

# Parametric Guess and Determine Attack on Stream Ciphers

A thesis submitted to the  
Graduate School of Natural and Applied Sciences

by

Ebru Küçükkubaş

in partial fulfillment for the  
degree of Master of Science

in

Cybersecurity Engineering

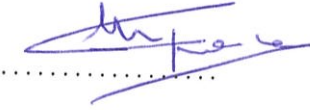
This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science in Cybersecurity Engineering.

**APPROVED BY:**

Prof. Ensar Gül  
(Thesis Advisor)

  
.....

Assoc. Prof. Orhun Kara  
(Thesis Co-advisor)

  
.....

Prof. Erkay Savaş

  
.....

Assoc. Prof. Alptekin Küpçü

  
.....

Asst. Prof. Erdiñç Öztürk

  
.....

This is to confirm that this thesis complies with all the standards set by the Graduate School of Natural and Applied Sciences of İstanbul Şehir University:

DATE OF APPROVAL: 17.07.2015

SEAL/SIGNATURE:



# Declaration of Authorship

I, Ebru Küçükkubaş, declare that this thesis titled, 'Parametric Guess and Determine Attack on Stream Ciphers' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: \_\_\_\_\_



Date: \_\_\_\_\_

17.07.2018

# Parametric Guess and Determine Attack on Stream Ciphers

Ebru Küçükkuş

## Abstract

The need for lightweight cryptography for resource-constrained devices gained a great importance due to the rapid evolution and usage of IoT devices in the world. Although it has been common in the cryptology community that stream ciphers are more efficient in speed and area than symmetric block ciphers, it has been seen in the last 10-15 years that most of ciphers designed for resource-constrained devices to take up less area and less energy on hardware-based platforms, such as ASIC or FPGA, are lightweight symmetric block ciphers.

On the other hand, the design and analysis of stream ciphers using keyed internal update function is put forward against this belief and it has become one of the popular study subjects in the literature in the last few years. Plantlet, proposed in 2017, its predecessor Sprout, proposed in 2015 and Fruit proposed in 2016, are famous algorithms as instances of stream ciphers using keyed internal update function. Sprout was broken after a short time by many researchers but Plantlet hasn't been successfully broken yet and there has been only one attack mounted on Fruit since it was proposed.

Traditionally, key stream generators of stream ciphers update their internal states only by using their current internal state. Since the use of the key in the internal update is a new approach, the security analysis of this approach is not fully understood. In this study, the security analysis of the key stream generators with keyed update function has been studied. A new attack algorithm for internal state recovery and key recovery has been developed and mounted on Plantlet algorithm as an instance of stream ciphers with keyed update function. The state bits and key bits are successfully recovered. In the second phase, the attack algorithm was mounted on Fruit algorithm and state bits and key bits are also recovered successfully.

**Keywords:** Stream Ciphers, lightweight, Grain family, Sprout, Plantlet, Fruit

# Dizi Şifreleme Algoritmaları için Parametrik Tahmin Et ve Belirle Saldırısı

Ebru Küçükkubaş

## ÖZ

Dünyadaki IoT cihazlarının hızlı evrimi ve kullanımı nedeniyle, kaynak kısıtlı cihazlar için hafif sıklet kriptografi ihtiyacı büyük önem kazanmıştır. Dizi şifreleme algoritmalarının, özellikle belli platformlarda daha hızlı çalışmaları ve ya daha az yer kaplamaları açısından blok şifreleme algoritmalarına nazaran daha verimli olduğu konusunda kriptoloji camiasında oluşmuş ortak bir kanı olsa da son 10-15 yılda tasarlanmış blok şifreleme algoritmaları bu kanıyı yıkacak niteliktedir. Özellikle ASIC ya da FPGA gibi donanım tabanlı platformlarda az yer kaplayacak ya da az enerji harcayacak şekilde tasarlanmış simetrik şifreleme algoritmalarının birçoğunun blok şifreleme algoritmaları olduğu görülmektedir.

Diğer taraftan bu kaniya aykırı olacak şekilde ortaya atılan anahtarlı içsel durum güncelleme tekniğiyle kayan anahtar üreci kullanan dizi şifreleme algoritmalarının tasarımı ve analizi literatürde son birkaç yıl içinde popüler çalışma konularından birisi olmuştur. 2015'te yayınlanan Sprout algoritması ve 2017'de yayınlanan Sprout'un üst versiyonu olarak tasarlanmış Plantlet algoritması ve ve 2016 yılında yayınlanan Fruit algoritması anahtarlı içsel durum güncellemesi yapan dizi şifreleme algoritmalarının ünlü örnekleridir. Sprout yayınladıktan kısa bir süre sonra birçok araştırmacı tarafından farklı kriptanaliz metodlarıyla kırılmıştır ancak Plantlet algoritması henüz başarılı olarak kırılmamıştır. Fruit algoritmasına da yayınladığından beri bir adet atak yapılmıştır.

Genellikle dizi şifreleme algoritmalarının kayan anahtar üreteçleri içsel durumlarını sadece mevcut içsel durumlarını kullanarak güncellemektedir. Anahtar kullanımı ile içsel durum güncelleme yeni bir yaklaşım olması nedeniyle, bu yaklaşımın güvenlik analizleri tam olarak olgunlaşmamıştır.

Bu tezde anahtar kullanımı ile içsel durum güncellemesi yapan kayan anahtar üreteçlerinin güvenlik analizi çalışılmıştır. Yapılan analizin literatürdeki belirli algoritmalara uygulanması çalışmaları yapılmıştır. Bu kapsamda içsel durum ve anahtar elde etme için genel bir atak algoritması geliştirilmiş ve bu atak algoritması örnek olarak Plantlet ve Fruit algoritmalarına uygulanmış, içsel durum ve anahtar bitleri elde edilmiştir.

**Anahtar Kelimeler:** Dizi Şifreleme, Grain Ailesi, Sprout, Plantlet Algoritması, Fruit Algoritması



*This thesis is dedicated to my family and my son.*

# Acknowledgments

I would like to express my gratitude to my thesis co-supervisor Assoc. Prof. Orhun Kara for his guidance and constant encouragement. I am very grateful to him for his scientific advice, knowledge, and many insightful discussions and suggestions. It has been an honor to work with him.

I would like to thank to the rest of my thesis committee: Prof. Ensar Gül, Prof. ErKay Savaş, Assoc. Prof. Alptekin Küpçü, Asst. Prof. Erdiñ Öztürk for their encouragement, insightful comments and hard questions.

I would also thank to TÜBİTAK BİLGEM management providing the opportunities to attend this MS program and TÜBİTAK BİLGEM UEKAE for its opportunities and contribution on my experience on cryptography that let me complete my MS study.

Finally, I would like to thank my family and son. This thesis, indeed, is the result of their efforts. This thesis would never be completed without their support.

# Contents

<b>Abstract</b>	<b>iii</b>
<b>Öz</b>	<b>iv</b>
<b>Acknowledgments</b>	<b>vi</b>
<b>List of Figures</b>	<b>ix</b>
<b>Abbreviations</b>	<b>x</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Related Work . . . . .	3
1.2 Contributions . . . . .	4
1.3 Outline . . . . .	6
<b>2 Preliminaries</b>	<b>8</b>
2.1 Stream Ciphers . . . . .	8
2.1.1 Types Of Stream Ciphers . . . . .	9
2.1.2 Keystream Generator Internal Structure . . . . .	10
2.1.3 Shift Register Based Stream Ciphers . . . . .	11
2.1.4 Stream Cipher Attack Models . . . . .	13
2.1.5 Basic Attacks To Stream Ciphers . . . . .	14
2.2 Grain Family . . . . .	15
2.2.1 History of Grain Family . . . . .	15
2.2.2 Grain Family Structure . . . . .	15
2.2.3 Grain Family Design Criteria/Choices . . . . .	17
<b>3 Sprout</b>	<b>19</b>
3.1 Sprout . . . . .	19
3.1.1 Keystream-equivalent states . . . . .	20
3.1.2 Keystream Generator With Keyed Update Function . . . . .	21
3.1.3 Sprout Structure . . . . .	22
3.1.4 Guess-and-Determine Attacks Against Sprout . . . . .	24
<b>4 Plantlet and Fruit</b>	<b>26</b>
4.1 Plantlet . . . . .	26
4.1.1 Plantlet Design Goals . . . . .	26
4.1.2 Planlet Specification . . . . .	26
4.1.3 Planlet Design Rationale . . . . .	28



---

4.2	Fruit . . . . .	29
4.2.1	Fruit Specification . . . . .	29
4.3	Guess Capacities of Plantlet and Fruit . . . . .	31
<b>5</b>	<b>New and Parametric Guess and Determine Attack</b>	<b>33</b>
5.0.1	New Guess and Determine Attack Mounted On Plantlet . . . . .	33
5.0.2	Parametric Guess and Determine Attack Mounted On Plantlet . . . . .	35
5.0.3	Improving New Guess and Determine Attack Through Trade-Off . . . . .	40
5.0.4	New and Parametric Guess and Determine Attacks Mounted On Fruit . . . . .	43
<b>6</b>	<b>Conclusion</b>	<b>46</b>
	<b>Bibliography</b>	<b>48</b>



# List of Figures

2.1	Stream Ciphers . . . . .	9
2.2	Synchronous Stream Ciphers . . . . .	9
2.3	Self-synchronizing Stream Ciphers . . . . .	10
2.4	LFSR . . . . .	12
2.5	GSM A5/1 Structure . . . . .	13
2.6	Grain Family Structure . . . . .	16
2.7	Grain Key and IV Initialization . . . . .	16
3.1	Key Stream Equivalent States . . . . .	20
3.2	Keystream Generator with Fixed Internal State Parts . . . . .	20
3.3	Keystream Generator with Keyed Update Function . . . . .	21
3.4	Sprout . . . . .	22
3.5	Round Key Function . . . . .	23
3.6	Initialization Phase of Sprout . . . . .	24
4.1	Plantlet Structure . . . . .	27
4.2	Fruit Structure . . . . .	30

# Abbreviations

**ASIC**    Application Specific Integrated Circuit

**FPGA**    Field Programmable Gate Array

**IoT**      Internet of Things

**RFID**    Radio Frequency Identification

**CTR**     Counter

**OFB**     Output Feedback

**CFB**     Counter Feedback

# Chapter 1

## Introduction

As the use of the IoT devices like RFID tags, wireless sensors becomes more and more pervasive and ubiquitous, the need for exchange of confidential and sensitive data through unsecure channels such as the Internet by these resource-constrained devices and systems increases and these sensitive data become susceptible to various attacks. Since these devices are resource-constrained, conventional cryptographic algorithms are not convenient for these devices. Lightweight cryptography aims to provide algorithms for resource-constrained devices having restricted hardware environments where the power or energy consumption, gate count and the memory is limited.

Many ultra lightweight block ciphers have been developed in the last 10 decade like Midori [1], KTANTAN [2], PRESENT [3], LED [4], SIMON/SPECK [5], Simeck [6] and Piccolo [7] but ultra lightweight stream ciphers are not so easily designed because of the design principle that to achieve a  $K$ -bit security, internal state size of the cipher must be at least  $2K$  bits.

At FSE (Fast Software Encryption) 2015, Armknecht and Mikhalev proposed a new stream cipher, Sprout [8] with a novel idea for keystream generators having internal state size shorter than  $2K$  by using a fixed key in the internal state update function. They defined it as keystream generator with keyed update function. Sprout was broken after a short time by many researchers [9], [10], [11], [12], [13], so the designers of Sprout developed another algorithm, Plantlet [14] by fixing the bugs and the weaknesses of Sprout. Plantlet has been in the literature for about 2 years and there has been no successful attack proposed for Plantlet yet. Another algorithm Fruit [15] having keyed update function like Sprout

and Plantlet was proposed in 2016 and there has been only one attack on Fruit since it was proposed.

One of the attacks mounted on Sprout is internal state recovery attack by Kara and Esgin [9]. Sprout's main weakness lies on its round key function. The key bits are not always used in the internal state update function where feedback values can be determined or guessed without knowing the key bits for some of the internal states. Just after their attack on Sprout, they generalized the attack idea and introduced a new algorithm that can be successfully mounted on any keystream generator with keyed update function where the key bits are incorporated into the states during state update in a biased manner [11]. It may still be possible to guess the feedback value without knowing the key with an overwhelming probability.

They define the notion of "guess capacity" as the probability of guessing the feedback value correctly for a given internal state without knowing the key. Their generic attack is successful if the guess capacity is strictly higher than one-half. The guess capacity of Sprout is much higher than one-half because of the weak structure of round key function; incorporating the key bits into the feedback function.

After introducing the notion of guess capacity of the feedback function of a keystream generator with keyed update function, it was immediately adopted as a security criterion. Indeed, the fixed version of Sprout, Plantlet has the guess capacity of one-half. Similarly, the guess capacity of Fruit is fixed to one-half in its ultimate version. As a conclusion, the Kara and Esgin attack is applicable to neither Plantlet nor Fruit. In fact, there are no successful attacks so far on any of the both ciphers. Hence, the security analysis of the keystream generators with keyed update function having the guess capacities of one-half is an open problem in such stream cipher designs.

In this study, we proposed a new generic internal state and key recovery attack for stream ciphers with keyed update function having guess capacity one-half and applied it on Plantlet and Fruit. We used the attack developed by Kara and Esgin [11] as the starting point. Their attack was applied to Sprout but could not be applied to Plantlet because of the round key function of Plantlet. We developed a new attack using the weakness of involving the key bit directly into the internal state update function. By this new attack, we recovered key bits and internal state bits at the same time. Since the internal state size of Plantlet is greater than key size, the attack complexity is bigger

than exhaustive key search. To decrease the complexity of the attack, we made two extensions. First, we modified the new attack by using variables for some taps of the LFSR and also added another phase by solving nonlinear equations during key recovery. As a proof of concept, we tested the idea of decreasing complexity practically with six variables and successfully implemented the attack and recovered internal state and key bits. This test decreased the attack complexity but it is still slower than exhaustive key search. The attack complexity should be decreased using more variables but this would need precomputation and memory to generate and solve nonlinear equations efficiently. As a second extension to decrease the attack complexity, we combined our attack with trade-off attacks and made a generalization for the internal state size for Plantlet like ciphers to be resistant to TMDTO attacks.

In the second phase of our study, we mounted our generic attack on Fruit algorithm which has guess-capacity one-half like Plantlet. The state and key bits are also recovered successfully. Unlike Plantlet, Fruit was that it has internal state size equal to key size, 80 bits. Our attack is faster than the exhaustive key search even without using variables since initialization phase is not considered and implemented during attack which should be taken into consideration for exhaustive key search attack.

## 1.1 Related Work

Since the notion of stream ciphers with keyed update function is a new topic in lightweight cryptography, the research on stream ciphers with keyed update function are made in the last few years. [9], [10], [11], [12], [13], [16] are some examples of them.

Kara and Esgin introduced a guess and determine attack combined with a divide and conquer attack on full Sprout [9]. Lallemand and Plasencia proposed a divide-and-conquer attack for recovering the key bits of Sprout [10]. Kara and Esgin introduced generalized divide and conquer attack on stream ciphers with keyed update function. It is an internal state recovery attack and can be mounted if guess capacity of the cipher is greater than one-half [11]. Zhang and Gong introduced TMD tradeoff attack developed by for stream ciphers having shorter internal states [12]. Dey and Santanu Sarkar proposed a divide and conquer attack for cryptanalysis of full round Fruit using the weakness in round key generation [16].

## 1.2 Contributions

The main contribution of this thesis is the introducing a novel and generic internal state and key recovery attack for stream ciphers with keyed update function having guess capacity one-half. This was an open question for the security of stream ciphers having keystream generators with keyed update function.

Stream ciphers with keyed update function was born four years ago with an algorithm, Sprout [8] proposed in FSE 2015. This algorithm with the keyed update function design has attracted the attention of many researchers since using the key in the update function of internal state enables using shorter internal state size which is critical in lightweight cryptography. Sprout was broken practically after a short time [9], [11] and then the designers of Sprout fixed the bugs in Sprout and proposed another algorithm, Plantlet [14], two years later. Plantlet hasn't been broken yet successfully. The attacks mounted on Sprout couldn't be applied to Plantlet. Another algorithm Fruit [15] having keyed update function was proposed in 2016. The algorithm has internal state size like in Sprout but the authors claimed that their design rationale doesn't have the weaknesses of Sprout and there has been only one attack against [16] Fruit since it was published.

As one of the rule of thumb of a keystream generator with keyed update function, its guess capacity should be one-half in order not to be exploited by the Kara and Esgin attack [11]. We see that the ultimate versions of such designs are in compliance with this security criterion. Indeed, both Plantlet and the latest version of Fruit have guess capacities exactly one-half. There are no successful attacks mounted on them yet using guess capacity, thanks to the new criterion.

In this thesis, we study the security of Plantlet and Fruit like ciphers. We introduce the question whether divide and conquer type attacks can be mountable on ciphers with the guess capacities of one-half to recover their keys or to distinguish the correct internal state among arbitrary states.

We proposed a new generic attack and mounted it on Plantlet and Fruit. This attack can be applied to any stream cipher with keyed update function having guess capacity one-half. Guess capacity is the probability of guessing the feedback value of the internal state without knowing the key. Since the key bit is directly involved in the update function of

Plantlet and Fruit, the probability of guessing the feedback value of the internal state is one-half.

Our generic attack uses the weakness of involving key bits directly and cyclically in the update function of keystream generator. The attacker has an output stream and his aim is to recover the internal state bits which generated this output stream among arbitrary internal state candidates and recover key bits used during this generation. Using algorithm output function, feedback values of the keystream generator is either determined from output stream or guessed. In Plantlet, keystream generation is run and formulated in backward direction and feedback values are directly determined from output stream values. In Fruit, no need to formulate in backward direction since the feedback values can be directly determined from output stream. After determining the feedback values for each internal state candidate, the key bits are determined using the update function of the keystream generator for each internal state candidate. Since the key bits are used cyclically, after cycle period, key bits determined should be equal to each other as  $k_t = k_{t-period}$  for the correct internal state. Using these equalities, wrong internal state candidates are eliminated and correct internal state key bits conform this equality. Correct internal state is recovered and at the same time key bits are also determined at the end of the attack.

For internal state recovery, the attack complexity is  $2^{internalstatesize}$ . The attack algorithm should use all possible internal states to recover the correct internal state. If the internal state size is greater than key size like in Plantlet, the attack is not efficient since exhaustive key search would be preferable. We reduced the attack complexity with two novel ideas. The first one is using variables for some bits of the internal state. Since in keystream generators, internal state update and output functions are simple XOR and multiplication functions of internal state bits, using variables for some bits of internal state and running and parameterizing our generic algorithm using these variables is successful in reducing complexity. At the end of the attack, the key bits are nonlinear equations of the variables. These equations are solved for the variables and a unique solution is achieved for the correct internal state and other internal states are eliminated. After solving equations and recovering the correct internal state, the key bits are determined using the solved equations. The second idea to reduce attack complexity is to combine our algorithm with trade-off attack. Given an output stream sequence, the attacker makes a guess for internal state and using our attack, he can determine



key bits and the following output stream sequence. This can be formulized as a one-way function of internal state bits and output stream sequences. Using Hellman tables with our attack, instead of searching all  $2^{internalstatesize}$ , attack complexity is reduced to  $2^{80}$  time complexity,  $2^{61}$  memory complexity and  $2^{80}$  data complexity where the data complexity can be decreased by increasing the memory complexity. This would be the trade-off for the attacker. Using this trade-off, a generalization is made for Plantlet like ciphers where guess capacity is one-half and key cycle is equal to number of key bits as  $T(DM)^2 = (NK)^2$ . By using this formula, two corollaries are achieved. The first one is that for Plantlet like ciphers, the internal state size must be at least  $keysize * (3/2)$  to be resistant to TMDTO attacks. The second one is that using our new guess and determine attack with trade-off attack, the attack complexity for Planlet is reduced to  $2^{72.4}$ .

The contributions of the thesis are summarized as:

- Guessing and determining the feedback values and key bits of the keystream generators having guess capacity one-half.
- Designing a new guess and determine attack for internal state recovery and key recovery at the same time.
- Keystream generation in backward direction to simplify the attack.
- Using variables for internal state bits in order to decrease the complexity of the attack.
- Combining the attack with trade-off attacks to decrease complexity.
- Making a generalization for the internal state size of Plantlet like ciphers to be resistant to TMDTO attacks.

### 1.3 Outline

This thesis is organized into six chapters.

The first two chapters contain introductory information about the subject of the thesis. The third chapter contains related information about the generic attack introduced in the thesis. The fourth chapter gives specifications and the security of the algorithms we

studied for our attack. The fifth chapter give the new information about the details of the generic attack developed and the sixth chapter gives a brief conclusion of the study. Some figures are used from literature in the thesis. Their sources are cited in the caption of the figures.

The summary of chapters are:

**Chapter 1** provides brief introduction to the stream ciphers with keyed update function, the generic attack mounted on Sprout and the new successor attack mounted on Plantlet and Fruit.

**Chapter 2** gives an introduction on cryptology, symmetric ciphers, stream ciphers, types and basic cryptological concepts of stream ciphers, attacks and make an introduction to lightweight stream ciphers with Grain family. Design criteria of Grain family is discussed.

**Chapter 3** is dedicated on Sprout algorithm and guess and determine attack applied to Sprout which is the starting point of parametric guess and determine attack.

**Chapter 4** is focused on Planlet algorithm and Fruit algorithm specifications.

**Chapter 5** is focused on the new guess and qetermine attack and parametric guess and determine attack mounted on Plantlet and Fruit algorithms. An extension made to improved guess and determine attack for Plantlet like ciphers is also given.

**Chapter 6** gives a brief conclusion of the thesis.

## Chapter 2

# Preliminaries

### 2.1 Stream Ciphers

Cryptography is the science of using mathematics for data security. By cryptography, data is stored or transmitted across insecure networks so that only receiver can read it. While cryptography's aim is the data security, cryptanalysis is used to analyze and break secure communications of data. Application of mathematical tools, analytical reasoning, pattern finding, determination and patience are basic items of cryptanalysis. Cryptology comprises both cryptography and cryptanalysis.

The most common service of cryptology is confidentiality (by encryption) but it is also used for authentication, integrity, non-repudiation, anonymity, availability, privacy, etc.. Confidentiality is provided by encryption algorithms; ciphers. Ciphers are classified as symmetric and asymmetric ciphers. In symmetric ciphers, same key is used in encryption and decryption. Symmetric key ciphers are also classified as block ciphers and stream ciphers.

Stream cipher is a symmetric key cipher where a pseudorandom cipher digit stream, called keystream is combined with plaintext digits. The basic structure of encryption and decryption of stream ciphers is shown in Figure 2.1 . A secret key and a public initialization vector is shared between two parties. After an initialization stage, keystream is generated by a keystream generator and plaintext digits are XOR'ed with generated keystream forming ciphertext digits. At the receiver side same scenario is applied. The keystream bits generated by the same key stream generator are XOR'ed with the ciphertext digits

forming the plaintext digits. Use of IV is necessary, otherwise the same stream will be produced each time unless the key is not changed.

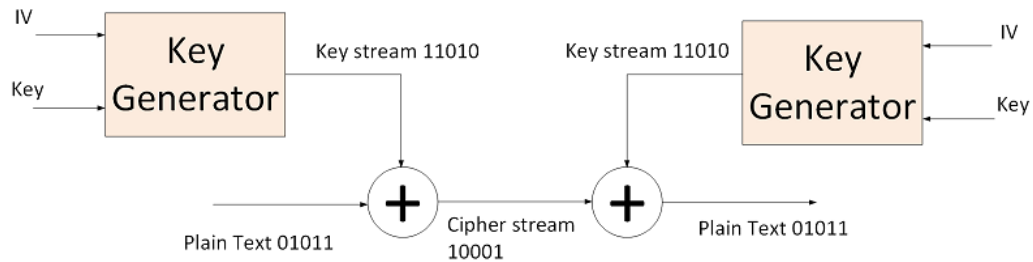


FIGURE 2.1: Stream Ciphers [17]

### 2.1.1 Types Of Stream Ciphers

Stream ciphers are classified as synchronous and asynchronous (self-synchronizing) with respect to the key stream generation.

#### Synchronous Stream Ciphers

In synchronous stream ciphers, the plaintext message and the ciphertext have no effect on keystream generation. Basic structure is shown in Figure 2.2. Changing a bit in ciphertext during transmission only affects corresponding plaintext on receiver side. There is no error propagation but inserting/deleting a bit in ciphertext during transmission affects the rest of the plaintext on the receiver side and synchronization is lost.

#### Asynchronous (self-synchronizing) Stream Ciphers

In asynchronous stream ciphers, the key and a fixed number of previous ciphertext digits are used in keystream generation. Basic structure is shown in Figure 2.3. Changing a bit in ciphertext during transmission affects some number of plaintext on receiver side.

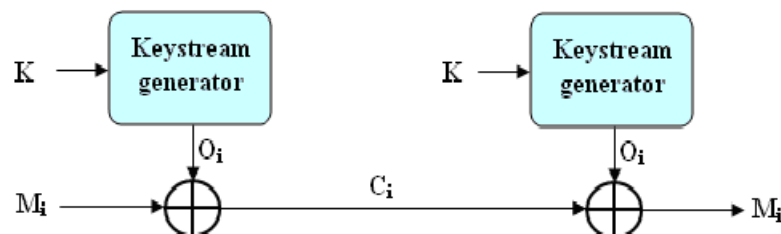


FIGURE 2.2: Synchronous Stream Ciphers [18]

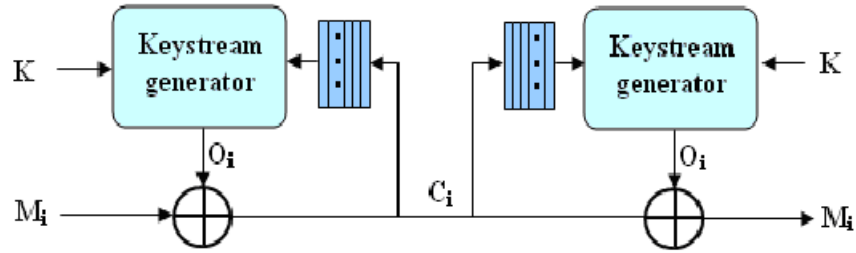


FIGURE 2.3: Self-synchronizing Stream Ciphers [19]

There is small error propagation. Inserting/deleting a bit in ciphertext will be recovered after some bits on the receiver side which implies self-synchronization.

### 2.1.2 Keystream Generator Internal Structure

Keystream generation is the focus of the stream ciphers. Designing a stream cipher consists of designing the keystream generator mainly. The formal definition of a keystream generator (KSG) is given below: A KSG is executed on three space sets;

- the key space  $K = GF(2)^k$
- the IV space  $IV = GF(2)^v$
- the state space  $S = GF(2)^s$

and consists of three functions

- an initialization function,  $\text{Init} : IV \times K \rightarrow S$
- an update function,  $\text{Upd} : S \rightarrow S$
- an output function,  $\text{Out} : S \rightarrow GF(2)$

A keystream generator operates in two phases:

- An IV and a secret key are used as inputs and the internal state is set to an initial state in the initialization phase,  $st_0 := \text{Init}(iv, k)$
- Then, the following operations are executed repeatedly in the keystream generation phase:

- Generate the next keystream bit  $z_t = \text{Out}(st_t)$
- Update the internal state  $st_t$  to  $st_{t+1} := \text{Upd}(st_t)$

Keystream generation does not directly involve key anymore in the conventional scenario. In stream ciphers, since encryption or decryption is just an XOR operation, an adversary can easily recover the keystream for the known plaintext scenario. For a stream cipher to be secure:

- Adversary can not generate the keystream in forward or backward direction or can not recover the key, if she knows some part of the keystream.
- Adversary can not recover any internal state of the cipher at time  $t$  where  $t > 0$ . If adversary recovers the state, he can generate the keystream in forward or backward direction.

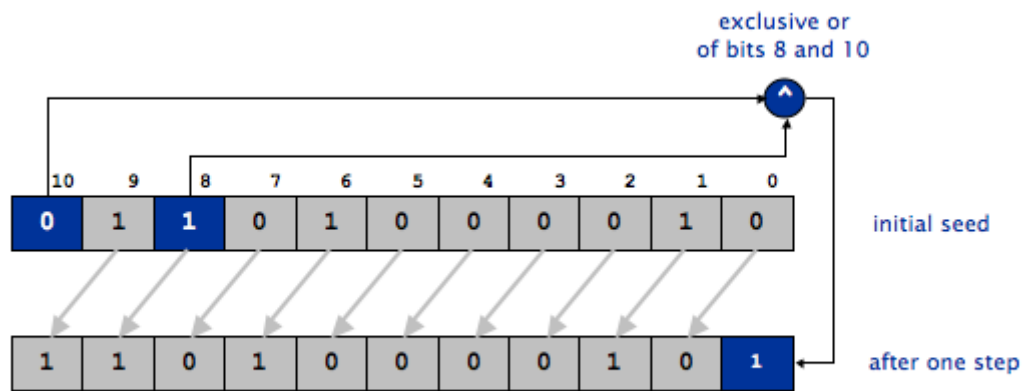
The basic expectation for a generated keystream is that using a polynomial time algorithm, the keystream must be indistinguishable from a truly random sequence. This is achieved by cryptographically secure pseudo random number generators (PRNG). Cryptographically secure PRNGs are generally implemented in two ways:

- Using block ciphers in CFB, CTR, OFB mode
- Using shift registers with feedback.

### 2.1.3 Shift Register Based Stream Ciphers

In stream ciphers, traditionally one or multiple linear feedback shift registers(LFRS) are used. An LFSR is shift register having a linear feedback function. An example LFSR is shown in Fig 2.4.

LFSRs are constructed by clocked storage elements (flip-flops) and a feedback path. Flip-flop count determines the degree of an LFSR. The input for the last flip-flop is computed by the feedback network. It is the XOR-sum of some certain flip-flops in the shift register which is called the feedback function.



One step of an 11-bit LFSR with initial seed 01101000010 and tap at position 8

FIGURE 2.4: LFSR [20]

The maximum sequence length produced by an LFSR of degree  $m$  is  $2^m - 1$  because all-zero state is excluded. Feedback function can be represented by a polynomial and for a maximum sequence, polynomial must be primitive. Feedback functions should be chosen to create maximum sequence length.

Although the sequence generated by an LFSR has good statistical properties, it is unfortunately cryptographically weak because of linearity. If an attacker knows  $2m$  output bits of an LFSR of size  $m$ , she can exactly construct the LFSR by solving a system of linear equations. It is assumed that feedback coefficients of LFSR are also not known. If feedback coefficients are known,  $m$  output bits are enough. Since using an LFSR is not a secure solution for keystream generation design in stream cipher, cipher designers used the following solutions:

1. Multiple LFSR with irregular clocking
2. Combination with nonlinear feedback shift register(NFSR) (A NFSR has a nonlinear feedback function)
3. Nonlinear output function

GSM A5/1 is an example of LFSR based stream ciphers. Its multiple LFSR with irregular clocking structure is shown in Fig 2.5. Key Generation is done with XORing of 3 LFSR outputs but at each clock three LFSRs are not clocked. Some LFSR bits are used to decide which LFSR to clock. Probability of one LFSR to clock is  $6/8$ .

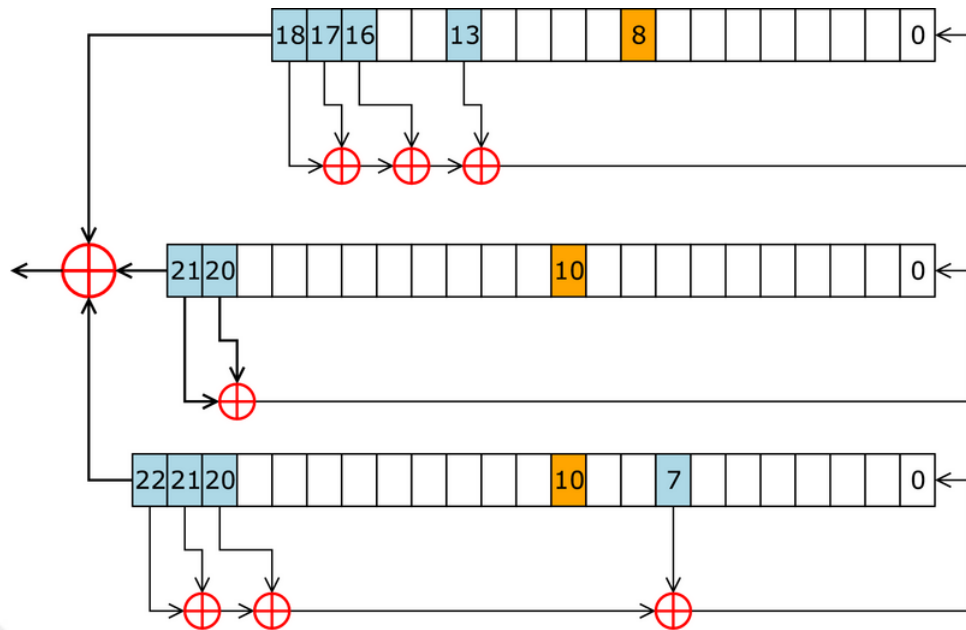


FIGURE 2.5: GSM A5/1 Structure [21]

### 2.1.4 Stream Cipher Attack Models

#### Attacker Goals

For stream cipher cryptanalysis, attacker aim is to generate the keystream. This can be done in three ways.

1. The attacker tries to recover key. This is a standard attack for all ciphers like in symmetric block ciphers.
2. The attacker tries to recover a state value at any time  $t > 0$  in order to generate the keystream in forward or backward direction. With an invertible initialisation function, she can also recover the key.
3. The attacker tries to find a distinguisher and predicts next-bit value.

#### Attacker Access

For stream cipher cryptanalysis, attacker can have access to ciphertexts, or chooses plaintexts and receives corresponding ciphertexts or chooses IVs and receives the corresponding ciphertexts.



### 2.1.5 Basic Attacks To Stream Ciphers

Using the attack models above, basic attacks [22], [23], [24], [25], [26] against stream ciphers are listed below:

- **Correlation Attack:** The attacker tries to find a statistical dependence between the keystream and the output of one of LFSRs/a linear combination of few inputs and tries to find the initial state of this LFSR independently of the other LFSRs.
- **Distinguishing Attack:** The attacker's aim is to find a distinguisher for the generated keystream from a truly random sequence and identifies the relations between internal state variables and output keystream.
- **Fault Attack:**
  1. A fault is injected and the keystream is produced.
  2. A guess is made for the effect of the fault.
  3. Guess is checked whether the guess is correct, otherwise a new guess is made.
  4. Steps 1-3 are repeated many times
  5. Linear equations are solved to find the initial state of the LFSR.
- **Guess-and-Determine Attack:** A part of the internal state is guessed and the remaining state elements and running key sequence is determined. The resulting key sequence is compared with the real key sequence.
- **Time-Memory Trade-off Attack:** The attack has two phases. The general structure of the algorithm is explored and findings are summarized in large tables in the preprocessing phase. In real time phase, using the real data generated from a particular unknown key, the precomputed tables are used to find the unknown key.
- **Chosen-IV Attack:** The attacker chooses IVs many times and combines the relations of resulting keystream and state bits. He then derives simple relations between the state bits and secret key bits.

## 2.2 Grain Family

### 2.2.1 History of Grain Family

Lightweight cryptography algorithms should be either very fast in software or should be very small in hardware to be preferable against block ciphers in OFB or CTR mode. This idea has been reflected by the eSTREAM Project [27], which was launched in 2004 as part of ECRYPT, the European Network of Excellence in Cryptology. The aim of eSTREAM was to promote the design of new stream ciphers that would be either very fast in software or very resource-efficient in hardware.

A new stream cipher Grain[28] was developed by Hell, Johansson and Meier for eSTREAM Project and Grain v1[28] was one of the seven final ciphers of the eSTREAM portfolio for the hardware-oriented part in 2008. Since Grain v1 has 80 bit key length, the designers developed Grain128 [29] and Grain128a [30] for the 128 bit key security.

### 2.2.2 Grain Family Structure

The Grain family algorithm structure is shown in Figure 2.6. The Grain family algorithm is a synchronous stream cipher. Its design is based on two shift registers, an LFSR and an NFSR and an output function. Both NFSR and LFSR sizes are 80/128 bits. The key size is 80/128 bits and the IV size is 64/96 bits.

Grain128 design specification is given as an example of Grain family. The notation of the LFSR is shown as  $s_i, s_{i+1}, \dots, s_{i+128}$  and the NFSR is shown as  $b_i, b_{i+1}, \dots, b_{i+128}$ .

The LFSR update function is defined as:

$$s_{i+128} = s_i + s_{i+7} + s_{i+38} + s_{i+70} + s_{i+81} + s_{i+96}.$$

The NFSR update function is defined as:

$$b_{i+128} = s_i + b_i + b_{i+26} + b_{i+56} + b_{i+91} + b_{i+96} + b_{i+3}b_{i+67} + b_{i+11}b_{i+13} + b_{i+17}b_{i+18} + b_{i+27}b_{i+59} + b_{i+40}b_{i+48} + b_{i+61}b_{i+65} + b_{i+68}b_{i+84}$$

An  $h(x)$  function is defined as:

$$h(x) = b_{i+12}s_{i+8} + s_{i+13}s_{i+20} + b_{i+95}s_{i+42} + s_{i+60}s_{i+79} + b_{i+12}b_{i+95}s_{i+95}$$

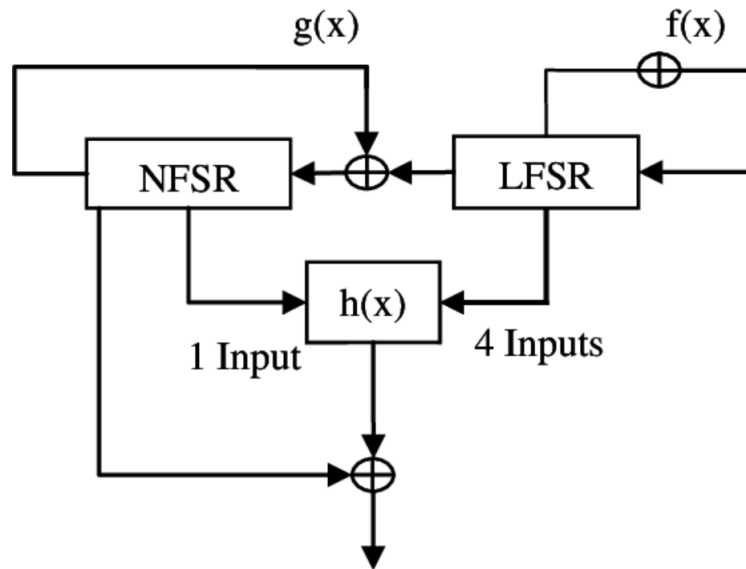


FIGURE 2.6: Grain Family Structure [31]

The output function is defined as  $z_i = \sum_{j \in A} b_{i+j} + h(x) + s_{i+93}$  where

$$A = \{2, 15, 36, 45, 64, 73, 89\}$$

### Cipher Initialization

The cipher is initialized with the key and the IV before the generation of the keystream.

Grain-128 key and IV initialization structure is shown in Figure 2.7

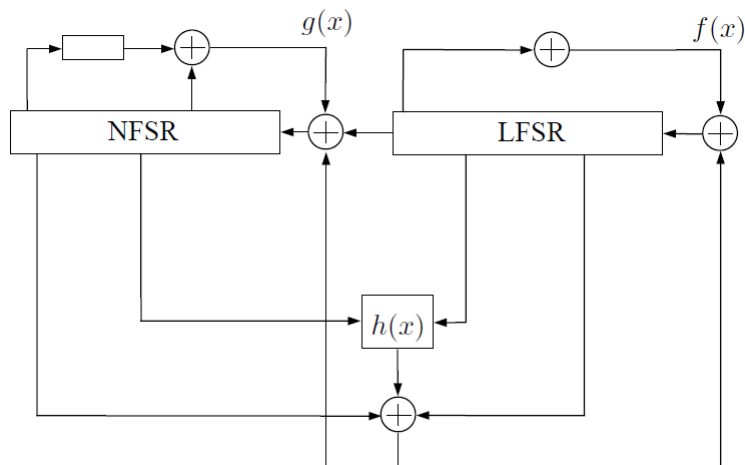


FIGURE 2.7: Grain Key and IV Initialization [29]

The steps followed in initialization are:

- All 128 bits of NFSR are filled with the key bits.

- The 96 bit IV is loaded to the first 96 bits of LFSR. The remaining 32 bits of LFSR are loaded by ones. In Grain 128a last bit is set to zero.
- Without producing any keystream, the cipher is clocked  $2K$ (number of key bits) times.
- The output function is fed back and combined (XORed) with the inputs of the LFSR and the NFSR.

### 2.2.3 Grain Family Design Criteria/Choices

In this section, reasons and usage of the NFSR, LFSR and output functions will be given against cryptanalytic attacks.

#### Usage of NFSR

An NFSR is used since using an LFSR without an NFSR would be vulnerable to algebraic attacks. Nonlinear update of the NFSR makes it impossible to solve equations for 160/256 bit state

#### Size of LFSR and NFSR

Internal state size must be at least twice of key size, so the state size is chosen as 160/256 bit (state size of LFSR + state size of NFSR). Computational complexity will be  $2^{80}$  or  $2^{128}$  for TMDTO attacks.

#### Choice of $f()$

Feedback polynomial of LFSR is primitive polynomial assuring a period at least  $2^{80} - 1$  or  $2^{128} - 1$ . Number of taps entering the polynomial should be greater than five for correlation attacks. Big number of taps is not preferable for hardware implementation. In Grain family LFSR feedback polynomials have 6 taps.

#### Choice of $g()$

Feedback function of NFSR,  $g()$  is used to achieve high nonlinearity. NFSR is masked with output of LFSR and add linear terms for balancedness against linear approximation attacks.

### **Choice of Output Function**

Output function depend on both registers.  $h(x)$  is a nonlinear and balanced function. Output function uses this non linearity and linear terms added to prevent unbalancedness for linear approximation attacks.

### **Initialization Choice**

Contents of shift registers are scrambled by the initialization phase before the keystream generation. The number of clockings for initialiation phase is a tradeoff between speed and security. The number of clocking of the initialization phase will be criticalif the cipher is initialized often with a new IV. The LFSR is filled with the IV and ones at the beginning. The initialization with two different IVs, differing by only one bit, should end with shift register bits are the same for both initializations should be close to 0.5. 160/256 clockings provides this probability.

### **Throughput Rate**

At regular clocking the output rate is 1 bit/clock in Grain128 design but it is possible to increase the speed of cipher with using more hardware since last 31 bits of shift registers are not used in update and output function. Using this, speed can be multiplied by 32. The output rate can be 32 bits/clock.If speed is multiplied by 32 , shift registers should be designed to be shifted by 32.

## Chapter 3

# Sprout

In this chapter, Sprout algorithm and the generic guess and determine attack which is the starting point of our proposed attack, mounted on Sprout is explained in detail.

### 3.1 Sprout

A new algorithm named as Sprout using the basic design of Grain family was developed at 2015 by Armknecht and Mikhalev [8]. The aim was to achieve a resistancy to TMDTO attacks for stream ciphers even using shorter internal states.

TMDTO attacks against keystream generators can be done in two ways:

- Recover key: It is like exhaustive key search but it is precomputed. Search space is  $2^k$ .
- Recover the internal state: The attacker takes an internal state and generates output stream and saves the result. Search space is  $2^s$ . To achieve a k bit security, the internal state size should be greater than k since knowing one internal state provides computing all succeeding and preceding keystreams, there is no need to search all  $2^s$  space. It is calculated as the internal state size should be greater than  $2k$ .

### 3.1.1 Keystream-equivalent states

Two states,  $st$  and  $st'$  are called as keystream-equivalent states if there exists an integer  $m$  such that after we update  $st$  by  $Upd()$  function  $m$  times, we get the same keystream for both  $st$  and  $st'$ . The structure of key stream equivalent states is shown in Figure 3.1.

Keystream-equivalent class of states:

$$[St] = \{St' \in S \mid St \equiv_{kse} St'\}$$

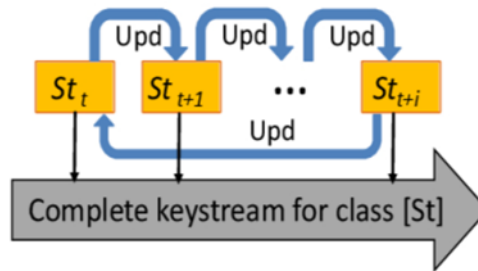


FIGURE 3.1: Key Stream Equivalent States [32]

Using key equivalent states, state space  $S$  is composed of  $L$  different equivalence classes. A TMDTO attack will be the combination of TMDTO attacks, each of them will be for each equivalence class. So aim is to design a cipher where  $L > 2^k$ .

In order to decrease the state size of the cipher, the designers of *Sprout* developed a strategy for key stream equivalent states. The design was adding a distinct fixed part to state. State will have a fixed and a variable part. Since fixed part can not be changed, two different values of fixed part result in two different equivalent classes. A KSG with fixed internal state parts is shown in Figure 3.2.

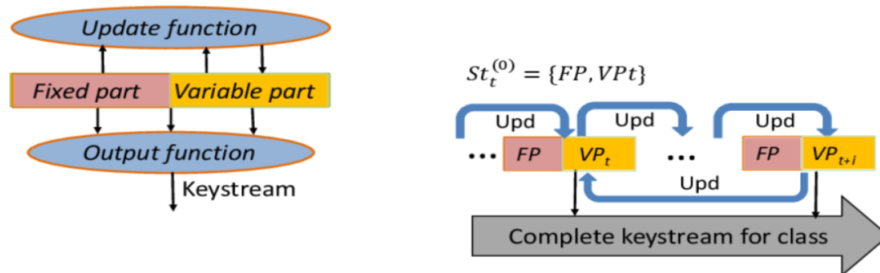


FIGURE 3.2: Keystream Generator with Fixed Internal State Parts [32]

### 3.1.2 Keystream Generator With Keyed Update Function

The designers of Sprout developed a novel keystream generator using the fixed part idea in the state of the stream cipher. They defined it as "Keystream generator with keyed update function".

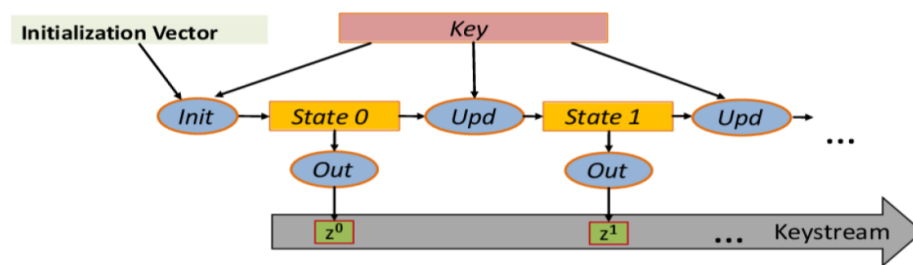


FIGURE 3.3: Keystream Generator With Keyed Update Function [32]

A keystream generator with keyed update function (KSGUF) works on three spaces

- the key space  $K = GF(2)^k$
- the IV space  $IV = GF(2)^v$
- the state space  $S = GF(2)^s$

and has three functions

- an initialization function  $\text{Init} : IV \times K \rightarrow S$
- a bijective update function  $\text{Upd} : K \times S \rightarrow S$
- an output function  $\text{Out} : S \rightarrow GF(2)$

A variable  $st$  and a fixed  $k$  composes the internal state. The fixed secret key is involved in the state update. Simplified structure of a KSG is shown in Figure 3.3.

The advantage of using a fixed key is that the cipher would have at least  $2^k$  different key-stream equivalence since  $(st, k)$  and  $(st', k')$  when  $k$  is not equal to  $k'$  will not generate the same keystream so it is possible to use shorter internal state and save area size. Using a fixed value is preferred because it uses less area than using a variable value.





- 9 bit counter splits into 2 parts. 7 bits for index of key bit. 2 bits for initialization phase to count until  $4 \times 80 = 320$

- Round Key Function is:

The content of the LFSR :  $l_i, l_{i+1}, \dots, l_{i+39}$

The content of the NFSR :  $n_i, n_{i+1}, \dots, n_{i+39}$

Key :  $k_0, k_1, \dots, k_{79}$

$k_t^*$  : the round key bit produced at the clock-cycle  $t$

$k_t^* = k_t, 0 \leq t \leq 79$

$k_t^* = (k_{t \bmod 80}) * (l_4 + l_{21} + l_{37} + n_9 + n_{20} + n_{29}), t \geq 80$

The circuit design of round key function is shown in Figure 3.5.

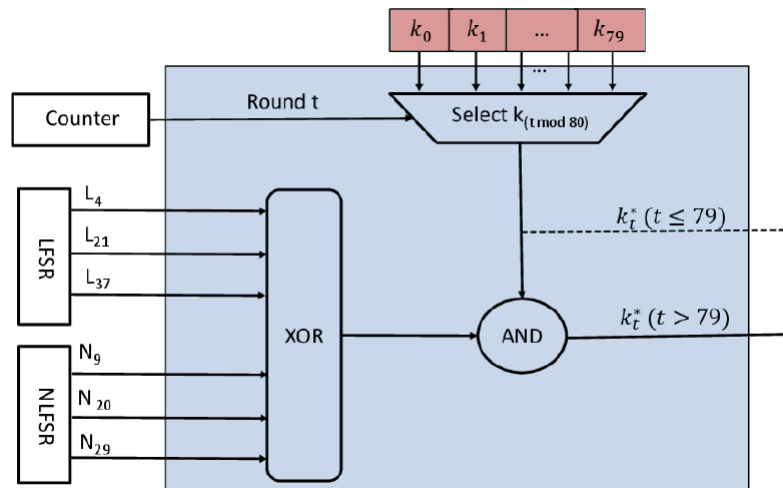


FIGURE 3.5: Round Key Function [32]

- Output function is a nonlinear function of the LFSR and the NFSR bits.
- Initialization Phase: Since the IV is 70 bits, the 40 bit NFSR is loaded with first 40 bits of IV and the 30 bit LFSR is loaded with last 30 bits of IV. Remaining 9 bit of LFSR is loaded with 1 and last bit of LFSR,  $l_{39}$  is loaded with 0. Algorithm is clocked 320 times. During this stage, the keystream is not generated. The output function is fed back into the inputs of LFSR and NFSR.

The Initialization Phase of Sprout is shown in Figure 3.6.



suggest a feedback value and check whether this guess is correct or not using the output stream.

In the attack, the attacker has a keystream and she tries to find the internal state which produced this keystream in a set of candidate states. The attacker predicts feedback values of candidate states and also checks whether the feedback value can be determined from the keystream. They used the guess capacity of the state to check whether the feedback value is the expected one or not. They defined the mismatch as the unexpected value for the feedback value. The number of mismatches for a correct state would be much less than the number of mismatches for the wrong states but for a wrong state, the number of mismatches is expected to be half of the total number of iterations. The attacker goes on recovering the next feedback value for each candidate state and also keeps the counts of mismatched feedback values for each candidate. The recovering feedback value and counting procedure continues for each state candidate until the state is eliminated because of exceeding the threshold value for mismatch count.

The mismatch count for a wrong state is approximately one half of the iteration where mismatch count for the correct state is  $(1 - \text{guesscapacity})$  times number of iterations. Hence, the attack can be applied to keystream generators with KFF having guess capacities greater than one half. If guess capacity is one-half, number of mismatches for correct and wrong states will be the same and correct states can not be distinguished. After determining the internal state, the next issue will be to recover the key. This is done by computing real feedback values from the internal state and determine the information about the key from the feedback values by solving a system of equations.

The main weakness in Sprout round key function was that key bits are not directly used in the NFSR update function which determines the feedback value of NFSR. Guess capacity is not one-half for Sprout which makes Sprout as an instantiation of KSGs with Boolean KFF having guess capacity greater than one-half and the attack could be applied to Sprout.

The designers of Sprout proposed a new algorithm Plantlet [14] having guess capacity equal to one-half where this attack can not be applied. This new algorithm type having guess capacity equal to one-half is the target of our study.

## Chapter 4

# Plantlet and Fruit

In this chapter, Plantlet algorithm and Fruit algorithm are explained in detail.

### 4.1 Plantlet

#### 4.1.1 Plantlet Design Goals

Plantlet [14] is a lightweight 80 bit stream cipher designed for low area requirements. It inherits the overall structure of Sprout but implements fixes for discovered vulnerabilities of Sprout. The fixes done in Plantlet design are:

- LFSR size is enlarged from 40 bits to 61 bits, IV size enlarged from 70 to 90 bits.
- Key selection round key function is updated. At each update a key bit is involved.
- Double-layer LFSR is introduced for high period and avoids LFSR being initialized with the all-zero case.

#### 4.1.2 Plantlet Specification

The Plantlet structure is shown in Figure 4.1. The following notation is given below which will be used to understand Plantlet specification and the generic attack we applied on it.



$$l_{59}^{t+1} = l_{54}^t + l_{43}^t + l_{34}^t + l_{20}^t + l_{14}^t + l_0^t \text{ for } 0 \leq t \leq 319$$

$$l_{60}^{t+1} = l_{54}^t + l_{43}^t + l_{34}^t + l_{20}^t + l_{14}^t + l_0^t \text{ for } t \geq 320$$

### NFSR and Counter

40 bit NFSR and 9 bit counter are adopted from Sprout. NFSR update function is XOR of non-linear combination of several NFSR bits, current key bit, output of the LFSR  $l_0^t$  and a counter bit.

$$n_{39}^{t+1} = k^t + l_0^t + c_4^t + n_0^t + n_{13}^t + n_{19}^t + n_{35}^t + n_{39}^t + n_2^t \cdot n_{25}^t + n_3^t \cdot n_5^t + n_7^t \cdot n_8^t + n_{14}^t \cdot n_{21}^t + n_{16}^t \cdot n_{18}^t + n_{22}^t \cdot n_{24}^t + n_{26}^t \cdot n_{32}^t + n_{14}^t \cdot n_{21}^t + n_{33}^t \cdot n_{36}^t \cdot n_{37}^t \cdot n_{38}^t + n_{10}^t \cdot n_{11}^t \cdot n_{12}^t + n_{27}^t \cdot n_{30}^t \cdot n_{31}^t$$

The counter is a 9 bit register. The first seven bits of the counter are used to count cyclically from 0 to 79. The two most significant bits is used with the first seven bits for 320 clock cycles during initialization phase.

### Round Key Function

80 bit key is used. The next key bit is selected cyclically for the NFSR update function.

$$k^t = k^{t \bmod 80}, t \geq 0$$

**Output Function** Original output function of Sprout is used. It has nonlinear parts from both LFSR and NLFSR and linear XOR of bits from both LFSR and NFSR.

$$z^t = n_4^t \cdot l_6^t + l_8^t \cdot l_{10}^t + l_{32}^t \cdot l_{17}^t + l_{19}^t \cdot l_{23}^t + l_{30}^t + n_4^t \cdot l_{32}^t \cdot n_{38}^t + n_1^t + n_6^t + n_{15}^t + n_{17}^t + n_{23}^t + n_{28}^t + n_{34}^t$$

## 4.1.3 Planlet Design Rationale

### Round Key Function

Imbalanced key involvement in Sprout was a major weakness by looking at longer periods where no key bit is used. In Plantlet, the key always influences the state feedback value.

### Internal State Size

Having internal size equal to key size equal to key size made Sprout vulnerable to guess-and-determine attacks. The size of LFSR is increased by 21 bits and also this allows for a higher period of the output sequence. The designers calculated that enlarging with 15 bits is enough but 6 bits are also added to increase security margin.

## Double Layer LFSR / Weakness of Initialization Phase of Sprout

In Sprout, output is fed back into LFSR and NFSR to ensure that the whole internal state depends on all of the key and IV bits. This may cause LFSR fall into an all-zero state. LFSR would remain in the all zero state during all encryption and causes keystream to have a very short period. Using this weakness, a key recovery attack was mounted on Sprout.

A countermeasure will be setting one LFSR bit to 1 after initialization but this may cause another weakness of having the same initial state of two inputs. Two different LFSR are used in different phases of the cipher. The LFSR of the initialization is extended by one additional bit which is set to 1. For lightweight hardware implementation, almost same polynomials are used differing only in the maximum degree term.

## 4.2 Fruit

### 4.2.1 Fruit Specification

Fruit [15] is a lightweight 80 bit stream cipher like Sprout and Planlet designed for low area requirements. The Fruit structure is shown in Figure 4.2. The internal state consists of 43 bit LFSR, 37 bit NFSR and two counters; one 7 bit  $C_r$ , the other 8 bit  $C_c$  counter.

The following notation is given below which will be used to understand Fruit specification and the generic attack we applied on it. Since the generic attack is an internal state recovery attack, initialization specification is ignored.

- $t$  - the clock-cycle number
- $L^t = (l_0^t, l_1^t, l_2^t, \dots, l_{42}^t)$  - the content of the LFSR at the clock-cycle  $t$
- $N^t = (n_0^t, n_1^t, n_2^t, \dots, n_{36}^t)$  - the content of the NLFSR at the clock-cycle  $t$
- $C_r^t = (c_0^t, c_1^t, c_2^t, \dots, c_6^t)$  - the content of the  $C_r$  counter at the clock-cycle  $t$
- $C_c^t = (c_7^t, c_8^t, c_9^t, \dots, c_{14}^t)$  - the content of the  $C_c$  counter at the clock-cycle  $t$
- $k = (k_0, k_1, k_2, \dots, k_{79})$  -key
- $k^t$  - the round key bit generated at the clock-cycle  $t$



- $z^t$  - the keystream bit generated at the clock-cycle  $t$

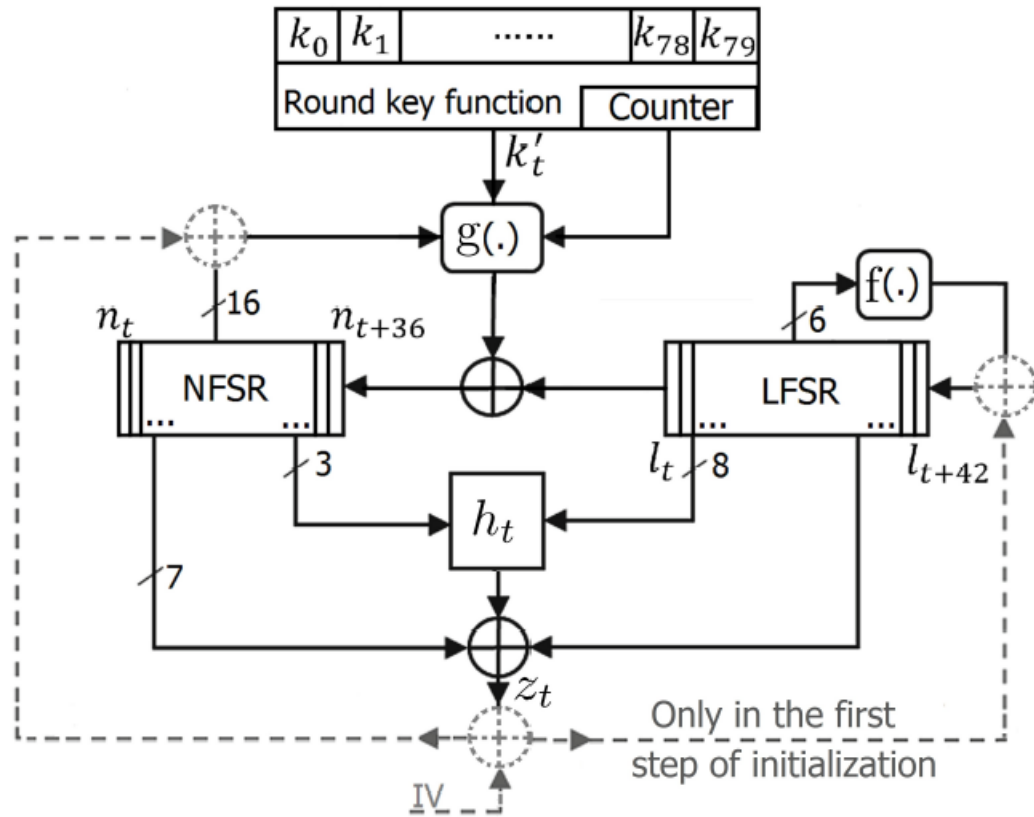


FIGURE 4.2: Fruit Structure [15]

### Counter

The seven bits of the counter,  $C_r$ , is used for round key function and the last eight bits of the counter,  $C_c$ , is used in initialization and key generation. These two counters increase at each clock independently.  $C_r$  is shown as  $c_0^t c_1^t c_2^t c_3^t c_4^t c_5^t c_6^t$  and  $c_6^t$  is the least significant bit.

### LFSR

LFSR update function is defined as:

$$l_{43}^{t+1} = l_0^t + l_8^t + l_{18}^t + l_{23}^t + l_{28}^t + l_{37}^t$$

## NFSR

NFSR update function is defined as:

$$n_{37}^{t+1} = k^t + l_0^t + c_3^t + n_0^t + n_{10}^t + n_{20}^t + n_{12}^t \cdot n_3^t + n_{14}^t \cdot n_{25}^t + n_5^t \cdot n_{23}^t \cdot n_{31}^t + n_8^t \cdot n_{18}^t + n_{28}^t \cdot n_{30}^t \cdot n_{32}^t \cdot n_{34}^t$$

## Round Key Function

80 bit key is used. Indexes of the key bits used in NFSR update function are determined with  $C_r$  counter and they change at each clock. Some variables are defined as:

$$s = c_0^t c_1^t c_2^t c_3^t c_4^t, y = c_5^t c_6^t c_0^t c_1^t c_2^t, u = c_3^t c_4^t c_5^t c_6^t, p = c_0^t c_1^t c_2^t c_3^t, q = c_4^t c_5^t c_6^t c_0^t c_1^t, r = c_2^t c_3^t c_4^t c_5^t c_6^t$$

Round key value is determined as:  $k^t = k_s \cdot k_{y+32} + k_{u+64} \cdot k_p + k_{q+16} + k_{r+48}$

## Output Function

Output function is defined as:

$$z^t = l_6^t \cdot l_{15}^t + l_1^t \cdot l_{22}^t + n_{35}^t \cdot l_{27}^t + l_{11}^t \cdot l_{33}^t + n_1^t \cdot n_{33}^t \cdot l_{42}^t + n_0^t + n_7^t + n_{13}^t + n_{19}^t + n_{24}^t + n_{29}^t + n_{36}^t + l_{38}^t$$

## 4.3 Guess Capacities of Plantlet and Fruit

Since  $k_t$  is XOR'ed in the update functions of Plantlet and Fruit, the average guess capacities of Plantlet and Fruit are one-half. Formally, the guess capacity for an internal state is formulated in [11] as:

$$Pr_g(S) = \frac{1}{2} + \left| \frac{\#\{K : f_F(K, S) = 0\}}{2^k} - \frac{1}{2} \right|,$$

For Plantlet, the NFSR update function or feedback value of NFSR is:

$$n_{39}^{t+1} = k^t + l_0^t + c_4^t + n_0^t + n_{13}^t + n_{19}^t + n_{35}^t + n_{39}^t + n_2^t \cdot n_{25}^t + n_3^t \cdot n_5^t + n_7^t \cdot n_8^t + n_{14}^t \cdot n_{21}^t + n_{16}^t \cdot n_{18}^t + n_{22}^t \cdot n_{24}^t + n_{26}^t \cdot n_{32}^t + n_{14}^t \cdot n_{21}^t + n_{33}^t \cdot n_{36}^t \cdot n_{37}^t \cdot n_{38}^t + n_{10}^t \cdot n_{11}^t \cdot n_{12}^t + n_{27}^t \cdot n_{30}^t \cdot n_{31}^t$$

For Fruit, the NFSR update function or feedback value of NFSR is:

$$n_{37}^{t+1} = k^t + l_0^t + c_3^t + n_0^t + n_{10}^t + n_{20}^t + n_{12}^t \cdot n_3^t + n_{14}^t \cdot n_{25}^t + n_5^t \cdot n_{23}^t \cdot n_{31}^t + n_8^t \cdot n_{18}^t + n_{28}^t \cdot n_{30}^t \cdot n_{32}^t \cdot n_{34}^t$$

In the guess capacity formula, the function in the absolute value calculates the number of keys where feedback value of the NFSR equals to zero. Since  $k$  bit is directly XOR'ed

in the NFSR update functions of both Plantlet and Fruit and the other bits are same for all possible keys for an internal state, half of the possible keys make the feedback value zero, half of the possible key values make the feedback value one. The formula becomes:

$$Pr_g(S) = \frac{1}{2} + \left| \frac{2^{k-1}}{2^k} - \frac{1}{2} \right| = \frac{1}{2}$$

For all internal states of Plantlet and Fruit, the guess capacity is one-half, so the average guess capacity is one-half for Plantlet and Fruit.



## Chapter 5

# New and Parametric Guess and Determine Attack

In this chapter, new guess and determine attack and parametric guess and determine attack mounted on Plantlet and Fruit algorithms and improving new guess and determine attack through trade-off mounted on Plantlet algorithm are explained in detail.

### 5.0.1 New Guess and Determine Attack Mounted On Plantlet

The Plantlet round key function is an improved version of Sprout. It cyclically selects the next key bit for the NFSR update function. It is impossible to use guess capacity as a distinguisher for Kara and Esgin attack [11] but this design, involving key bit cyclically in the NFSR update function created a new weakness. The same key bit is used at clocks  $t$  and  $t + keysize$  for an internal state and this is used for both internal state recovery and key bit recovery at the same time.

We improved and changed Kara and Esgin algorithm [11] for our attack. The internal state candidates are not eliminated for their mismatch counts. Since all of them would have approximately same mismatch counts because of having same guess capacity, one-half. Number of iterations is a little above key size, 90 for Plantlet. The feedback value is either determined from the output stream or guessed. Since the key bit is directly XOR'ed in the NFSR update function, we calculated the key value at each clock from feedback value and internal state values and saved the key value at each clock. After

80 clocks, we begin also comparing key value of the internal state candidate at clock  $t$  and the key value we saved at  $(t - 80)$ . For the correct state, they would be same at the corresponding clocks but for an incorrect internal state candidate, they would be randomly equal and after a few clocks, the state will be eliminated since key values will not match. The correct state will exist until the end of clocks (iteration) and at the same time key bits are also determined.

Determination of feedback value from output stream is given below:

$$z^t = n_4^t \cdot l_6^t + l_8^t \cdot l_{10}^t + l_{32}^t \cdot l_{17}^t + l_{19}^t \cdot l_{23}^t + l_{30}^t + n_4^t \cdot l_{32}^t \cdot \mathbf{n}_{38}^t + n_1^t + n_6^t + n_{15}^t + n_{17}^t + n_{23}^t + n_{28}^t + n_{34}^t$$

$$z^{t+1} = n_5^t \cdot l_7^t + l_9^t \cdot l_{11}^t + l_{33}^t \cdot l_{18}^t + l_{20}^t \cdot l_{24}^t + l_{31}^t + n_5^t \cdot l_{33}^t \cdot \mathbf{n}_{39}^t + n_2^t + n_7^t + n_{16}^t + n_{18}^t + n_{24}^t + n_{29}^t + n_{35}^t$$

$$z^{t+2} = n_6^t \cdot l_8^t + l_{10}^t \cdot l_{12}^t + l_{34}^t \cdot l_{19}^t + l_{21}^t \cdot l_{25}^t + l_{32}^t + n_6^t \cdot l_{34}^t \cdot \mathbf{n}_{39}^{t+1} + n_3^t + n_8^t + n_{17}^t + n_{19}^t + n_{25}^t + n_{30}^t + n_{36}^t$$

$$n_6^t \cdot l_{34}^t \cdot \mathbf{n}_{39}^{t+1} = z^{t+2} + n_6^t \cdot l_8^t + l_{10}^t \cdot l_{12}^t + l_{34}^t \cdot l_{19}^t + l_{21}^t \cdot l_{25}^t + l_{32}^t + n_3^t + n_8^t + n_{17}^t + n_{19}^t + n_{25}^t + n_{30}^t + n_{36}^t$$

This implies for a internal state candidate at clock  $t$  when  $n_6^t$  and  $l_{34}^t$  are equal to 1, feedback value of NFSR can be calculated from  $z^{t+2}$ , LFSR and NFSR tap values.

Determination of key value at clock  $t$  is given below:

$$n_{39}^{t+1} = k^t + l_0^t + c_4^t + n_0^t + n_{13}^t + n_{19}^t + n_{35}^t + n_{39}^t + n_2^t \cdot n_{25}^t + n_3^t \cdot n_5^t + n_7^t \cdot n_8^t + n_{14}^t \cdot n_{21}^t + n_{16}^t \cdot n_{18}^t + n_{22}^t \cdot n_{24}^t + n_{26}^t \cdot n_{32}^t + n_{14}^t \cdot n_{21}^t + n_{33}^t \cdot n_{36}^t \cdot n_{37}^t \cdot n_{38}^t + n_{10}^t \cdot n_{11}^t \cdot n_{12}^t + n_{27}^t \cdot n_{30}^t \cdot n_{31}^t$$

We just replaced key and feedback values and get the key value at clock  $t$ .

$$k^t = n_{39}^{t+1} + l_0^t + c_4^t + n_0^t + n_{13}^t + n_{19}^t + n_{35}^t + n_{39}^t + n_2^t \cdot n_{25}^t + n_3^t \cdot n_5^t + n_7^t \cdot n_8^t + n_{14}^t \cdot n_{21}^t + n_{16}^t \cdot n_{18}^t + n_{22}^t \cdot n_{24}^t + n_{26}^t \cdot n_{32}^t + n_{14}^t \cdot n_{21}^t + n_{33}^t \cdot n_{36}^t \cdot n_{37}^t \cdot n_{38}^t + n_{10}^t \cdot n_{11}^t \cdot n_{12}^t + n_{27}^t \cdot n_{30}^t \cdot n_{31}^t$$

The generic attack algorithm named as "New Guess and Determine Attack" steps can be summarized as follows:

- **Input:** Output stream bits, internal state candidates,
- **Output:** The key bits and the correct internal state which generated the given output stream bits,
- **Attack Phase**
  - For each internal state candidate execute the following steps:

1. Determine or guess the feedback value of the NFSR from output stream bit, NFSR and LFSR bits at each clock.
  2. Determine the key value from the feedback value of NFSR, the other NFSR and LFSR bits.
  3. Eliminate internal state if guessed feedback value is incorrect in the next clock.
  4. Check key bits which should be equal after key bits start repeating. (e.g.  $k_t == k_{t-80}$  for Plantlet)
  5. Eliminate internal state candidate if key bits are not equal.
- The correct internal state which generated the output stream will be left at the end of iterations. Key bits are automatically determined during iterations for the correct internal state.

This algorithm can be used for any KSGs with Boolean KFF where key size and internal state size are equal but it is not effective for Plantlet since Plantlet internal state size is 101 bits. (61 bits from LFSR and 40 bits from NFSR) For this attack, the attacker should use  $2^{101}$  internal state candidates to recover the correct internal state and key bits which is much higher than  $2^{80}$  key candidates of exhaustive key search. In order to decrease the complexity of the attack, two novel ideas/extensions are implemented and combined with the "New Guess and Determine Attack":

- Parametric Guess and Determine Attack
- Improving New Guess and Determine Attack Through Trade-Off

### 5.0.2 Parametric Guess and Determine Attack Mounted On Plantlet

Before applying the first idea on the attack, two improvements are done to simplify the attack for Plantlet. In the first step, cipher was run in backward direction and the keystream generation and feedback functions are formulated in backward direction as in [9]. This improvement is done since in the output function of the cipher,  $n_1^t$  alone is XORed with other taps. In backward direction,  $n_0^{t+1}$  will be the feedback value and  $n_1^t$  will be the feedback value after two clocks which can be determined from output stream value  $z^{t+2}$  and the output function. In the previous attack, the feedback value

was either determined or guessed, but in our new attack, the feedback values can be always determined from the output stream. The derivations are shown below:

LFSR update in forward direction:

$$l_{60}^{t+1} = l_0^t + l_{54}^t + l_{43}^t + l_{34}^t + l_{20}^t + l_{14}^t,$$

$$l_i^{t+1} = l_{i+1}^t$$

LFSR update in backward direction:

$$l_0^{t+1} = l_{60}^t + l_{53}^t + l_{42}^t + l_{33}^t + l_{19}^t + l_{13}^t,$$

$$l_i^{t+1} = l_{i-1}^t$$

NFSR update in forward direction:

$$n_{39}^{t+1} = k^t + l_0^t + c_4^t + n_0^t + n_{13}^t + n_{19}^t + n_{35}^t + n_{39}^t + n_2^t \cdot n_{25}^t + n_3^t \cdot n_5^t + n_7^t \cdot n_8^t + n_{14}^t \cdot n_{21}^t + n_{16}^t \cdot n_{18}^t + n_{22}^t \cdot n_{24}^t + n_{26}^t \cdot n_{32}^t + n_{14}^t \cdot n_{21}^t + n_{33}^t \cdot n_{36}^t \cdot n_{37}^t \cdot n_{38}^t + n_{10}^t \cdot n_{11}^t \cdot n_{12}^t + n_{27}^t \cdot n_{30}^t \cdot n_{31}^t,$$

$$n_i^{t+1} = n_{i+1}^t$$

NFSR update in backward direction:

$$n_0^{t+1} = k^t + c_4^t + (l_{60}^t + l_{53}^t + l_{42}^t + l_{33}^t + l_{19}^t + l_{13}^t) + n_{39}^t + n_{12}^t + n_{18}^t + n_{34}^t + n_{38}^t + n_1^t \cdot n_{24}^t + n_2^t \cdot n_4^t + n_6^t \cdot n_7^t + n_{13}^t \cdot n_{20}^t + n_{15}^t \cdot n_{17}^t + n_{21}^t \cdot n_{23}^t + n_{25}^t \cdot n_{31}^t + n_{13}^t \cdot n_{20}^t + n_{32}^t \cdot n_{35}^t \cdot n_{36}^t \cdot n_{37}^t + n_9^t \cdot n_{10}^t \cdot n_{11}^t + n_{26}^t \cdot n_{29}^t \cdot n_{30}^t,$$

$$n_i^{t+1} = n_{i-1}^t$$

Output function does not change:

$$z^t = n_4^t \cdot l_6^t + l_8^t \cdot l_{10}^t + l_{32}^t \cdot l_{17}^t + l_{19}^t \cdot l_{23}^t + l_{30}^t + n_4^t \cdot l_{32}^t \cdot n_{38}^t + n_1^t + n_6^t + n_{15}^t + n_{17}^t + n_{23}^t + n_{28}^t + n_{34}^t$$

In the second step, we simulated Plantlet running in backward direction as if it runs in forward direction.

LFSR update function:

$$l_{60}^{t+1} = l_0^t + l_7^t + l_{18}^t + l_{27}^t + l_{41}^t + l_{47}^t,$$

$$l_i^{t+1} = l_{i+1}^t$$

NFSR update function:

$$n_{39}^{t+1} = k^t + c_4^t + (l_0^t + l_7^t + l_{18}^t + l_{27}^t + l_{41}^t + l_{47}^t) + n_0^t + n_{27}^t + n_{21}^t + n_5^t + n_1^t + n_{38}^t \cdot n_{15}^t + n_{37}^t \cdot n_{35}^t + n_{33}^t \cdot n_{32}^t + n_{26}^t \cdot n_{19}^t + n_{24}^t \cdot n_{22}^t + n_{22}^t \cdot n_{24}^t + n_{18}^t \cdot n_{16}^t + n_{14}^t \cdot n_8^t + n_7^t \cdot n_4^t \cdot n_3^t \cdot n_2^t + n_{30}^t \cdot n_{29}^t \cdot n_{28}^t + n_{13}^t \cdot n_{10}^t \cdot n_9^t,$$

$$n_i^{t+1} = n_{i+1}^t$$

Output function:

$$z^t = n_{35}^t \cdot l_{54}^t + l_{52}^t \cdot l_{50}^t + l_{28}^t \cdot l_{43}^t + l_{41}^t \cdot l_{37}^t + l_{30}^t + n_{35}^t \cdot l_{28}^t \cdot n_1^t + \mathbf{n}_{38}^t + n_{33}^t + n_{24}^t + n_{22}^t + n_{16}^t + n_{11}^t + n_5^t$$

Derivation of the determination of the feedback value from output function is given below:

$$z^t = n_{35}^t \cdot l_{54}^t + l_{52}^t \cdot l_{50}^t + l_{28}^t \cdot l_{43}^t + l_{41}^t \cdot l_{37}^t + l_{30}^t + n_{35}^t \cdot l_{28}^t \cdot n_1^t + \mathbf{n}_{38}^t + n_{33}^t + n_{24}^t + n_{22}^t + n_{16}^t + n_{11}^t + n_5^t$$

$$z^{t+1} = n_{36}^t \cdot l_{55}^t + l_{53}^t \cdot l_{51}^t + l_{29}^t \cdot l_{44}^t + l_{42}^t \cdot l_{38}^t + l_{31}^t + n_{36}^t \cdot l_{29}^t \cdot n_2^t + \mathbf{n}_{39}^t + n_{34}^t + n_{25}^t + n_{23}^t + n_{17}^t + n_{12}^t + n_6^t$$

$$z^{t+2} = n_{37}^t \cdot l_{56}^t + l_{54}^t \cdot l_{52}^t + l_{30}^t \cdot l_{45}^t + l_{43}^t \cdot l_{39}^t + l_{32}^t + n_{37}^t \cdot l_{30}^t \cdot n_3^t + \mathbf{n}_{39}^{t+1} + n_{35}^t + n_{26}^t + n_{24}^t + n_{18}^t + n_{13}^t + n_7^t$$

$$\mathbf{n}_{39}^{t+1} = z^{t+2} + n_{37}^t \cdot l_{56}^t + l_{54}^t \cdot l_{52}^t + l_{30}^t \cdot l_{45}^t + l_{43}^t \cdot l_{39}^t + l_{32}^t + n_{37}^t \cdot l_{30}^t \cdot n_3^t + n_{35}^t + n_{26}^t + n_{24}^t + n_{18}^t + n_{13}^t + n_7^t$$

As it can be seen above, feedback value of the NFSR can always be determined from the output stream. This simplified our attack, since we didn't have to guess and check the feedback value of NFSR. Using output stream values, the feedback values of NFSR are directly determined at each clock.

The first new idea in order to decrease the complexity of the attack was replacing some taps/elements of the LFSR with variables during the attack. We randomly defined some taps of the LFSR as variables and implemented the attack algorithm using the internal state candidates with LFSR having variables in some taps instead of values. The key bits determined would be nonlinear equations of the variables. A new stage is added to attack algorithm as solving equations for the variables and the correct internal state would have the unique solution for the variables. After determining variable values, key bits are also determined using them.

The generic attack algorithm named as "Parametric Guess and Determine Attack" steps can be summarized as follows:



- **Input:** Output stream bits, internal state candidates,
- **Output:** The key bits and the correct internal state which generated the given output stream bits,
- **Initialization Phase:**
  - Check if the NFSR feedback value can be determined from output stream in forward or backward direction. If the NFSR feedback value can be determined from output stream, formulate feedback functions in backward direction.
  - Replace a fixed number of random bits of LFSR with variables.
- **Attack Phase**
  - For each internal state candidate execute the following steps:
    1. Determine the feedback value of the NFSR from output stream bit, NFSR and LFSR bits as a nonlinear function of variables of LFSR at each clock.
    2. Determine the key value from the feedback value of NFSR, the other NFSR and LFSR bits as a nonlinear function of variables of LFSR at each clock.
    3. Generate equations for the keys which should be equal after key bits start repeating. (e.g.  $k_t == k_{t+80}$  for Plantlet)
    4. Solve the nonlinear equations generated in the previous step.
    5. Eliminate internal state candidate if there is no solution.
  - The correct internal state which generated the output stream will be left with a unique solution for the variables.
  - Calculate the key bits using the unique solution of the variables.

As a proof of concept, we implemented algorithm using six variables and recovered the correct internal state bits and key bits successfully. Six random taps of the LFSR are chosen as variables  $x, y, z, w, k, u$ . The LFSR update function is simple XOR of LFSR bits but the NFSR update function has both XOR and multiplication of NFSR bits. Because of the multiplication effect, the bits and the feedback values of the NFSR and the key values determined from the feedback values are calculated as nonlinear equations of  $x, y, z, w, u, k$  variables consisting both XOR and multiplication operations. Since there are six variables, nonlinear equations have  $2^6 = 64$  combination of  $x, y, z, w, k,$

u. The number of variables is chosen as six to write the code for multiplication of two bits of the NFSR having both 64 combinations of x, y, z, w, u, k easily, e.g. if we chose the number of the variables as seven, multiplication code should be written for multiplying two taps both having 128 different combinations of the variables. At the end of the implementation of the algorithm, we have nonlinear equations of the variables representing key values  $k_t, t = 0, 1, 2, \dots, 90$  where  $k_t = k_{t-80}$  which should be equal to each other for the correct internal state candidate.

In the second stage, an exhaustive search is done to recover the variables; x, y, z, w, k, u. There are  $2^6 = 64$  different candidates for them. The equations are solved for a unique solution using 6.2 equations on average. A candidate is either eliminated or a unique solution is left. For the correct internal state candidate, the unique x, y, z, w, k, u solution satisfies all equations and key is also recovered by solving the equations for  $k_0$  to  $k_{80}$ .

In the previous attack, the attacker should use  $2^{101}$  internal state candidates for Plantlet but this improvement decreased internal state candidate number to  $2^{95}$ . Same attack algorithm is used but comparison of the key values differed. Key values are nonlinear equations of variables where  $k_t$  should be equal to  $k_{t-80}$ . At the end of iterations, the correct internal state and the key values are recovered since wrong internal candidates are eliminated during solving equations for variables or checking whether the next equation is satisfied for the left unique solution of variables. The time for multiplying the NFSR bits as nonlinear equations and solving the nonlinear equations should be added to attack complexity.

The elapsed times for both attacks are compared for 1000 internal state candidates. Both new guess and determine attack and parametric guess and determine attack with 6 variables are executed on a standard PC. The time elapsed during new guess and determine attack is 0.12 seconds and the time elapsed during parametric guess and determine attack is 6.6 seconds. The new guess and determine attack is about 55 times faster than the parametric guess and determine attack, but this is normal since each bit is represented by 64 bits and multiplication of two bits transformed into the multiplication of two 64 bits. This test coding is done as a proof of concept not for performance but it is still faster than new guess and determine attack using  $2^{101}$  candidates.

For Plantlet, the attacker should use the attack algorithm with 22 variables to be faster than exhaustive key search but implementation of using 22 variables would not be similar to using 6 variables. We have two problems to solve:

1. Generating nonlinear equations for  $2^{22}$  combinations of variables and multiplication of two nonlinear equations having  $2^{22}$  combinations of variables.
2. Solving nonlinear equations having  $2^{22}$  combinations of 22 variables.

These two items can not be done online during attack since it would decrease the attack speed exponentially. For the first item, the multiplication of two nonlinear equations having  $2^{22}$  combinations of variables can be implemented as multiplication of two polynomials and can be precomputed offline and saved in a table. During the attack, when two NFSR bits are multiplied, the result will be read directly from this table. The memory access should be done for each multiplication of NFSR bits. For the second item, solving nonlinear equations having  $2^{22}$  combinations of 22 variables can also be precomputed offline saved in group of tables as  $(2^{22}, 23)$  solutions. Since the nonlinear equations will be solved on average 23 equations but this will need a huge memory. When the nonlinear equations for key equalities are created at the end of the attack algorithm, the solutions should be searched from these tables.

The memory needed for the attack for  $p$  variables can be formulated as  $2^{2p}$  for multiplication table and  $\binom{2^p}{p+1}$  for solving the nonlinear equations. The attack complexity would be  $2^{n-p}$  if the generation and solving the nonlinear equations would be done by table lookups and the memory needed for the attack would be  $2^{2p} + \binom{2^p}{p+1}$ . Multiplication table would be accessed for each multiplication of two NFSR bits.

### 5.0.3 Improving New Guess and Determine Attack Through Trade-Off

The second idea to decrease the complexity of the attack was to use our new guess and determine attack with trade-off. We made an extension for the new guess and determine attack for Plantlet like ciphers where the key bit is directly and cyclically XOR'ed in the internal state update function and key cycle period is equal to key bit size and we combined our attack with TMDTO attacks.

The extension attack steps applied for Plantlet are :

- The attacker has a generated 80 bit output stream,  $z_1$ .
- Make a guess for the internal state which generated this output stream,  $z_1$ .
- Using this internal state candidate and output stream, run our new guess and determine attack and determine 80-bit key bits.
- Using the determined key bits and internal state candidate, generate the 101 bit output stream,  $z_2$ , following the  $z_1$  sequence.
- For a constant  $z_1$ , for each guess of the internal state, we determine a unique 101 bit  $z_2$ .
- For a constant  $z_1$ , this can be called as a one way function between internal state candidate and output stream  $z_2$ . If we have internal state candidate, we can determine the following output stream  $z_2$  for a constant  $z_1$ , but if we have  $z_2$ , we can not determine internal state value for a constant  $z_1$ .
- Using Hellman [33] tables, the formula  $T.M^2 = N^2$  and our new guess and determine attack, for a constant  $z_1$ , to recover the internal state and key bits, instead of making exhaustive search for internal state candidates as  $2^{101}$  trials, with  $T = 2^{80}$  complexity, we need a memory M as  $2^{61}$  to determine the internal state value which generated this output stream. Data complexity would be  $2^{80}$  for this case.
- Data complexity can be reduced by increasing memory in Hellman tables. In the previous item, we have a constant  $z_1$  and we prepare Hellman tables for  $z_1$  and search  $z_1$  in the given data. If we use another 80 bit output stream as input, we can decrease data complexity. Since we would prepare Hellman tables for each of output stream sequence as  $2 \cdot 2^{61}$ . Memory complexity, M, would be  $2^{62}$  where data complexity, D, would decrease to  $2^{79}$  to search for the target output stream. There is a trade-off between memory M and data D where  $M \cdot D$  is fixed.

Using the trade-off attack with our new guess and determine attack, we developed a theorem for Plantlet like ciphers as:

**Theorem:** For Plantlet like ciphers where the key bit is directly and cyclically XOR'ed in the internal state update function and key cycle period is equal to the key size, internal

state recovery can be done using new guess and determine attack with trade-off attack with a formula as:

$$T(DM)^2 = (NK)^2$$

where T: Time Complexity, D: Data Complexity, N: Internal State Size Complexity, K : Key Complexity

**Sketch:** Assume attacker has k bit output stream. Using this k bit output stream, for n bit internal state, he generates the following n bit output stream.

This can be defined as a one-way function between n bit internal state to n bit output stream.

Using Hellman tables for this function, the time and memory complexity would be:  $T.M^2 = N^2$  and for this ciphers  $D = K$  since length of the input data to be searched should be equal to size of the key to determine key bits.

If we multiply both side of the equation with M :  $D.M = K.N/\sqrt{T}$  which implies when T is fixed D.M is fixed.

We generalize the equation as:

$$T(DM)^2 = (NK)^2$$

**Corollary1:**

For Planlet like ciphers, for the cipher to be resistant to TMDTO attacks, the internal state size complexity should be as  $N \geq K^{3/2}$  so for Plantlet, the internal state size be 120 bits in order to resist TMDTO attacks.

**Proof :** When  $T = D = M = K$ , using the above formula  $K^5 = K^2.N^2 \rightarrow N = K^{3/2}$   
For Plantlet, internal state size complexity should be  $2^{(80*3/2)} = 2^{120}$  bits in order to resist TMDTO attacks.

**Corollary2:**

The attack complexity reduced to  $2^{72.4}$  for Plantlet by using the trade-off attack with our new guess and determine attack.

**Proof** : For Plantlet  $N = 2^{101}$  and  $K = 2^{80}$ , using the above formula:  $T(DM)^2 = 2^{362}$  for the best trade-off when  $T = D = M$ ,  $T = D = M = 2^{72.4}$

#### 5.0.4 New and Parametric Guess and Determine Attacks Mounted On Fruit

Since  $k_t$  is directly involved and XOR'ed with the other elements in the Fruit NFSR update function, guess capacity is one-half and our generic attack can be applied to Fruit algorithm. The same key bit is used at clocks  $t$  and  $t + 128$  for an internal state. The key period is different from Plantlet because of the round key function of Fruit. The key bits start repeating at  $C_r$  counter roll over as  $2^7 = 128$ .

Since  $n_{36}^t$  is directly XOR'ed in the output function alone, feedback value of NFSR can be determined directly from output stream. Determination of feedback value from output stream is given below:

$$z^t = l_6^t \cdot l_{15}^t + l_1^t \cdot l_{22}^t + n_{35}^t \cdot l_{27}^t + l_{11}^t \cdot l_{33}^t + n_1^t \cdot n_{33}^t \cdot l_{42}^t + n_0^t + n_7^t + n_{13}^t + n_{19}^t + n_{24}^t + n_{29}^t + n_{36}^t + l_{38}^t$$

$$z^{t+1} = l_7^t \cdot l_{16}^t + l_2^t \cdot l_{23}^t + n_{36}^t \cdot l_{28}^t + l_{12}^t \cdot l_{34}^t + n_2^t \cdot n_{34}^t \cdot l_{43}^t + n_1^t + n_8^t + n_{14}^t + n_{20}^t + n_{25}^t + n_{30}^t + n_{37}^t + l_{39}^t$$

$$z^{t+2} = l_8^t \cdot l_{17}^t + l_3^t \cdot l_{24}^t + n_{37}^t \cdot l_{29}^t + l_{13}^t \cdot l_{35}^t + n_3^t \cdot n_{35}^t \cdot l_{43}^{t+1} + n_2^t + n_9^t + n_{15}^t + n_{21}^t + n_{26}^t + n_{31}^t + n_{37}^{t+1} + l_{40}^t$$

$$n_{37}^{t+1} = z^{t+2} + l_8^t \cdot l_{17}^t + l_3^t \cdot l_{24}^t + n_{37}^t \cdot l_{29}^t + l_{13}^t \cdot l_{35}^t + n_3^t \cdot n_{35}^t \cdot (l_0^t + l_8^t + l_{18}^t + l_{23}^t + l_{28}^t + l_{37}^t) + n_2^t + n_9^t + n_{15}^t + n_{21}^t + n_{26}^t + n_{31}^t + l_{40}^t$$

As it can be seen above, feedback value of the NFSR can always be determined from the output stream,  $z^{t+2}$ , LFSR and NFSR tap values.

Determination of key value at clock  $t$  is given below:

$$n_{37}^{t+1} = k^t + l_0^t + c_3^t + n_0^t + n_{10}^t + n_{20}^t + n_{12}^t \cdot n_3^t + n_{14}^t \cdot n_{25}^t + n_5^t \cdot n_{23}^t \cdot n_{31}^t + n_8^t \cdot n_{18}^t + n_{28}^t \cdot n_{30}^t \cdot n_{32}^t \cdot n_{34}^t$$

Just replace key and feedback value and get the key value at clock  $t$ .

$$k^t = n_{37}^{t+1} + l_0^t + c_3^t + n_0^t + n_{10}^t + n_{20}^t + n_{12}^t \cdot n_3^t + n_{14}^t \cdot n_{25}^t + n_5^t \cdot n_{23}^t \cdot n_{31}^t + n_8^t \cdot n_{18}^t + n_{28}^t \cdot n_{30}^t \cdot n_{32}^t \cdot n_{34}^t$$

Since Fruit algorithm key size and internal state size are equal, to test our attack, in the first step, we didn't use any variables and applied the new guess and determine attack whose details are given in the previous chapter directly to the cipher. In the second step, we mounted the parametric guess and determine attack. The counter  $C_r$  value is set to

zero at the beginning of the attacks. For each internal state candidate, we determined key bits for 128 clocks and after clock 128, key bits are determined but we also started comparing key values. If key values are not equal ( $k_t = k_{t-128}$ ), internal state candidate is eliminated and correct internal state candidate having key values equal is left after a few clock cycles and key value included in the NFSR update function is also determined at the same time with internal state bits.

Round key function of Fruit enables nonlinear combination of key bits at each round. Since resulting key bit involved in NFSR update function is determined, in order to determine real key bits, solving nonlinear equations will be needed at the end of the attack.

In Fruit,  $C_r$  counter starting value is determined at the end of initialization phase of the cipher and it is based on the key, NFSR and LFSR bits, so an attacker would not know the  $C_r$  counter starting value to mount the attack. This issue can be easily solved since counter bit  $c_3^t$  is directly involved in NFSR update function. Even the  $C_r$  counter starting value is not correct, during comparison of key values at  $t$  and  $t-128$ ,  $c_3^t$  and  $c_3^{t-128}$  values are equal which will not affect equality. Correct feedback values are determined from output stream and our generic attack recovers the correct internal state using the key comparisons but key bits recovered depend on the  $C_r$  counter starting value. At the end of our attack, 128 bit values of  $(k^t + c_3^t)$  are recovered.

To recover the correct key bits of the Fruit algorithm:

- Mount improved or parametric guess and determine attack.
- Recover the correct internal state and key bits involved in the NFSR update function depending on  $C_r$  counter value.  $(k^t + c_3^t)$
- Solve nonlinear equations for 128 key bits and get candidate key bits for each counter value.
- Apply exhaustive search for 128 candidate key bits to recover the real key bits.

The attack complexity without using any variables is  $2^{80}$  and this is faster than exhaustive search since initialization phase is not included in the attack which should be included in the exhaustive search. The attack complexity can be reduced to  $2^{74}$  when we have

applied parametric guess and determine attack using six variables since instead of using  $2^{80}$  internal state candidates for the attack, we used  $2^{80-6}$  internal state candidates. Solving equations for six variables are done during attack as a proof of concept for our attack implementation but multiplication of the NFSR bits and solutions of the nonlinear equations for six variables must be done offline and saved in tables as explained in Plantlet section to achieve  $2^{74}$  complexity.





## Chapter 6

# Conclusion

In this thesis, the security analysis of stream ciphers having keystream generator with keyed update function is studied and a generic internal state and key recovery attack is introduced for keystream generators with keyed update function having guess capacity one-half for the first time. This generic attack is mounted on Plantlet and Fruit algorithms and internal state and key bits are recovered successfully for both of the ciphers. Involving key bits directly and cyclically in the update function of a keystream generator leads to a weakness for recovering key bits where key bits can be determined from the feedback values at the corresponding clocks. Since it is an internal state recovery attack, the initialization phase is not considered and implemented during attack which should be taken into consideration for the exhaustive key search attack. When the internal state size is greater than key size like in Plantlet, the exhaustive key search is faster than this attack so the complexity of the attack is reduced by two novel ideas and two new extensions are added on the new generic attack.

The first idea to decrease the complexity of the attack is using variables and solving nonlinear equations during key recovery. As a proof of concept, the number of variables is chosen as six and the attack is successfully mounted on Plantlet and Fruit algorithms and internal state bits and key bits are recovered. Increasing number of variables will reduce attack complexity, but it would not be easy to generate and multiply the nonlinear combinations of the variables and to find the solution of the generated nonlinear equations. Precomputation and memory would be needed for both generation and

multiplication of nonlinear combinations of these variables and for finding the solution of the generated nonlinear equations.

The second idea to decrease the complexity of the attack is combining the attack with trade-off attack. Using Hellman tables with the new generic guess and determine attack, instead of searching all  $2^{\text{internalstatesize}}$ , the attack complexity can be reduced to  $2^{80}$  time complexity, T,  $2^{61}$  memory complexity, M, and  $2^{80}$  data complexity, D where the data complexity can be decreased by increasing the memory complexity since M.D is fixed. This would be the trade-off for the attacker.

Using this trade-off, for Plantlet like ciphers where the key bit is directly and cyclically XOR'ed in the internal state update function and key cycle period is equal to the key size, a generalization is made as  $T(DM)^2 = (NK)^2$ . By using this formula, two corollaries are achieved. The first one is that for Plantlet like ciphers, the internal state size must be at least  $\text{keysize} * (3/2)$  to be resistant to TMDTO attacks. The second one is that using our new guess and determine attack with trade-off attack, the attack complexity for Planlet is reduced to  $2^{72.4}$ .

# Bibliography

- [1] S. Banik, A. Bogdanov, T. Isobe, K. Shibutani, H. Hiwatari, T. Akishita, and F. Regazzoni. Midori: A block cipher for low energy. *ASIACRYPT. Springer*, pages 411–436, 2014.
- [2] C. D. Canni'ere, O. Dunkelman, and M. Knezevic. Katan and ktantan a family of small and efficient hardware-oriented block ciphers. *CHES 2009, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg*, 5747:272–288, 2009.
- [3] A. Bogdanova, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. Present: An ultra-lightweight block cipher. *CHES 2007, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg*, 4727:450–466, 2007.
- [4] J. Guo, T. Peyrin, A. Poschmann, and M. J. B. Robshaw. The led block cipher. *CHES 2011, ser. Lecture Notes in Computer Science*, 6917:326–341, 2011.
- [5] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers. The simon and speck families of lightweight block ciphers. *IACR Cryptology ePrint Archive*, page 404, 2013.
- [6] G. Yang, B. Zhu, V. Suder, M. D. Aagaard, and G. Gong. The simeck family of lightweight block ciphers. *CHES 2015, ser. Lecture Notes in Computer Science*, pages 307–329, 2015.
- [7] K. Shibutani, T. Isobe, H. Hiwatari, A. Mitsuda, T. Akishita, and T. Shirai. Piccolo: An ultra-lightweight blockcipher. *CHES 2011, ser. Lecture Notes in Computer Science, Springer*, 6917:342–357, 2011.
- [8] V. M. Frederik Armknecht. On lightweight stream ciphers with shorter internal states. *FSE - 22nd International Workshop on Fast Software Encryption*, 2015.

- [9] O. Kara and M. F. Esgin. Practical cryptanalysis of full sprout with tmd tradeoff attacks. *SAC*, pages 67–85, 2015.
- [10] V. Lallemand and M. N. Plasencia. Cryptanalysis of full sprout. *Advances in Cryptology CRYPTO 2015*, pages 663–682, 2015.
- [11] O. Kara and M. F. Esgin. On analysis of lightweight stream ciphers with keyed update. *IEEE Transactions on Computers PP(99)*, 2018.
- [12] B. Zhang and X. Gong. Another tradeoff attack on sprout-likestream ciphers. *Advances in Cryptology - ASIACRYPT 2015, ser. Lecture Notes in Computer Science*, 9453:561–585, 2015.
- [13] S. Maitra, S. Sarkar, A. Baksi, and P. Dey. Key recovery from state information of sprout: Application to cryptanalysis and fault attack. *Cryptology ePrint Archive, Report 2015/236*, 2015.
- [14] V. Mikhalev, F. Armknecht, and C. Mueller. On ciphers that continuously access the non-volatile key. *FSE - 24th International Workshop on Fast Software Encryption*, 2017.
- [15] V. A. Ghafari, H. Hu, and Y. Chen. Fruit-v2: Ultra-lightweight stream cipher with shorter internal state. *IACR Cryptology ePrint Archive 2016*, 2016.
- [16] S. Dey and S. Sarkar. Cryptanalysis of full round fruit. *IACR Cryptology ePrint Archive 2017*, 2017.
- [17] <https://medium.com/asecuritysite-when-bob-met-alice/light-weight-cryptography-trivium-aa986f0b881>, 2018.
- [18] [https://www.researchgate.net/figure/General-structure-of-a-synchronous-stream-cipher-encryption-and-decryption\\_fig1\\_267858656](https://www.researchgate.net/figure/General-structure-of-a-synchronous-stream-cipher-encryption-and-decryption_fig1_267858656), 2010.
- [19] [https://www.researchgate.net/figure/Self-Synchronizing-Stream-Cipher-Scheme\\_fig7\\_326209454](https://www.researchgate.net/figure/Self-Synchronizing-Stream-Cipher-Scheme_fig7_326209454), 2018.
- [20] <https://www.cs.princeton.edu/courses/archive/spring19/cos126/assignments/lfsr/>, 2019.
- [21] <https://tr.m.wikipedia.org/wiki/Dosya:A5-1.png>, 2019.

- [22] T. Siegenthaler. Decrypting a class of stream ciphers using ciphertext only. *Computers, IEEE Transactions*, 1985.
- [23] A. Biryukov and A. Shamir. Cryptanalytic time/memory/data tradeoffs for stream ciphers. *Advances in Cryptology ASIACRYPT 2000*, 2000.
- [24] D. Coppersmith, S. Halevi, and C. Jutla. Cryptanalysis of stream ciphers with linear masking. *Advances in Cryptology CRYPTO 2002*, 2002.
- [25] A. Joux and F. Muller. A chosen iv attack against turing. *Selected Areas in Cryptography, volume 3006 of Lecture Notes in Computer Science*, 2004.
- [26] J. J. Hoch and A. Shamir. Fault analysis of stream ciphers. *Cryptographic Hardware and Embedded Systems CHES 2004, Lecture Notes in Computer Science*, 2004.
- [27] <http://www.ecrypt.eu.org/stream>. estream: Ecrypt stream cipher project, ist-2002-507932. *ECRYPT*, 2002.
- [28] M. Hell, T. Johansson, and W. Meier. Grain - a stream cipher for constrained environments. *International Journal of Wireless and Mobile Computing 2(1)*, pages 86–93, 2007.
- [29] M. Hell, T. Johansson, A. Maximov, and W. Meier. A stream cipher proposal: Grain-128. *EEE International Symposium on Information Theory*, 2006.
- [30] M. Agren, M. Hell, T. Johansson, and W. Meier. Grain-128a: A new version of grain-128 with optional authentication. *International Journal of Wireless and Mobile Computing 5(1)*, pages 48–59, 2011.
- [31] [https://www.researchgate.net/figure/Key-and-IV-length-in-Grain-Family\\_ofCiphers\\_tbl1\\_307643189](https://www.researchgate.net/figure/Key-and-IV-length-in-Grain-Family_ofCiphers_tbl1_307643189), 2014.
- [32] [https://www.cryptolux.org/mediawiki-esc2015/images/d/d0/Small-ESC\\_Armknecht.pdf](https://www.cryptolux.org/mediawiki-esc2015/images/d/d0/Small-ESC_Armknecht.pdf), 2015.
- [33] M. E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions On Information Theory, Vol. IT-26, NO. 4*, 1980.