbf{K} = \begin{bmatrix}
1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\
\end{bmatrix}
\tag{5.7}
\]

For their second approach \( \mathbf{K} \) depends on the value of \( \sigma_m \) or \( \sigma_z \) which are predefined by the cipher specifications. For example, if \( \sigma_m = \sigma_z = (0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0) \) and the feedback polynomial \( f(a) = a^8 + a^7 + a^2 + a + 1 \), then again \( c^T = (1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1) \), and \( \mathbf{K} \) for a message length of \( l = 16 \) bits \( \mathbf{K} \) will look like:

\[
\mathbf{K} = \begin{bmatrix}
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\
1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
\end{bmatrix}
\tag{5.8}
\]

5.2.2 Extension to generate MAC tags

Our model generalises Nakano et al.’s proposals to the wider context of generating MAC tags for either authentication only or authenticated encryption applications. In this context, the input message being accumulated may be either plaintext or ciphertext; if it is ciphertext, the plaintext may not be known to the attacker.

Furthermore, the internal state of the register in the Nakano et al. model does not use a key and \( IV \) to initialise these stages and so they assumed these to be initialised with zero values. In our extended model, as discussed previously, we use the keystream sequence generated using a key \( K \) and an \( IV \) to initialise the internal states of the cipher in the preparation phase of MAC generation. So
when the model is used to generate a MAC, $A_0$ in Equation 5.4 will consist of unknown values.

5.2.3 Extension to word based registers

In Nakano et al.’s work, the nonlinear filter generator is based on a binary register and all the operations involved in generating a hash value are in GF(2). However, modern stream cipher designs frequently use word based registers to gain efficiency. We extend the work to include the word based registers and investigate the security of MAC generation using word based stream ciphers.

Word based LFSRs can be more efficient in software especially when the finite field that underlies the LFSR operations is suited to a processor. The best choices for such a field are the Galois Fields that have the number of elements equal to $2^w$ elements (GF($2^w$)), where $w$ is the size of item in the entire processor used by the application. All the elements and the coefficients of the recurrence relation in this field utilize one unit of storage in this processor. Note that the common word size $w$ in literature are 16, 32 and 64 bits.

With word based registers, the designer can choose either bitwise XOR or addition modulo $2^w$ as addition operations on the word based variables, while multiplication uses the multiplication operation over GF($2^w$). This can easily be represented in our model, by changing the entries in Equation 5.4 to words instead of bits. That is, the elements of the register update matrix $C$ (in particular, the feedback row $c^T$), $M_t$ and $Z_t$ will all be words rather than bits and the matrix multiplication operations will use the relevant definition for adding and multiplying elements.

5.3 Current proposals which follow this model

In this section, we examine three stream ciphers with authentication mechanisms which can be considered as instances of the general model presented in Section 5.1. All three ciphers are word based, with word sizes of either 16 or 32 bits. These algorithms are SSS [55], NLSv2 [56] and SOBER-128 [54]. In this section, we do not consider the provision of confidentiality. We investigate the integrity assurance component only. In particular, the accumulation process of the input message is described in the following subsections for each cipher.
Note that the full descriptions for these three stream ciphers including the three phases for MAC generation were given in Chapter 3.

5.3.1 SSS

The SSS stream cipher uses a word size of 16 bits, so operations are performed over GF($2^{16}$). The input message word is accumulated directly into the final stage of the accumulation register $A$. The accumulated input word is combined with the feedback word by a bitwise XOR operation, and then the resulting word is injected into stage $a[16]$ of the register, as shown in Figure 5.2. Note that, the input message is the plaintext message which is unknown to an attacker. In SSS, a nonlinear filter is not used, so in this case, $\sigma_z = 0$ and $\sigma_m$ is given by:

\[
\sigma_m = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}
\]

For this particular feedback function, the companion matrix $C$ is of size $(17 \times 17)$, and the final row of this matrix is given by: $
\left( \delta_{\text{SSS}} \ 0 \ 0 \ 0 \ 1 \ 0 \ \cdots \ 0 \ 1 \ 0 \right)$. Where $\delta_{\text{SSS}} = 0x0001$.

\[ A_f = C^f A_0 \oplus K_m M_{l-1} \]
5.3. Current proposals which follow this model

\[ A_l = C^t A_0 \oplus \begin{pmatrix} c^T C^{d-1} \sigma_m & \ldots & 0 & 0 & 0 \\ c^T C^{d-2} \sigma_m & \ldots & 0 & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ c^T C^{l-3} \sigma_m & \ldots & c^T \sigma_m & 1 & 0 \\ c^T C^{l-2} \sigma_m & \ldots & c^T C \sigma_m & c^T \sigma_m & 1 \end{pmatrix} \begin{pmatrix} m_0 \\ m_1 \\ \vdots \\ m_{l-2} \\ m_{l-1} \end{pmatrix} \]  \hspace{1cm} (5.9)

where \( A_0 \) is the initial content of register \( A \) which is initialised using keystream words and \( C \) is the companion matrix of SSS which can be described as follows:

\[
C = \begin{pmatrix}
0 & 1 & 0 & 0 & 0 & \ldots & 0 & 0 & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\
0 & 0 & \ldots & \ldots & 0 & 0 & \ldots & 0 & 0 & 1 \\
\delta_{SSS} & 0 & 0 & 0 & 1 & 0 & \ldots & 0 & 1 & 0
\end{pmatrix}
\]

5.3.2 NLSv2

NLSv2 \[56\] uses a word size of 32 bits, so operations are performed over GF(2^{32}). During the accumulation phase, the input message word is accumulated directly into the final stage of the accumulation register \( A \). The input message word is accumulated by combining it with the word of the feedback using the bitwise XOR operation. Then the resulting word is injected into stage \( a[7] \) of the register as shown in Figure \[5.3\]. Note that the input message is the plaintext message which is unknown to an attacker. In NLSv2, a nonlinear filter is also not used, so again, \( \sigma_z = 0 \) and \( \sigma_m \) will look like the following:

\[
\sigma_m = \begin{pmatrix}
0 \\
0 \\
\vdots \\
0 \\
1
\end{pmatrix}
\]

For this particular feedback function, the companion matrix \( C \) is of size (8\times8), and the final row of this matrix will be \( \begin{pmatrix} \delta_{NLSv2} & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \) where \( \delta_{NLSv2} = 0x000001 \).

Let \( A_l \) be the final state of register \( A \) after accumulating the input message.
Then the equation for the whole accumulation process of the input message that is described in Equation 5.4 will look like the following in the matrix representation form:

\[
A_l = C^l A_0 \oplus K_m M_{l-1}
\]

\[
A_{l+1} = C^{l+1} A_0 \oplus \begin{bmatrix}
    c^T C^{l-d-1} \sigma_m & \ldots & 0 & 0 & 0 \\
    c^T C^{l-d} \sigma_m & \ldots & 0 & 0 & 0 \\
    \vdots & \ddots & \ddots & \ddots & \vdots \\
    c^T C^{l-3} \sigma_m & \ldots & c^T \sigma_m & 1 & 0 \\
    c^T C^{l-2} \sigma_m & \ldots & c^T C \sigma_m & c^T \sigma_m & 1
\end{bmatrix}
\begin{bmatrix}
    m_0 \\
    m_1 \\
    \vdots \\
    m_{i-2} \\
    m_{i-1}
\end{bmatrix}, \quad (5.10)
\]

where \( A_0 \) is the initial content of register \( A \) which is initialised using keystream words and \( C \) is the companion matrix of NLSv2 which can be described as follows:

\[
C = \begin{bmatrix}
    0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
    \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
    0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
    \delta_{NLSv2} & 0 & 0 & 0 & 1 & 0 & 0
\end{bmatrix}
\]

### 5.3.3 SOBER-128

SOBER-128 [54] has a word size of 32 bits, so operations are performed over \( \text{GF}(2^{32}) \). In this cipher the input message is accumulated into one stage of the register. The input message word is combined with the contents of stage \( a[4] \) of the register \( A \) by addition modulo \( 2^{32} \), and the output word then forms an
input word to the nonlinear filter. Then the output result of the nonlinear filter is used to replace the previous contents of the stage $a[4]$ of the internal state of the register as shown in Figure 5.4. Note that the input message is the plaintext message which is unknown to an attacker.

![Figure 5.4: Message accumulation mechanism in SOBER-128](image)

For SOBER-128, the filter output $z_t$ depends directly on the message word $m_t$ and $m_t$ is not injected separately into the register, so $\sigma_m$ is effectively all zeroes. As noted above, the filter output $z_t$ replaces the old contents of stage $a[4]$ with the new contents, so $\sigma_z$ in this case will look like the following:

$$\sigma_z = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}$$
Let the nonlinear filter function in Figure 5.4 be denoted by \( f \). Then we have
\[
Z_{l-1} = \begin{pmatrix}
    z_0 \\
    z_1 \\
    \vdots \\
    z_{l-2} \\
    z_{l-1}
\end{pmatrix},
\]
where \( z_t = f(m_t + a_t[4]) \) and + is an addition modulo \( 2^{32} \). For this particular feedback function, the state update matrix \( C \) is of size \( (17 \times 17) \), and the final row of this matrix will be
\[
\begin{pmatrix}
    \delta_{SOBER-128} & 0 & 0 & 0 & 1 & 0 & \cdots & 0 & 1 & 0
\end{pmatrix}
\]
where \( \delta_{SOBER-128} = 0x000001 \).

Then the whole accumulation process for the input message can be presented using Equation 5.4, as follows:
\[
A_t = C^tA_0 \oplus K_zZ_{l-1}
\]
where \( A_0 \) is the initial content of register \( A \) which is initialised using keystream words. \( C \) is the state update matrix of SOBER-128 which can be described as follows:
\[
C = \begin{pmatrix}
    0 & 1 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\
    0 & 0 & 1 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\
    0 & 0 & 0 & 1 & 0 & 0 & 0 & \cdots & 0 & 0 \\
    0 & 0 & 0 & 0 & 1 & 0 & 0 & \cdots & 0 & 0 \\
    0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 1 & 0 \\
    0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 1 \\
    \delta_{SOBER-128} & 0 & 0 & 0 & 1 & 0 & \cdots & 0 & 1 & 0
\end{pmatrix}
\]
Then the full matrix equation will be as follows:

\[
A_l = C^l A_0 \oplus \\
\begin{pmatrix}
0 & \ldots & 1 & 0 & 0 & 0 & 0 \\
0 & \ldots & 0 & 1 & 0 & 0 & 0 \\
0 & \ldots & 0 & 0 & 1 & 0 & 0 \\
0 & \ldots & 0 & 0 & 0 & 1 & 0 \\
0 & \ldots & 0 & 0 & 0 & 0 & 1 \\
\end{pmatrix}
\begin{pmatrix}
0 & \ldots & 0 & 0 & 0 & 0 & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\
0 & \ldots & 0 & 0 & 0 & 0 & 0 \\
0 & \ldots & 0 & 0 & 0 & 0 & 0 \\
\end{pmatrix}
\begin{pmatrix}
z_0 \\
z_1 \\
z_2 \\
\vdots \\
z_{l-3} \\
z_{l-2} \\
z_{l-1} \\
\end{pmatrix}
\]

(5.11)

5.4 Security analysis

In this model of message injection, the message accumulation phase plays a critical role in the security of the MAC generation. This analysis is an evaluation of the security of the message accumulation process for direct message injection with respect to forgery attacks based on collisions.

Consider the possibility of a forgery attack being conducted as follows. Suppose for a message \( M \), a MAC tag \( MAC_{K,IV}(M) \) is generated using key \( K \) and \( IV \). The sender intends to transmit the message-MAC tag pair to a particular receiver. Assume a man in the middle attacker intercepts the message-MAC tag pair, and tries to modify \( M \) and possibly also \( MAC_{K,IV}(M) \) to calculate a valid MAC tag \( MAC_{K,IV}(M') \) for a modified message \( M' \). The attacker then sends the new pair \( (M', MAC_{K,IV}(M')) \) to the intended recipient. If it is possible to alter \( M \) to \( M' \) and provide a valid \( MAC_{K,IV}(M') \) without any knowledge of the keystream sequences used to generate \( MAC_{K,IV}(M) \), the forgery attack is then successful.

In this section, we investigate the possibility of such forgeries for MACs formed by direct message injection, for modifications of the original message \( M \) which result in collisions during the accumulation process of the input message. Any collision which occurs in this phase for any two distinct messages using the same keystream sequence will result in the same MAC tag for those two messages regardless of the process performed in the other two phases of
Chapter 5. General model for MAC generation using direct injection

MAC generation (preparation and finalisation), thus giving an obvious forgery. We start by analysing the general case for collisions in our model. Then, we apply this analysis to the Nakano et al. model used for hash functions, as outlined in Section 5.2. Finally, the MAC generation component of the three stream ciphers that represent instances of our model, described in Section 5.3, are analysed.

5.4.1 Analysis of collisions in the model

The following analysis considers the possibility of obtaining collisions in the final contents of the register, which would lead to successful MAC forgery attacks as discussed above. In doing so, we consider for completeness the case where the message and the initial contents of the register are known. We note, however, that a MAC cannot be secure under these conditions, since any attacker can replicate the initial state of the register and obtain the corresponding final state for any message by simply accumulating that message in the normal way.

As discussed in Section 5.1, the two methods for inserting message bits and filter output bits into the internal state of a binary register are: insertion with XOR and insertion with replacement. We first analyse the situation where insertion is performed by XORing, as described in Section 5.1.3.

The nonlinear filter in our model may or may not be used in the accumulation process. Based on that, we further divide the analysis into two main cases.

(1) Insertion using XOR

Case 1: If nonlinear filter is not used

In this case Equation 5.4 will be as follows:

\[ A_t = C^t A_0 \oplus K_m M_{t-1} \] (5.12)

Since the nonlinear filter output is not used, all the operations in the accumulation process will be linear. This makes it possible to manipulate the message to force a collision for two distinct messages without knowing the contents of the message or the initial contents of the register. The manipulation can be clearly explained from the matrix formulation of \( K_m \) according to the following theorem:

Theorem 5.1 Providing \( C \) is the companion matrix for a LFSR with a primitive feedback polynomial of degree \( d \), then any \( d \) consecutive columns \( C^{t+d-1} \sigma_m, \ldots, C^{t+1} \sigma_m, C^t \sigma_m \) in \( K_m \) must be linearly independent.
Proof We show that $\sigma_m, C\sigma_m, \ldots, C^{d-1}\sigma_m$ are linearly independent. Since $C$ is non-singular, it follows that this also applies to $C\sigma_m, C^{i+1}\sigma_m, \ldots, C^{i+d-1}\sigma_m$ for any $i \geq 0$. Recall that, a word-based register contain a word size of $w$ bits.

Let $k$ be the smallest number for which $C^k\sigma_m = \sum_{i=0}^{k-1} a_i \sigma_m + a_{k-1} C^{k-1}\sigma_m$, where $a_i \in\{0, 1\}$, for $0 \leq i \leq k - 1$. Note that $C^i\sigma_m$ can then be expressed as $b_i\sigma_m + b_i C\sigma_m + \cdots + b_k - 1 C^{k-1}\sigma_m$ for $i \geq 0$. (Since $C^{k+1}\sigma_m = C(a_0\sigma_m + a_1 C\sigma_m + \cdots + a_{k-2} C^{k-2}\sigma_m) + a_{k-1} C^k\sigma_m$.) By induction, the same is obviously true for $C^i\sigma_m$, for any $l \geq k$. Now consider the set $U = \{u = C^i\sigma_m | l \geq 0\}$. From the expression above, $|U| \leq 2^{wk}$, since $b_i \in\{0, 1, \ldots, 2^w - 1\}$ for $i = 0, \ldots, k - 1$. But if the LFSR has a primitive feedback polynomial, then $U$ must cycle through all $2^{wd} - 1$ non-zero vectors in $\{0, 1, \ldots, 2^w - 1\}^d$, so we must have $k \geq d$. But $k \leq d$, since we are working in a $d$-dimensional vector space, so $k = d$ and the $d$ vectors $\sigma_m, C\sigma_m, \ldots, C^{d-1}\sigma_m$ are linearly independent.

Since the columns $C^i\sigma_m$ have exactly $d$ elements, any $d+1$ successive columns must be linearly dependent, so we may express $C^d\sigma_m = a_0 C^0\sigma_m + a_1 C^1\sigma_m + \cdots + a_{d-1} C^{d-1}\sigma_m$. If we now consider the process of multiplying these columns by the corresponding bits of $M_{t-1}$, we see that flipping the message bit corresponding to $C^d\sigma_m$ is equivalent to flipping the message bits corresponding to those columns $C^i\sigma_m$ for which $a_i = 1$. It is clear from this that any message of length $l \geq d + 1$ bits, can be changed in such a way that the final register contents are unchanged. Hence a collision will occur and as a result of that an attacker can send a forged MAC tag utilizing this type of collision attack.

Note that the above results apply whether the initialisation of register $A$ is known or unknown, since all the operations are linear.

If the model under this assumption is used for authentication only, then the message is the plaintext which is known to the attacker. This makes it easy to manipulate the input message such that the accumulation of two messages gives the same content for register $A$ at time $t$, and hence the collision will occur. If the model is used for authenticated encryption where the input message is the plaintext and it is assumed to be unknown, then an attacker can still forge a MAC tag by manipulating ciphertext without knowing the plaintext.

Case 2: If nonlinear filter is used

If we include the nonlinear filter in the accumulation process for the input message, then there are three further cases for message and filter output injection into the internal states.
Case 2a: if message and initial state are both known, and \((\sigma_m = \sigma_z)\). Then \(K_m = K_z\) and we can still manipulate the input message. Since \(z_t\) can be determined at each step and we are accumulating \((m_t + z_t)\), so \(m_t\) can be complemented when required to achieve the same result as before and hence force the collision to occur.

Case 2b: if message and initial state are both known, and \((\sigma_m \neq \sigma_z)\). Then \(K_m \neq K_z\) and so it may not be possible to adjust for the effects of \(z_t\). In principle any change in \(z_t\) at a single time point can be counteracted by a suitable combination of manipulated message bits \(m_{t+1}, \ldots, m_{m+d}\), but these changes may then result in changes to later bits of \(Z\), and there is no guarantee that all of these can be adjusted for. It may be possible to manipulate the register contents to obtain a collision for some special cases of message content and initial register contents, but these cases would be extremely rare.

Case 2c: if either plaintext or initial state is unknown, and \((\sigma_m = \sigma_z \text{ or } \sigma_m \neq \sigma_z)\). Then, \(z_t\) will be unknown, so it is not possible to determine how to manipulate \(m_t\) to get the desired result. Thus, the accumulation process will be secure in this case. (However, if the initialisation is known, the overall MAC process will be insecure, as discussed previously.)

For Case 2, the accumulation will be secure in providing for both authentication and AE applications, since the register should be initialised with key-dependent values. For the application of AE, the plaintext will also be unknown further guaranteeing the security of the MAC generation.

(2) Insertion with replacement

As before, consider two cases according to whether the filter output is used. If the filter output is not used, we can apply a similar analysis to the one given above, to show that a collision can be forced during the accumulation process. In fact, for insertion with replacement, fewer columns of \(K_m\) are needed to generate a collision, due to the form of \(K_m\). Note from Equation 5.5 that the first \(i\) stages of LFSR are only affected by the final \(i\) bits of the message in \(K_m\). Therefore, to obtain a collision in the register states, we only need to consider \((d + 1 - i)\) bits among the first \((l - i)\) bits of the input message of length \(l\).

If the filter output is used, then similar arguments apply as given previously, that is: collisions can again be forced if both the input message and initialisation...
are known and the input message and the filter output are injected into the same stages, but not if either the input message or initialisation is unknown.

A summary for different cases of these analyses is presented in Table 5.1.

Table 5.1: Summary of the analysis for MAC generation using direct message injection.

<table>
<thead>
<tr>
<th>Cases</th>
<th>Nonlinear filter</th>
<th>Message and Register initialisation</th>
<th>Forced collisions?</th>
<th>Overall Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>not used</td>
<td>any</td>
<td>Yes</td>
<td>not secure</td>
</tr>
<tr>
<td>2a</td>
<td>used</td>
<td>both known</td>
<td>Yes</td>
<td>not secure</td>
</tr>
<tr>
<td>2b</td>
<td>used</td>
<td>both known</td>
<td>Unlikely</td>
<td>not secure</td>
</tr>
<tr>
<td>2c</td>
<td>used</td>
<td>exactly one known</td>
<td>No</td>
<td>secure</td>
</tr>
</tbody>
</table>

5.4.2 Analysis of Nakano et al. models

As mentioned previously, Nakano et al. [76] analysed two methods for direct message injection into the binary register of a nonlinear filter generator to generate a hash value. Both of these methods can be described by Case 2a in Table 5.1. The first method is to inject the input message bit combined with the filter output bit and the feedback bit into the final stage of the entire register. For this method, an attacker can control the content of the entire register using the input message, since the value of $z_t$ at any time can be determined by knowing the register contents. Therefore, a collision in the register can be forced to occur. Our analysis confirms their conclusion that the full register can be manipulated and hence it is easy to force a collision.

The second method is to inject the combination of the input message bit and the filter output bit into multiple regularly spaced stages of the internal state. If the message is injected into $r$ stages at intervals of $d/r$, they claim that the probability of obtaining a collision $p(\text{collision})$ will be:

$$p(\text{collision}) = 2^{-d(1-1/r)/2}.$$ (5.13)

In obtaining this result, Nakano et al. assumed that the effect of modifying the input message could only be counteracted on one of the intervals between insertion points in the register. However, our analysis shows that the full register can still be manipulated by suitable modification of the input message in this
case. From Theorem 5.1, we can see that after $d$ consecutive columns in the $K$ matrix, we have a column that is dependent on the previous columns. So we can adjust for the effect of changing a bit in the input message by changing a suitable combination of other bits in the message. The only constraint on obtaining a collision by using this process is that the length of the input message must be greater than $d$ bits.

### 5.4.3 Analysis of current proposals using this model

In this section we analyse the MAC generation process for three authenticated encryption stream ciphers that can be considered as instances of the direct message injection model. All of these ciphers use word based registers.

**SSS**

As discussed in Section 3.3.1 for the SSS stream cipher [55], during the preparation phase of MAC generation, register $A$ is initialised with keystream words, and as noted from Equation 5.9 the $A_0$ term is present in the equation. It is obvious also from Figure 5.2 that the accumulation process of the input message does not involve using a nonlinear filter in the accumulation register. The input message words that are accumulated into the register are the plaintext words which are supposed to be unknown to the attacker. Based on that, SSS is considered to meet the conditions for Case 1 in Table 5.1; as discussed above, the accumulation process is not secure against forgery attacks based on collision in this case.

However, for SSS the input message words are also used to generate keystream words in a self synchronous manner described in Chapter 3. This process is beyond the scope of our analysis in this section, as we focus on the accumulation process only. Because the keystream is generated using a self synchronous mechanism, manipulating the input message words results in different output keystream words and so the resulting ciphertext will be changed unpredictably. So the fact that SSS uses a self synchronous stream cipher instead of using nonlinear filter in the accumulation process will prevent a valid forgery from being obtained in this manner. Even if register $A$ uses a known initialisation value, such as all zero values, and the first term of Equation 5.9 is zero, this additional mechanism prevents the attacker from generating a MAC corresponding with a modified message.
NLSv2

As discussed in Section 3.3.4 for the NLSv2 stream cipher [56], during the preparation phase of MAC generation, register \( A \) is initialised with keystream words and we also note from Equation 5.10 the \( a_0 \) term is present in the equation. It is obvious from Figure 5.3 that the accumulation process of the input message words does not involve using a nonlinear filter in the accumulation register. The input message words that are accumulated into the register are the plaintext words which may be unknown to the attacker. Based on that, NLSv2 is considered to meet the conditions for Case 1 in Table 5.1; as discussed above, the accumulation process is not secure against forgery attacks based on collision in this case.

However, the input message word in NLSv2 also is accumulated at the same time into a different register, as described in Chapter 3. This accumulation process is not linear, a nonlinear function is used when the input message is accumulated into this different register. In this case, manipulating the input message words will affect the nonlinear register in an unpredictable way. Introducing any differences in the input message word to force a collision in the linear register will result in different contents in the nonlinear register. So when the attacker tries to cancel the difference that was introduced in the input word for the linear register, this word will also introduce another different word in the second register and so the attacker cannot control that collision by this way. So the fact that NLSv2 uses both a linear and a nonlinear register for message accumulation prevents simultaneous collisions in the two registers. Even if the initialisation values are known such as using zero values and the first term of Equation 5.10 is zero, the attacker cannot manipulate the input message to obtain collisions in both registers at the same time.

SOBER-128

As discussed in Section 3.3.5 for the SOBER-128 stream cipher [54], during the preparation phase of MAC generation, register \( A \) is initialised with keystream words and also as noted from Equation 5.4 the \( a_0 \) term is present in the equation. It is shown in Figure 5.4 that the accumulation process of the input message words involves using a nonlinear filter in the accumulation register. The input message words that are accumulated into the register are the plaintext words which are supposed to be unknown to the attacker. Based on that, SOBER-128
is considered to meet the conditions for Case 2c in Table 5.1 as discussed above, the accumulation process is secure against forgery attacks based on collision in this case. There are two inputs for the nonlinear filter; the input message and the content of stage \( a[4] \) of the register. So manipulating the input message will result in unknown values for the nonlinear filter output since the nonlinear filter also uses the content of the register to accumulate the input message which is unknown to the attacker. Also the nonlinear behaviour of the filter prevents the collision due to introducing differences in the input message words on the entire register.

However, there is a forgery attack on SOBER-128 [98] that applies a similar method of introducing differences in the input message words and cancelling out this difference after a certain number of clocks, but this attack is not due to the weakness of the accumulation process. It is due to the weakness of the nonlinear filter that is used in the accumulation process. This attack assumes that the input message and \( A_0 \) have certain formats to force the collision to occur. So for this type of accumulation process, the security of the model relies on the strength of the nonlinear filter that is used in the accumulation process which is one of our assumptions for using this model.

5.5 Summary and conclusion

This chapter describes a general model for generating MAC tags using a nonlinear filter keystream generator by injecting the input message directly into the internal state of the keystream generator. This message could be either plaintext or ciphertext. The injected message could be in bit level or in word level. For each level of the input message, the other two inputs (the feedback and the output of nonlinear filter) will be consistent with the message level. The MAC tag provides a mechanism for message authentication; one of the two security services provided by AE stream ciphers.

We present two methods for direct message insertion into the internal state of LFSR. The first method is to combine the input message with the filter output and the previous content of the state by XOR operation. The second method of insertion is by replacing the old content of the state with the new combination of the input message and the filter output. For each method, we describe the whole process of accumulating the input message using a matrix representation.
We also describe three existing stream cipher authentication algorithms that can be described by this model. Our model generalises the form of MAC generation using direct message injection based on the keystream generator for a stream cipher and includes the Nakano et al. model that was used to generate a hash value in the same manner.

We then provide a security analysis of the general model and describe conditions under which a collision can be forced which would enable a man-in-the-middle forgery to be successful. Specifically we show that suitable initialisation of the register and active use of the nonlinear filter prevent an attacker from finding a collision which would result in a forged MAC. We apply this analysis to the cases discussed by Nakano et al. and find that our results are in line with Nakano et al. for their first method of injection, that is injecting into the last stage of the register. However, for their second method of injection, injecting into multiple stages of the register, our matrix representation enabled us to show, contrary to their conclusion, that collision can always be forced in the entire register. We also applied the results of our security analysis to the existing authentication proposals using direct injection. We found in two cases (SSS and NLSv2) that the accumulation mechanism used is not secure by itself. However in both cases, additional features of the authentication mechanism were used to overcome this weakness. For example, use of a self synchronous construction prevents an attacker from forging a valid MAC in the SSS stream cipher.

Based on our analyses, we recommend the following procedure to prevent collision attacks when using this model in the accumulation process to generate a MAC tag:

- Use a secure nonlinear filter in the accumulation process of the input message.
- Initialise the internal state of the LFSR with key-dependent secret values.

The matrix representation of our model is based on the use of a nonlinear filter generator as the accumulation register in the MAC generation process. It does not apply to other types of accumulation components, but this has not yet been done and therefore, this is a limitation of this model. Further work is needed to construct a similar model for other keystream generators.
Chapter 5. General model for MAC generation using direct injection
Chapter 6

General model for MAC generation using indirect injection

In this chapter, the security properties of the integrity mechanism are investigated, for a specific method of message injection. This chapter describes a general matrix based model for MAC generation in stream ciphers using the input message indirectly. It examines different options used in the preparation and finalisation phases for injecting the input message indirectly into the internal state of the cipher. Some current stream ciphers which can be considered as instances of this model will be presented in this chapter. We conduct an analysis of the model in relation to possible forgery and side-channel attacks.

The chapter is organised as follows. Section 6.1 describes our general model for generating a MAC using a stream cipher with indirect message injection. Three specific stream cipher algorithms that generate a MAC tag in this way (128-EIA3 [38,40], Grain-128a [1] and Sfinks [18]) are examined in Section 6.2. These ciphers have been proposed for widespread use in applications such as mobile telephony, and hence warrant careful examination. Our security analysis of this model against forgery attacks and side-channel attacks are presented in Sections 6.3 and 6.4. A summary and concluding remarks of this chapter are presented in Section 6.5.
6.1 MAC generation using indirect message injection

Some recent AE stream cipher proposals use a different authentication strategy from the direct injection model discussed in Chapter 5. Rather than accumulating either the plaintext or ciphertext directly, these proposals use the plaintext as a means to control the accumulation of a keystream sequence, with the accumulated value forming the basis of the MAC tag for the input message. In this section we describe explicitly a model for generating a MAC tag using stream ciphers in this manner. Note that this model can also use a ciphertext to control the accumulation of a keystream sequence into the registers, so we use the term message to cover either case.

6.1.1 Structure of the integrity algorithm

As discussed in Section 3.2, when injecting the input message indirectly the integrity component in Figure 3.2 consists of two registers, each of \( d \) binary stages. The relationship between these two registers, the keystream sequence \( y \) and the input message \( M \) was shown in Figure 6.1.

![Figure 6.1: MAC generation using indirect message injection](image)

The first register is a binary shift register, denoted \( R \). Let \( R_t \) denote the contents of \( R \) at time \( t \), with \( R_t = (r_t[0], \ldots, r_t[d - 1]) \) and initial contents
$R_0 = (r_0[0], \ldots, r_0[d-1])$. At each time clock $t$, for $t > 0$, the contents of $R$ are updated using the binary sequence $y$ as follows:

$$r_t[i] = \begin{cases} r_{t-1}[i+1], & \text{for } i = 0, \ldots, d-2 \\ y_{t-1}, & \text{for } i = d-1 \end{cases} \quad (6.1)$$

Thus the contents of register $R$ can be considered as a “sliding window” of length $d$ on the sequence $R_0||y$, where $||$ denotes concatenation.

The second register is an accumulation register, denoted $A$. Let $A_t$ denote the contents of $A$ at time $t$, with $A_t = (a_t[0], \ldots, a_t[d-1])$ and initial contents $A_0 = (a_0[0], \ldots, a_0[d-1])$. The contents of $A$ are updated using the contents of $R$, conditional on the value of the input message bit $m_t$, according to Equation $6.2$:

If $m_t = 1$ then $A$ is updated by XORing the current contents with the contents of $R$ at time $t$; otherwise $A$ remains unchanged:

$$A_t = \begin{cases} A_{t-1} \oplus R_{t-1}, & \text{if } m_t = 1 \\ A_{t-1}, & \text{otherwise} \end{cases} \quad (6.2)$$

The input message $M$ controls which $d$-bit segments of the sequence $R_0||y$ are accumulated into register $A$. When the $i^{th}$ message bit has been processed, the value in register $A$ can be expressed as $A_i = A_0 \oplus T_i M_i$, where $M_i$ consists of the first $i$ bits of the message $M$ and $T_i$ is a $d \times i$ matrix such that the $j^{th}$ column of $T_i$ contains $R_j$, for $j = 0, \ldots, i-1$; that is:

$$A_i = A_0 \oplus T_i M_i = \begin{pmatrix} a_0[0] \\ a_0[1] \\ \vdots \\ a_0[d-1] \end{pmatrix} \oplus \begin{pmatrix} r_0[0] & r_0[1] & \cdots & r_0[d-1] & y_0 & \cdots & y_{i-d-1} \\ r_0[1] & r_0[2] & \cdots & y_0 & y_1 & \cdots & y_{i-d} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ r_0[d-1] & y_0 & \cdots & y_{d-2} & y_{d-1} & \cdots & y_{i-2} \end{pmatrix} \begin{pmatrix} m_0 \\ m_1 \\ \vdots \\ m_{i-1} \end{pmatrix}.$$ 

Note also that each row of $T_i$ consists of $i$ consecutive bits from the sequence $R_0||y$ and is closely related to both the preceding and following rows.

When all of the message bits have been processed, the MAC finalisation phase
begins. This involves combining the final contents of $A$ with a masking value $F = (f[0], \ldots, f[d-1])$. Thus, for a message $M$ of length $l$ we have

$$MAC(M_l) = A_l \oplus F = A_0 \oplus T_l \oplus M_l \oplus F$$

(6.3)

### 6.1.2 Optional processes

For the general model using indirect message injection in the accumulation phase, as shown in Figure 6.1, we consider several options for the preparation and finalisation phases, respectively.

**In the preparation phase**, options relate to the initialisation of the two registers $R$ and $A$, and to preparation of the message. Either register could be initialised with fixed values, such as all zeroes; or with key dependent values, such as a segment from the keystream sequence $y$. The input message could be padded, by appending a specified sequence of bits at either the beginning, the end, or at both places. Alternatively no padding could be applied. If $M_l$ is padded with $n$ bits, we use $M_p$ to denote the padded message, where $p = l + n$.

**In the finalisation phase**, the final contents of accumulation register $A$ are combined with a mask, as shown in Figure 6.1. The mask may be obtained from the sequence used for the accumulation phase, $y$, or from another sequence. For AE this may be the sequence $z$ used for the confidentiality application. In some cases a null mask (all zero values) is used.

### 6.2 Current proposals using this model

Several currently proposed AE algorithms use the model presented in Section 6.1.1, but with slight differences in the options applied. In this section, we examine the MAC generation process for these algorithms. As each uses the same accumulation process, we describe only the options taken for the preparation and finalisation phases for these ciphers.

#### 6.2.1 128-EIA3 version 1.4

Version 1.4 of the 128-EIA3 [37] integrity algorithm uses the ZUC stream cipher [38] as a keystream generator. ZUC is a word-based stream cipher with word
6.2. Current proposals using this model

size 32 bits, that uses a 128-bit secret key and a 128-bit IV. The 128-EIA3 MAC tag has length \( d = 32 \) bits. The structure of 128-EIA3 is shown in Figure 6.2 and the two discretionary phases of MAC generation are as follows:

![Figure 6.2: Structure of 128-EIA3](image)

**Preparation phase:** 128-EIA3 does not use physical registers for \( R \) and \( A \) but instead uses variables \( k \) and \( T \) to represent these registers respectively. Register \( A \) is initialised with all zero values and register \( R \) is initialised with keystream sequence. Let \((y_{-32}, y_{-31}, \ldots, y_{-1})\) denote the first 32 bits of the keystream \( y \) produced; then

\[
R_0 = \begin{pmatrix}
y_{-32} \\
y_{-31} \\
\vdots \\
y_{-1}
\end{pmatrix}
\]

The input message \( M_l \) is padded by adding one bit at the end of the message, so \( M_p = M_l || 1 = (m_0, \ldots, m_{l-1}, 1) \).

**Finalisation phase:** After all \( l + 1 \) bits of the padded message have been processed, the contents of \( A \) at time \( l + 1 \) are combined with the final mask, a
32-bit segment from $y$ starting at $y_{l+32}$. The final MAC tag is given by:

$$MAC(M_p) = \begin{pmatrix} y_{-32} & y_{-31} & \cdots & y_{-1} & y_0 & \cdots & y_l \cdot 32 \\ y_{-31} & y_{-30} & \cdots & y_0 & y_1 & \cdots & y_{l-31} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ y_{-2} & y_{-1} & \cdots & y_9 & y_{30} & \cdots & y_{l-2} \\ y_{-1} & y_0 & \cdots & y_9 & y_{30} & \cdots & y_{l-1} \end{pmatrix} \begin{pmatrix} m_0 \\ m_1 \end{pmatrix} \oplus \begin{pmatrix} y_{l+32} \\ y_{l+33} \end{pmatrix}.$$ .

### 6.2.2 128-EIA3 version 1.5

Version 1.5 of 128-EIA3 was proposed in response to a successful forgery attack [44] on version 1.4. We discuss this attack in more detail in Section 6.3.3. This version is identical to version 1.4 except in the starting position of the final mask, obtained from the sequence $y$. Instead of starting at bit number $(l+32)$, the mask in version 1.5 starts at the beginning of the next 32-bit word of the bitstream. The final MAC tag is given by:

$$MAC(M_p) = \begin{pmatrix} y_{-32} & y_{-31} & \cdots & y_{-1} & y_0 & \cdots & y_l \cdot 32 \\ y_{-31} & y_{-30} & \cdots & y_0 & y_1 & \cdots & y_{l-31} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ y_{-2} & y_{-1} & \cdots & y_9 & y_{30} & \cdots & y_{l-2} \\ y_{-1} & y_0 & \cdots & y_9 & y_{30} & \cdots & y_{l-1} \end{pmatrix} \begin{pmatrix} m_0 \\ m_1 \end{pmatrix} \oplus \begin{pmatrix} m_{l-1} \\ 1 \end{pmatrix} \oplus \begin{pmatrix} y_{(\lceil l/32 \rceil +1) \cdot 32} \\ y_{(\lceil l/32 \rceil +1) \cdot 32 +1} \\ \vdots \\ y_{(\lceil l/32 \rceil +1) \cdot 32 +30} \\ y_{(\lceil l/32 \rceil +1) \cdot 32 +31} \end{pmatrix}.$$ .

### 6.2.3 Grain-128a

Grain-128a [1] is a bit-based cipher from the Grain family [60] with an added authentication mechanism. Grain-128a uses a 128-bit secret key and a 96-bit $IV$, and generates two sequences, $z_t$ and $y_t$, used for confidentiality and integrity applications, respectively. The MAC tag has length $d = 32$ bits. The structure of the MAC generation using Grain-128a is shown in Figure 6.3 and the two discretionary phases of MAC generation are as follows:
6.2. Current proposals using this model

Preparation phase: Both $R$ and $A$ are initialised directly from the Grain-128a bitstreams. Let $(y_{-64}, y_{-63}, \ldots, y_{-33}, y_{-32}, \ldots, y_{-1})$ denote the first 64 bits of the keystream $y$ produced; then

$$A_0 = \begin{pmatrix} y_{-64} \\ y_{-63} \\ \vdots \\ y_{-33} \end{pmatrix} \quad \text{and} \quad R_0 = \begin{pmatrix} y_{-32} \\ y_{-31} \\ \vdots \\ y_{-1} \end{pmatrix}.$$

The input message is padded at the end with a single bit of value 1, so $M_p = M_l || 1 = (m_0, \ldots, m_{l-1}, 1)$.

Finalisation phase: After all $l + 1$ bits of the padded message have been processed, the contents of $A$ at time $l + 1$ represents the final MAC tag. That is, the final mask is a null mask.

$$MAC(M_p) = \begin{pmatrix} y_{-64} \\ y_{-63} \\ \vdots \\ y_{-33} \end{pmatrix} \oplus \begin{pmatrix} y_{-32} & y_{-31} & \cdots & y_{-32} \\ y_{-31} & y_{-30} & \cdots & y_{-31} \\ \vdots & \vdots & \ddots & \vdots \\ y_{-1} & y_0 & \cdots & y_{l-1} \end{pmatrix} \begin{pmatrix} m_0 \\ m_1 \\ \vdots \\ m_{l-1} \\ 1 \end{pmatrix}. $$
6.2.4 Sfinks

The Sfinks [18] stream cipher proposal, submitted to eSTREAM [78], includes an authentication mechanism. The bit-based keystream generator uses an 80-bit secret key and an 80-bit IV, to form an initial state for the 256-bit LFSR, which is the major component of the keystream generator. Nonlinear filters are applied to the contents of the LFSR to produce two different sequences \( z \) and \( y \), used for confidentiality and integrity applications, respectively. In addition, two 64-bit registers are used in the authentication mechanism in the manner shown in Figure 6.1. The Sfinks MAC tag has length \( d = 64 \) bits. The structure of the MAC generation using Sfinks is shown in Figure 6.4, and the two discretionary phases of MAC generation are as follows:

Preparation phase: Before processing the \( l \)-bit message, both \( R \) and \( A \) are set to all zero and an initialisation algorithm is used to incorporate the first 128 bits of keystream \((y_{-128}, y_{-127}, \ldots, y_{-1})\) into the initial values of these registers. This algorithm consists of updating the two registers \( R \) and \( A \) 128 times according to Equations 6.1 and 6.2 but with \( m_i = 1 \) for \(-127 \leq i \leq 0\). Note that this process is equivalent to padding the message at the beginning by concatenating with a sequence of 128 ones. That is, \( M_p = (1, 1, \ldots, 1) || (m_0, \ldots, m_{l-1}) \).

Finalisation phase: The final contents of register \( A \) are combined (by XOR-ing) with a final mask that comprises 64 consecutive bits from the confidentiality sequence \( z \), beginning immediately after the segment used to encrypt the input.
message. The final MAC tag is:

\[
\text{MAC}(M_p) = \begin{pmatrix}
0 & 0 & \ldots & 0 & y_{-128} & \ldots & y_{-65} \\
0 & 0 & \ldots & y_{-128} & y_{-127} & \ldots & y_{-64} \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \ldots & y_{-67} & y_{-66} & \ldots & y_{-3} \\
0 & y_{-128} & \ldots & y_{-66} & y_{-65} & \ldots & y_{-2} \\
\end{pmatrix}
\begin{pmatrix}
m_0 \\
m_1 \\
\vdots \\
m_{l-1} \\
\end{pmatrix} \oplus
\begin{pmatrix}
z_l \\
z_{l+1} \\
z_{l+2} \\
z_{l+62} \\
z_{l+63} \\
\end{pmatrix}
\]

6.3 Security analysis against forgery attacks

Generating a MAC tag using the input message indirectly, as described above, is simple and efficient as the accumulation phase makes repeated use of the XOR operation. In this section, we analyse the security provided by this model with respect to forgery attacks. Then in the following section, we will analyse the security provided by this model with respect to side-channel attacks. Finally in the conclusion, we will give some recommendations for choosing options within this model to enhance its security against these attacks. In our analysis we assume that the binary sequences produced by the keystream generators are pseudo-random but cannot be distinguished from a truly random source.

Consider the possibility of a forgery attack being conducted as follows. Suppose for a message \(M\), a MAC tag \(MAC_{K,IV}(M)\) is generated using key \(K\) and \(IV\). The sender intends to transmit the message-MAC tag pair to a particular receiver, as shown in Figure 6.5. Assume a man in the middle attacker intercepts the message-MAC tag pair, and tries to modify \(M\) and possibly also \(MAC_{K,IV}(M)\) to calculate a valid MAC tag \(MAC_{K,IV}(M')\) for a modified message \(M'\). The attacker then sends the new pair \((M', MAC_{K,IV}(M'))\) to the intended recipient. If it is possible to alter \(M\) to \(M'\) and provide a valid \(MAC_{K,IV}(M')\) without any knowledge of the keystream sequences used to generate \(MAC_{K,IV}(M)\), the forgery attack is successful. In this section, we investigate the possibility of such forgeries for MACs formed by indirect message injection, for modifications involving flipping, deleting or inserting bits in the original message \(M\).

Our analysis against forgery attacks considers particularly the options for the preparation and finalisation phases, and explores the security implications of
Chapter 6. General model for MAC generation using indirect injection

Figure 6.5: Type of forgery attacks on our model

particular choices. In most cases, modifying bits between the ends of a non-zero message changes which segments of the (unknown) pseudo-random sequence are accumulated into register $A$, so this should not lead directly to forgery attacks. Instead we consider forgeries related to the modification of bits at the ends of the message $M$. Recall from Equation 6.3 that the MAC tag $MAC_{K,IV}(M_l)$ takes the form $MAC_{K,IV}(M_l) = A_0 \oplus T_l M_l \oplus F = (T_l M_l) \oplus (A_0 \oplus F)$. Note that the final MAC tag is simply the XOR combination of the separate effects of the accumulation process $T_l M_l$ and the masking vector $A_0 \oplus F$. In our analysis, we first consider the security of the accumulation process $T_l M_l$, and then the effect of the masking elements $A_0 \oplus F$.

6.3.1 Security of the accumulation process

Consider a message $M$ of length $l$, with no padding. Let the vector $X = (x_0, x_1, \ldots, x_{d-1})$ denote the output of the accumulation process. We represent the accumulation process in matrix form as follows:

$$X(M_l) = \begin{pmatrix}
   x_0 \\
   x_1 \\
   \vdots \\
   x_{l-1}
\end{pmatrix} = T_l M_l =
\begin{pmatrix}
   r_0[0] & r_0[1] & \ldots & r_0[d-1] & y_0 & \ldots & y_{l-d-1} \\
   r_0[1] & r_0[2] & \ldots & y_0 & y_1 & \ldots & y_{l-d-2} \\
   \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
   r_0[d-1] & y_0 & \ldots & y_{d-2} & y_{d-1} & \ldots & y_{l-2}
\end{pmatrix}
\begin{pmatrix}
   m_0 \\
   m_1 \\
   \vdots \\
   m_{l-1}
\end{pmatrix}.$$  \hspace{1cm} \text{(6.4)}

Where a message $M$ is modified to obtain a new message $M'$, we will use the notation $X' = (x'_0, x'_1, \ldots, x'_{d-1})^T$ to refer to $X(M')$. 

6.3. Security analysis against forgery attacks

Bit flipping forgeries

Consider firstly the case where \( R \) is initialised with zero values. Then all elements in the first column of matrix \( T_i \) are zero. Now \( m_0 \), the first bit of \( M_l \), has no effect on the value of \( X(M_l) \). Thus it is possible to modify \( M_l \) by flipping \( m_0 \). Let \( \overline{m}_l \) denote the complement of \( m_l \). The output \( X' \) for the modified message \( M' = (\overline{m}_0, \ldots, m_{l-1}) \) is exactly the same as \( X(M_l) \). This collision clearly leads directly to a forgery; the attacker can provide a valid \( X' \) for \( M' \) with probability of 1.

Similarly, since all elements of the second column are zero except possibly \( y_0 \), it follows that message bit \( m_1 \) only affects bit \( x_{d-1} \) of \( X(M_l) \). Modifying \( M_l \) by flipping \( m_1 \) requires the attacker to guess only \( x_{d-1}' \) to construct \( X' \) for the modified message. For the modified message \( M' = (m_0, \overline{m}_1, \ldots, m_{l-1}) \), the probability that \( X(M') \) will be exactly the same as \( X(M_l) \) is therefore 0.5. Similarly, for a message \( M'' \) modified in the first two bit positions, \( M'' = (\overline{m}_0, m_1, \ldots, m_{l-1}) \), the probability that \( X(M'') \) collides with \( X(M_l) \) is also 0.5. In general, if we flip bit \( m_i \) and any of the bits up to \( m_i \) in the original message, for \( 0 \leq i \leq d - 1 \), then the probability of collision is \( 2^{-i} \).

Consider now the case where \( R_0 \) is known and of the form \((1, 0, \ldots, 0)\). Then all elements in the first column of matrix \( T_i \) are zero except \( r_0[0] = 1 \). Now \( m_0 \), the first bit of \( M_l \), affects only bit \( x_0 \) of \( X(M_l) \). Thus it is possible to modify \( M_l \) by flipping \( m_0 \) and to provide a valid \( X' \) for \( M'_l = (\overline{m}_0, m_1, \ldots, m_{l-1}) \), by flipping \( x_0 \). That is, \( X' = (x_0, x_1, \ldots, x_{d-1}) \). Thus an attacker can produce a valid MAC for the forged message \( M' \) with probability of 1.

Similarly, for \( R_0 = (1, 1, 0, \ldots, 0) \), all elements of the first column are zero except \( r_0[0] \) and \( r_0[1] \), and in the second column all elements are zero except \( r_0[1] \) and possibly \( y_0 \). Thus message bit \( m_0 \) affects positions \( x_0 \) and \( x_1 \) of \( X(M_l) \), while message bit \( m_1 \) affects \( x_0 \) and possibly \( x_{d-1} \). Modifying \( M_l \) by flipping both \( m_0 \) and \( m_1 \) requires the attacker to flip only \( x_1 \) to form \( x'_1 \) and to guess \( x_{d-1}' \). Therefore, the probability that an attacker can construct a valid vector \( X' \) for \( M'' = (\overline{m}_0, m_1, \ldots, m_{l-1}) \) is 0.5.

In general, for any known \( R_0 \), if we flip bit \( m_i \) and any of the bits up to \( m_i \) in the original message, for \( 0 \leq i \leq d - 1 \), then we can construct a valid \( X' \) for this modified message \( M' \) by flipping the required bits in \( X \) and guessing the final \( i \) bits in vector \( X' \). Therefore, the probability that an attacker can construct a valid vector \( X' \) for \( M' \) is \( 2^{-i} \).
Resistance against this type of forgery attack can be provided in two ways. Firstly, we could initialise register $R$ using key dependent values, such as a segment from the keystream sequence, $y$. Alternatively, the message $M_l$ may be padded by concatenating with a segment of all ones, so that $M_p = (1, 1, \ldots, 1) || M_l$. The padding should consist of at least $d$ ones so that all message bits affect all bits in $X$, and hence all bits in the final MAC tag, in an unpredictable manner. Note that a key dependent initialisation of $R$ may be the more efficient approach, as padding the message increases both the length of the message and the size of the matrix $T$, requiring at least $(d + l)$ operations to generate the final MAC tag.

**Bit deletion forgeries**

Suppose we modify $M$ by deleting $m_0$ to obtain $M'_l = (m_1, m_2, \ldots, m_{l-1})$. Then the matrix $T_{l-1}$ for $M'_l$ is just the matrix $T_l$ for $M_l$ without the last column of $T_l$. Note in Equation 6.4, for $1 \leq i \leq d - 1$, that row $i$ in matrix $T$ is row $i - 1$ shifted one position to the left, and with a new value as the final element in the row. Applying this to $X = T_l M_l$ and $X' = T_{l-1} M'_{l-1}$, it follows that $x_i = r_0[i] m_0 + x'_{i+1}$, for $0 \leq i \leq d - 2$.

Now consider the case where $R$ is initialised with zero values. Then, all elements in the first column of matrix $T_l$ are zero and hence $x_i = x'_{i+1}$ for $i = 0, \ldots, d - 2$; that is, $X' = (\beta, X \gg 1)$ for some unknown $\beta$. We call this the sliding property of the product $T_l M_l$. An attacker can guess $\beta$, and hence provide a valid $X'$ for $M'$ with probability of 0.5. Similarly, an attacker can form a message $M''$ by deleting $i$ bits from the beginning of the message $M_l$, and obtain the new $X''_{l-1} = (\beta_0, \beta_1, \ldots, \beta_{i-1}, X \gg i)$, where the bits $\beta_0, \beta_1, \ldots, \beta_{i-1}$ must be guessed by the attacker. The attacker will provide a valid $X$ for $M''_{l-1}$ with probability $2^{-i}$, for $1 \leq i \leq d$. Note that for $i = d$, this is effectively a brute force attack on $X$, so this attack is only effective for deletion of up to the first $d - 1$ bits of the message. This attack can also be adapted for the case when $R_0$ is non-zero but known; all that is required is to flip appropriate bits of $X$, as described earlier in this section, before shifting $X$ and guessing $\beta_0, \beta_1, \ldots, \beta_{i-1}$.

Suppose now that $R_0$ is unknown but that the first $j$ bits of the message are known to be zeroes. These bits do not cause any keystream to be accumulated into register $A$, so we can again delete the first $i \leq j$ bits of the input message, shift $X$ by $i$ times and guess $\beta_0, \beta_1, \ldots, \beta_{i-1}$ to get a valid $X'$ for the modified message. The attacker will again provide a valid $X'$ for $M'_l$ with probability
2^{-i}, for 1 \leq i \leq d.

The attacks discussed above can all be prevented by either padding the message at the start with at least \( d \) ones or by initialising \( R \) with unknown bits and padding the message at the start with a single one.

Now suppose that we try deleting bits from the end of a message. Assuming that \( l > d \), if we delete the last bit of \( M \), then
\[
x_i = x'_i + y(l-d+i-1) m_{l-1},
\]
for \( 0 \leq i \leq d - 1 \). If \( m_{l-1} = 1 \), the second term is unknown since it involves keystream bits and the attacker must guess these elements of the keystream. Thus it is not feasible to obtain a forgery in this case. However, if \( m_{l-1} = 0 \), then \( X' = X \), and the attacker can forge the MAC tag for the new message \( M' \) with probability 1. Similarly, if the final \( j \) bits of the input message are known to be zero, then deleting these bits will not change the final content of the accumulation register and hence \( X' = X \) and a forgery is again possible. Such forgeries can be prevented by padding the message with a final one since this is equivalent to having a message of length \( l + 1 \) with \( m_{l+1} = 1 \).

### Bit insertion forgeries

Suppose we modify \( M_l \) by appending additional zero bits to the end of the message to obtain \( M'_{l+n} = M || (0, \ldots, 0) \). In our matrix representation, adding \( n \) zeroes to the end of \( M_l \) requires adding \( n \) columns to matrix \( T \). During the accumulation process, regardless of the values in these columns, multiplying by the additional message bits of value zero does not change the value of \( X \). Hence
\[
X(M'_{l+n}) = X(M_l), \text{ for any } n > 0.
\]
Again, this collision leads to an obvious forgery attack which succeeds with probability 1.

Now consider the effects of modifying \( M_l \) by inserting an additional bit of value 1 at the end of the message. During the accumulation process, the additional column of \( T \) will be multiplied by the additional message bit of value 1 and this may change the values in \( X \). An attacker must guess \( d \) elements in that additional column in order to calculate a valid \( X \). So the probability of obtaining a valid \( X \) for this modified message is the same as the probability of brute force attack on the MAC. Therefore forgery attacks consisting of inserting zero bits at the end of the message can be prevented if we pad the message with a bit of value 1 at the end. An equivalent solution, which we discuss in the following section, is to use a masking term that depends on the message length.

Now suppose we modify \( M_l \) by adding a zero bit to the beginning of the
message; that is, $M_{l+1} = 0 || M$. From the structure of $T_l$ and $T_{l+1}$, it follows that $x'_i = r_i(0) + x_{i+1}$, for $0 \leq i \leq d - 2$. That is $x'_i = x_{i+1}$ for $i = 0, \ldots, d - 2$, so $X' = (X \ll 1, \alpha)$ for some unknown value $\alpha$. An attacker can guess $\alpha$, and hence provide a valid $X'$ for $M'$ with probability of 0.5. Similarly, an attacker can form a message $M''_{l+i}$ by adding $i$ zeroes to the beginning of message $M_l$, and obtain the new $X''_{l+i} = (X \ll i, \alpha_0, \alpha_1, \ldots, \alpha_{i-1})$, where the bits $\alpha_0, \alpha_1, \ldots, \alpha_{i-1}$ must be guessed by the attacker. The attacker will provide a valid $X$ for $M''_{l+i}$ with probability $2^{-i}$, for $1 \leq i \leq d$. This is the basis of the previously reported attack on 128-EIA3 version 1.4 [44].

If $R$ is initialised with zeroes, then this attack will work for inserted bits of either value, since the first $i$ bits of $M''_{l+i}$ are all multiplied by zeroes in the first $d - i + 1$ rows of $T_{l+i}$. Therefore, the inserted bits affect only the last $i - 1$ bits of $X''_{l+i}$, which are bits that must be guessed anyway. Further, this forgery can again be adapted to the case of $R_0$ known (but not necessarily zero), since the effects of any inserted bits of value 1 can be determined and allowed for in applying the attack.

From the above discussion, it follows that attacks involving the insertion of bits at the start of a message can be prevented by ensuring that $R_0$ is initialised with unknown values (keystream) and that the start of the message is padded with at least one bit of value 1.

### 6.3.2 Security considerations for masking vector $A_0 \oplus F$

The forgeries discussed in Section 6.3.1 can all be prevented by suitable choices of $R_0$ and of message padding; specifically, by initialising $R$ with keystream bits and padding the message with a bit of value 1 at both ends. The masking vector $A_0 \oplus F$ provides an alternative method of preventing many of these forgeries. We now discuss the security implications associated with various options for this term. If this term is to contribute to the security of the MAC, it is important that its contents are unknown to the attacker. Therefore at least one of $A_0$ and $F$ must be sourced from keystream.

If $R_0$ is known and there is no message padding, the accumulation term $X = T_l M_l$ is vulnerable to attacks involving insertion or deletion of zeroes at the end of the message and to insertion or deletion of bits at the start of the message. (If $R_0$ is unknown, only the attacks involving insertion or deletion of zeroes apply.) Forgeries involving insertions or deletions at the start of a message
6.3. Security analysis against forgery attacks

rely on the sliding property of $T_i M_i$ described in Section 6.3.1. These can be prevented by using an appropriate mask. It is important that changes in the length of the message do not result in corresponding changes in the position of the mask bits. Otherwise the sliding property in the accumulation process will apply to the MAC as a whole. The easiest way to satisfy this requirement is to initialise $A$ with bits from a fixed position such as the start of the keystream sequence $y$.

Conversely, forgeries involving zeroes inserted or deleted at the end of the message rely on the fact that the additional bits have no effect on the accumulated value $X$. Such forgeries can be prevented by using an unknown mask that depends on the message length, for example by populating $F$ with a sequence of keystream bits starting at a fixed distance from the last bit used in the accumulation process. Note that padding the message with a final 1 is equivalent to including $F = (y_{l-d+1}, \ldots, y_l)$ in the final mask. That is, it is equivalent to starting $F$ at a position $d-1$ bits before the last bit ($y_{l-1}$) used in the unpadded case.

Together, the choices for $A_0$ and $F$ discussed above provide an effective alternative to message padding as a means of preventing bit insertion and deletion attacks. Note, however, that the masking term $A_0 \oplus F$ cannot prevent attacks based on flipping bits of the message.

6.3.3 Security analysis of existing ciphers

In this section, we show how the previous attack on version 1.4 of 128-EIA3 works using our model and then we extend this attack to a new attack on the same version of this algorithm. After this, we investigate the security provided by the existing ciphers that follow this model.

**Previous attack on 128-EIA3 version 1.4**

Recall that for 128-EIA3, $A_0 = (0, \ldots, 0)$, so the masking value is merely the value of $F$, that is, the 32 consecutive bits of $y$ starting a fixed distance after the last bit used in the accumulation process. Any increase or decrease in the message length therefore causes a corresponding shift in the keystream bits from $y$ used to form $F$.

Now consider inserting a zero at the start of the message. We noted in Section 6.3.1 that the result $X'$ of the accumulation process for this modified message is
related to the original value of $X$ by the sliding relationship $X' = (X \ll 1, \alpha)$. Since $F$ also slides by one bit when a zero is appended to the message, the entire MAC has this sliding property, as noted in [44]. The same process also applies for inserting $i < d$ zeroes at the start of the message: a forgery can be obtained with probability $2^{-i}$.

**New attack on 128-EIA3 version 1.4**

We extend the previous bit insertion attack on version 1.4 of 128-EIA3, to provide a bit deletion attack in the case where the message starts with one or more zeroes. As discussed in Section 6.3.1, if we delete $i$ of these zeroes, (where $i < d$), the result $X'$ of the accumulation process for this modified message is related to the original value of $X$ by the sliding relationship $X' = (\beta_0, \ldots, \beta_{i-1}, X \gg i)$. Since $F$ also slides by $i$ bits when these zeroes are deleted, the entire MAC has this property and a forgery again results with probability of $2^{-i}$.

**128-EIA3 version 1.5**

The modification to the starting position of the 32-bit segment of $y$ used to form $F$, introduced in version 1.5 of 128-EIA3, breaks the sliding property on $F$ and provides effective resistance to the types of forgery just discussed. Resistance to bit flipping forgeries is provided by initialising $R$ with keystream, and padding the end of the message prevents bit insertion or deletion forgeries at the end of the message.

**Grain-128a**

The cipher Grain-128a [1], described in Section 6.2.3, initialises both registers $R$ and $A$ with keystream sequence. Initialising register $R$ with keystream prevents forgery attacks due to bit flipping. Initialising register $A$ with keystream prevents forgeries involving insertion or deletion of bits at the start of the message by breaking the sliding property in the accumulation phase. Padding at the end of the message by one bit of value 1 also prevents bit insertion and deletion forgeries at the end of the message.
6.4 Security analysis against timing and power attacks

Sfinks

For Sfinks stream cipher [18], described in Section 6.2.4, padding is performed by prepending $2d$ ones to $M_l$ to initialise both registers $R$ and $A$ with keystream sequence. However, padding by $d$ bits is sufficient to ensure the register $R$ is loaded with keystream bits before $M_l$ is processed. Using this method of initialisation prevents forgery attacks due to flipping, insertion or deletion of bits at the start of the message. Sfinks uses confidentiality sequence $z$ started at $z_l$ for 64-bits as a final mask $F$. Using a sequence related to the length of the input message prevents bit insertion or deletion forgeries at the end of the message.

6.4 Security analysis against timing and power attacks

In the accumulation phase of this model, the input message acts as a control to determine which states of register $R$ are accumulated into $A$. This control mechanism is represented by the scalar multiplication $\otimes$ operation, as shown in Figure 6.1. An intuitive and efficient way to implement the accumulation phase is to use branching statements that are conditional on the input message bit at each clock. If the input bit at time $t$ is 1, then $R_t$ will be accumulated into the accumulation register $A$ using XOR operation. If the input is 0, we do not need to accumulate into register $A$, but only update register $R$ which is updated using Equation 6.1.

Under this scenario, there will be differences in computation times across the clocks in the accumulation phase, dependent on the individual bits of the input message. This opens up an avenue for timing attacks, whereby the attacker can learn whether a message bit is 1 or 0 at each clock by monitoring the time taken for the accumulation. The message can then be recovered bit by bit for as long as the cipher is monitored. Simple power analysis is also a possible attack method, since state accumulation at clocks where the input bit is 1 would have a specific power consumption profile.

The publicly available reference implementations of Sfinks [18] and 128-EIA3 [38, 40] are implemented with branching statements in the accumulation phase. These implementations are susceptible to the above side-channel attacks, which raise security concerns if used in actual applications.
Side-channel attacks on this model are based on the different timing and power consumption for different clocks when the message is accumulated in this model. Instead of using the model implied by Figure 6.1, we suggest the approach indicated in Figure 6.6 as a way to prevent these types of attacks. In presenting this proposal, we assume that an attacker cannot monitor the individual components within the system, but can only obtain information on the system as a whole. If the attacker could monitor individual registers, then he will know when registers $A_0$ and $A_1$ are updated and could utilise this information to infer the input message.

As shown in Figure 6.6, we have two accumulation registers $A_1$ and $A_0$ of the same size ($d$ stages). Then, if the input message bit has the value of 1, register $A_1$ is updated using Equation 6.2, otherwise the second register $A_0$ will be updated using the same Equation (6.2). In this case, we have the same timing clocks for all the input message bits and the same power consumed but the cost of this modification is that one register is added to the model and an extra accumulation (XOR) operation is required on average at every second step.

To generate a MAC tag using this modified model, the input message is processed as explained above, and then the final content of register $A_1$ is treated like register $A$ in the old model, and the finalisation phase applied to generate the MAC. The final content of register $A_0$ could be discarded. Alternatively it
could be rotated or permuted and used as a final mask to generate a MAC tag.

## 6.5 Summary and conclusion

The MAC tag provides a mechanism for message authentication; one of the two security services provided by AE stream ciphers. In this chapter, we describe a general matrix based model for generating MAC tags using stream ciphers by injecting the input message, either plaintext or ciphertext, indirectly. We outline the options available for various phases in this model and provide an initial security analysis of this model in terms of forgery attacks and some forms of side-channel attacks. Based on these analyses, we provide recommendations for achieving greater security in MAC designs which use this model. We also discuss several existing authentication algorithms that can be described by this model, namely 128-EIA3, Grain-128a and Sfinks.

The security analysis in Section 6.3 highlights the importance of initialising register $R$ with keystream bits. This prevents bit flipping forgeries and reduces the scope of the forgeries involving insertion or deletion of bits at the start of the message. Prepadding the message with at least $d$ ones is a feasible but arguably less efficient alternative.

To prevent the remaining bit insertion and deletion forgeries, both of the following practices must also be adopted:

1. The message is padded with a 1 at the start and/or register $A$ is initialised with keystream (from a fixed location in the keystream sequence).

2. The message is padded with a 1 at the end and/or the final mask $F$ comprises a keystream sequence that depends on the length of the message.

When implementing the cipher, we recommend an implementation that consumes the same amount of time and power for each accumulation step, so that this does not differ depending on the value of the input message bit. Such an implementation will prevent side channel attacks such as timing attacks, which could otherwise be utilized to find information related to the key used or the plaintext message. Such an implementation may be achieved at the cost of additional storage registers, and a slight reduction in throughput as shown in Figure 6.6.
Chapter 6. General model for MAC generation using indirect injection
Chapter 7

Conclusions and future research

Authenticated Encryption (AE) is a recently proposed cryptographic process for providing simultaneous confidentiality and integrity protection to input messages. AE is potentially more efficient than applying a two-step process of first providing confidentiality for a message by encrypting the message and then, in a separate pass, providing integrity protection by generating a Message Authentication Code (MAC) tag. Over the last eight years, there have been several proposed stream cipher based algorithms claiming to provide AE. However, to date there has not been extensive analysis of these algorithms.

This thesis presents the results of an extensive investigation into AE using stream ciphers. The overall aim of the research was to develop a general framework for analysing AE. From the classification, we see that ETM scheme is the best order of encryption and authentication to provide AE process within one algorithm, provided that MAC algorithm is strongly unforgeable. This scheme is used by several recently proposed AE stream ciphers. We have conducted an in depth investigation into when forgery is possible in the MAC component. As part of this investigation two matrix models for MAC formation were described based on the method of injecting the input message into the internal state of the ciphers. The security analyses of these models are investigated in more detail and the overall recommendations of the MAC design in AE algorithms are given as follows:

- Use Encrypt-then-MAC scheme in the order of encryption and MAC generation
• If the design involves direct message injection into the internal state, then accumulate the input in a nonlinear manner, and initialise the internal state with secret values such as keystream.

• If the design involves indirect message injection into the internal state, then initialise both registers $A$ and $R$ with secret values such as keystream sequence and pad the input message at the end with a single bit of value 1.

• For MAC length of 64 bits or less statistical analysis should be conducted to assure that MAC tags occur with equally likely probability.

• In software implementations for indirect injection, do not use branching statements in the accumulation process, to prevent leakage of information related to the input message.

The main emphasis of this thesis is on security issues for AE algorithms but we can also make some general comments regarding efficiency issues. We note firstly that the point of using AE algorithms rather than separate operations for confidentiality and integrity is to provide these services more efficiently. Ideally, the number of operations required when using AE should be lower than for using two separate processes but even when it is similar, using AE to perform these two processes in parallel should give clear savings in time, space and power consumption. Regarding the alternative methods for MAC generation discussed in Chapters 5 and 6, namely the direct and indirect injection methods, the accumulation process for the indirect model is more efficient in speed compared with the direct injection model which requires more than one XOR process at each iteration. However, the indirect accumulation process may open an avenue to side channel attacks, as discussed previously. Also, the indirect approach always requires a dedicated register (separate from the one used in the confidentiality component) whereas an additional register is not necessarily required when using the direct approach.

This chapter is organised as follows. Section 7.1 presents a review of the contributions of this thesis in more detail. Some avenues for possible future work are explored in Section 7.2.
7.1 Review of Contributions

There are four main contributions in this thesis. These are the classifications of seven AE stream cipher algorithms based on three categories, algebraic analyses of two AE stream ciphers, a general matrix-based model for MAC generation where the input message is injected directly into the internal state of a nonlinear filter generator and a general matrix-based model for MAC generation where the input message is injected indirectly into the internal state of the cipher. The following subsections review each contribution in more detail.

7.1.1 Classifications of existing AE stream ciphers

Chapter 3 introduced two new classifications of the AE stream cipher algorithms based on the way in which the input message is injected into the internal states of the cipher and on the way of generating the keystream sequence. The first classification identified two ways of message injection into the internal states of the cipher: direct injection and indirect injection. Direct message injection means the contents of input message is accumulated into the internal state during the accumulation phase of the MAC generation. Indirect message injection means the input message is not directly accumulated into the internal state of the cipher but is used to control segments of the keystream which are accumulated into the internal state.

The second classification is based on whether the two sequences that are used for confidentiality applications and integrity applications are generated using a single key and $\text{IV}$ or using two keys and $\text{IVs}$. The chapter also described an existing classification which was described by Bellare and Namprempre [11, 12], based on the ordering of the encryption and authentication occur for the input message. Based on these three classifications, the chapter classified seven existing AE stream ciphers.

A statistical test from [53] called the repetition test is applied to examine the distribution of output MAC tags. As discussed, this test can be applied efficiently for a MAC size of 32 bits such as the UIA2 algorithm for SNOW 3G and the 128-EIA3 algorithm for ZUC. The empirical results indicate that the probability of obtaining any of the possible $2^{32}$ MAC tags for a random message is equally likely for both ciphers. The other algorithms reviewed in this thesis had a MAC tag of 64 bits. In this case, such statistical analysis while feasible
would require a very large amount of data.

7.1.2 Algebraic analysis of two AE stream ciphers

Chapter 4 presented the algebraic analyses of the confidentiality component of two AE stream ciphers, namely SSS and ZUC. These were analysed algebraically using an indirect method to establish valid relationships between the initial state bits and the output keystream bits. This chapter is divided into two parts. The first part was describing the algebraic analysis on SSS and the second part is describing the algebraic analysis on the ZUC stream cipher.

The first analyses consider the nonlinear filter (NLF) of the keystream generator of SSS as a combiner. This attack approach was applied successfully to recover the initial states of the SOBER-t32. Since SSS and SOBER-t32 have similar confidentiality structure, we also applied this attack to SSS. The chapter described the process of the algebraic analysis on SSS including the construction of equations using the indirect method for the NLF of the cipher. The chapter concluded that using a key-dependent SBox in the NLF prevents this sort of algebraic attack on SSS, and that the use of a nonlinear update function for the state register also contributes to its resistance to algebraic attacks.

Previously, there were two independent evaluation reports [24, 67] which include algebraic analyses on the ZUC stream cipher. These analyses used direct method to establish a system of equations for NLF of the ZUC stream cipher. This chapter presented an indirect method to construct equations between the initial states of the cipher and the keystream outputs. The analyses consider the NLF as a combiner and attempts to construct a matrix for this combiner that relates the inputs of the combiner with the output. It was shown that the indirect method of obtaining the equations to be used in an algebraic attack offers a significant improvement over the direct method. However, the complexity of the indirect method is still significantly worse than exhaustive key search.

7.1.3 General model for MAC generation using direct injection of the input message

In Chapter 5, a new general matrix based model for MAC generation using direct message injection into the state of a nonlinear filter generator is proposed. The proposed model is based on the way the input message is injected into
7.1. Review of Contributions

the internal states of the stream cipher. This chapter examined two different options for injecting the input message directly into the internal state of the cipher. These two options are injecting the input message and/or filter output by XORing with the previous content of the stages or by replacing the content of those stages with the new content.

Nakano et al. [76] proposed two ways of injecting a message into the internal state of a binary nonlinear filter generator for generating a hash value. The first way is to inject the output of XORing the input message bit, filter output bit and the feedback bit into last stage of the register. The second way is to inject the message bit and/or filter output into selected regularly spaced stages of the register. This chapter extended that work to include word based registers as well as bit based registers and to generate a MAC tag instead of hash value of the input message.

Nakano et al. found that it is easy to force collision using the first method of injection, but that for the second method, the probability of collision is determined by birthday attack on the remaining stages and is not affected by the input message. Our analyses of this model shows that for the first method of injection, the results confirm the Nakano et al. conclusion that the full register can be manipulated and hence it is easy to force a collision. The results also showed that for the second method of injection, the full register can be manipulated by suitable modification of the input message. This demonstrated that the analysis from Nakano et al. was flawed in this case. The only constraint on obtaining a collision by using this process is that the length of the input message must be greater than \( d \) bits.

The chapter also analysed several AE stream ciphers that follow this model in the accumulation process to generate a MAC tag. These ciphers are SSS, NLSv2 and SOBER-128. For the first two ciphers, the accumulation process is linear and so it is easy to force a collision on the internal states in the accumulation process for two distinct messages. However, the two ciphers used additional different mechanisms in forming the MAC, so are able to prevent this sort of collision. For SSS, it is self-synchronous and for NLSv2, it uses another nonlinear register to accumulate the input message in parallel with the linear register. For SOBER-128 this type of forgery attack is not possible, since the accumulation method is not linear. It should be noted that there is a forgery attack on SOBER-128 [98] but the success of this attack is due to the weakness of the nonlinear filter and
not due to the method used for accumulating the input message.

Based on the analyses, the chapter presented several procedures that could be used to prevent collision attacks when using this model in the accumulation process to generate a MAC tag. These are to use a secure nonlinear filter in the accumulation process of the input message, to initialise the internal state of the LFSR with key-dependent secret values and, when using AE, to accumulate plaintext instead of ciphertext.

7.1.4 General model for MAC generation using indirect injection of the input message

Chapter 6 presents a new general matrix based model for the generation of a MAC tag in AE stream ciphers which use the input message indirectly. The three phases of the MAC generation using this model are the preparation phase, where the input message and the two registers are prepared, the accumulation phase where the keystream sequence is accumulated into the internal state of the cipher based on the value of the input message and the finalisation phase where the final content of the accumulation register is XORed with a final mask. Since the accumulation phase is the same in each case, this chapter examined different options used in the preparation and finalisation phases.

Some current stream ciphers which can be considered as instances of this model with different choices of options were described in this chapter. These ciphers are 128-EIA3, which is based on ZUC stream cipher, Sfinks and Grain-128a. Each cipher was described using matrix form for the final value of the MAC tag.

Security analyses of our model against forgery and side-channel attacks are presented in this chapter. Selecting proper initialisation options can prevent forgery attacks based on flipping, deletion and insertion of bits in the original message.

To prevent side-channel attacks when implementing the cipher, the implementation should consume the same amount of time and power for each accumulation step, so that this does not differ depending on the value of the input message bit. A recommended structure for achieving this aim was proposed and discussed.
To date (including also the research in this thesis) the confidentiality and integrity components of AE proposals have been analysed separately. A future area of investigation would be to investigate both components jointly, i.e. examine whether an attack identified in one component would lead to a feasible attack in the other component.

In the case of the algebraic analyses on the SSS and ZUC confidentiality algorithms, one possible extension is to investigate a direct relation between the initial state and the keystream output. Such a relation may possibly lead to a method to construct a system of equations and have a solution with complexity less than brute force attack on the key space.

The model for direct injection to form a MAC in Chapter 5 only describes the accumulation process of the input message. The other two MAC generation phases are not investigated yet. This model also considers only a nonlinear filter generator for its MAC accumulation component. One future direction for research is to investigate the other two phases of MAC generation within this model and to analyse the security for all phases against side-channel attacks. Also, it is worth investigating different types of keystream generators such as a nonlinear combiner with more than a single LFSR in this model.

A final future research direction would be to investigate the design of AE algorithms based on the results of this thesis. The aim would be to design AE algorithms which are both efficient and secure. Some general efficiency issues have been discussed briefly above, but further investigation is required in order to find more detailed results in the case of specific ciphers. Potential trade-offs between security and efficiency of AE algorithms also needs further investigation.
Bibliography


[16] Alex Biryukov and Adi Shamir. Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers. In In Tatsuaki Okamoto, editor, Advances in


