

國立交通大學

資訊工程系

碩士論文

感測器網路之安全串流密碼設計

Design of Secure Stream Ciphers on Sensor Networks



研究生：蔡志彬

指導教授：陳榮傑 教授

中華民國九十四年六月

感測器網路之安全串流密碼設計

Design of Secure Stream Ciphers on Sensor Networks

研究生：蔡志彬

Student: Jr-Bin Tsai

指導教授：陳榮傑

Advisor: Dr. Rong-Jaye Chen

國立交通大學

資訊工程研究系



Submitted to Department of Computer Science and
Information Engineering

College of Electrical Engineering and Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of Master

in

Computer Science and Information Engineering

June 2005

Hsinchu, Taiwan, Republic of China

中華民國九十四年六月

摘要

無線微型感測器網路近來愈來愈受到重視，其主要的元件即為感測器。感測器是個輕巧短小、容易大量散佈的裝置，有著簡單的處理器以及特殊的感應器可以輕易的被程式化來收集相關物理性質的資料。除此之外，感測器具備有無線通訊的能力，透過無線電的傳輸以及 Ad-hoc routing 的機制可以將收集到的相關資訊即時、動態的傳送至後端的數據中心做進一步的分析處理。無線感測器網路的目的即是在建立一個通用、有效能、低花費以及容易散佈的感測器網路平台。在此網路平台，使用者將可快速且輕易的在感測器上開發各種應用。

無線微型感測器網路有網路安全問題，無線嵌入式應用程式包含許多已知，甚至是未知的安全漏洞，而利用這些漏洞所引發的攻擊事件也日漸頻繁，對於系統的安全更是一大隱憂，而今日無線微型感測器網路，其安全機制的發展受制於其計算及通訊平台考量，因此若要建立一系列相關的安全機制來防範攻擊，還需依整個系統平台的特性限制加以考量，我們將發展一套滿足感測器所有限制的串流密碼系統，然後把它應用到感測器中，我們將探討各種串流密碼系統，找出一套最適合應用在無線微型感測器網路的串流密碼系統。

Abstract

There is more and more emphasis on Wireless Sensor Network in recent years. Sensor is a weightless device that is easy to be deployed and programmable to collect outward related information. Besides, the sensors also possess the communication ability transmitting the environmental information to the read-end data center. The objective of Wireless Sensor Networks is to build a sniffing network platform with the following features: 'Efficiency', 'Low-cost' and 'Ease of deployment'. With this platform, we can develop rapidly and easily diverse applications.

The Wireless Sensor Network has a network security problem. Wireless embedded system applications contain security flaws both known and unknown attacks that take advantage of these flaws have become ubiquitous. Protecting wireless embedded system applications from attacks requires the development of a suite of security related services, which are designed according to the constraints of wireless embedded platforms. We will develop the stream cipher to satisfy all constrains of sensors. We will explore all kinds of stream ciphers and find the fittest stream cipher to apply on Wireless Sensor Networks.

誌謝

論文的完成，要感謝許多人的幫助；首先要感謝我的指導老師陳榮傑教授，在論文的 research 過程中，給了我不少的意見和方向，讓我可以順利完成這份論文，也感謝張錫嘉教授，告訴我另一方向的研究，讓我的論文更加多元化；也感謝口試委員們，張仁俊學長和謝續平教授，在口試中，提出了許多寶貴的意見，讓我了解到我論文的不足，也讓我的論文更加完整。

此外也感謝實驗室的學長們：胡均祥學長，黃凱群學長，林志賢學長，吳緯凱學長，鄭文鼎學長，感謝他們可以提供意見給我，以及解決我的問題。還有和我同實驗室同學們：陳政愷，劉韋廷，梁漢璋，謝謝他們平時的建議和幫助。也感謝其他朋友們，謝謝你們平時的關心以及鼓勵。

最後，我要感謝我的家人，感謝你們對我無悔的付出，全心全意的支持我，相信我，讓我能順利完成我的學業，僅以此論文獻給我最愛的家人們。

Contents

Chinese Abstract	i
English Abstract	ii
Acknowledgement	iii
Contents	iv
List of Figures	vi
List of Tables	vii
Chapter 1 Introduction	1
1.1 Wireless Sensor Network	1
1.2 Stream ciphers	2
1.3 The organization of the thesis	3
Chapter 2 Stream Cipher	5
2.1 Linear feedback shift registers	10
2.2 Boolean functions	14
2.3 LFSR-based keystream generators	25
Chapter 3 Cryptanalysis on Stream Ciphers	28
3.1 The divide and conquer attack	28
3.2 The best affine approximation attack	34
3.3 The algebraic attack	36
3.4 Other attacks on filter generators	43
Chapter 4 Design of Stream Ciphers on Sensors	48
4.1 Implementing the stream cipher with software	52
4.2 Implementing the stream cipher with hardware	55
4.3 Analyzing the security and conclusion	58
Chapter 5 Conclusion and Future Research	63

List of Figure

Figure 2.1: Cryptosystem	5
Figure 2.2: Finite state machine of synchronous stream cipher	7
Figure 2.3: Finite state machine of self-synchronous stream cipher	8
Figure 2.4: Binary additive stream cipher architecture	9
Figure 2.5 Feedback shift register	10
Figure 2.6: Linear feedback shift register	11
Figure 2.7: The filter generator	25
Figure 2.8: The combination generator	26
Figure 2.9: The clock-control generator	26
Figure 2.10 Geffe clock-clock generator	27
Figure 3.1: The stream cipher constructed by the combination generator	29
Figure 3.2: Probability density function for H_0 and H_1	30
Figure 3.3: The filter generator	44
Figure 4.1: Scheme of block B	50
Figure 4.2: f_n	50
Figure 4.3: Memory	56
Figure 4.4: New instruction	57

List of Table

Table 1: Affine functions of three variables	16
Table 2: Hamming distance between $f(x)$ and affine functions	18
Table 3: Truth table of $f(X_1, X_2) = X_1 X_2$	22
Table 4: Different methods to obtain low degree equations from keystream bits ...	41
Table 5: Characteristics of prototype sensors	52
Table 6: Code size of three stream cipher	54
Table 7: Code size and execution time of the stream cipher	57
Table 8: Statistic test table of StreamCipher2	61
Table 9: Complexity of attacks	61
Table 10: Comparison among the filter generator, RC5, and A5	62



Chapter 1

Introduction

1.1 Wireless Sensor Network

There is more and more emphasis on Wireless Sensor Network in recent years, and ‘Sensor’ is a main component in this kind of network. Sensor is a weightless device with unsophisticated microprocessor and specific detector so that it is easy to be deployed and programmable to collect outward related information, such as temperature, pressure, strength of earth quake, etc. Besides, the sensors also possessed communication ability, transmitting the environmental information to the read-end data center through RF signaling and some ad-hoc routing methodology to make more advanced data analysis. The objective of Wireless Sensor Networks is to build a generic network platform with the following features: ‘Efficiency’, ‘Low-cost’ and ‘Ease of deployment’. With this platform, we can develop rapidly and easily diverse applications.

We have met some problem when building a modern Wireless Sensor Network; one of the problems is power consumption. A Wireless Sensor is a mobile device; for its mobility, and its power must be provided by the device itself. The power supply for a Wireless Sensor is for example a battery. A Wireless Sensor Network needs a long lasting battery in order to survive for a long period of time. The other problem is the network security problem. Wireless embedded system applications contain security flaws; both known and unknown attacks taking advantage of these flaws have become ubiquitous. A serious problem that limits the deployment and acceptance of wireless embedded system applications today is the lack of security services that are designed according to the unique properties of the computation and

communication platform. Protecting wireless embedded system applications from attacks requires the development of a suite of security related services, which are designed according to the constraints of wireless embedded platforms.

Because the sensors have limited processing power, storage, bandwidth, and energy, they need a different design of network security. Public key cryptosystem using large and complicated computation, such as RSA, Elliptic Curves Cryptosystem, is not suitable for a Wireless Sensor Network. Some block cipher systems in cryptosystems are also not suitable for a Wireless Sensor Network, due to their complicated architecture and long time computing. Stream cipher, differing from block cipher system, possesses a quite simple architecture and very fast encrypting rate and is very suitable for sensors which have limited memory and computing resources. Therefore, we plan to apply stream cipher on Wireless Sensor Networks.



1.2 Stream ciphers

We classify the modern cryptosystems into two groups: one is public key cryptosystem and the other is the secret key cryptosystem. Secret key cryptosystems are divided into block cipher and stream cipher. Block cipher includes Triple-DES, AES, RC6 and so on. Stream cipher usually uses linear feedback shift registers and a Boolean function as a random number generator. Then the method of encryption is to use the keystream which the random number generator produces to xor with the plaintext a bit at a time. The decryption is the same with the encryption, which is to use ciphertext to xor the keystream. Compared with secret key cryptosystems the rate of computing of public key cryptosystems is much slower. The encrypting rate of stream cipher is the fastest among secret key cryptosystems. Because of having

limited or no error propagation, stream cipher may also be advantageous in situations where transmission errors are highly probable.

Stream cipher has been developed very fast in recent years. Now in industries there are a series of related techniques of stream cipher, such as E_0 [1] in Bluetooth which is designed in the wireless network interface, and A5[2,3] in GSM and SOBER[4] which is software in the microprocessor. We plan to find out the most suitable stream cipher for Wireless Sensor Networks.

1.3 The organization of the thesis

The rest of this thesis is organized as follows. In Chapter 2, we first introduce several important components in stream cipher systems: linear feedback shift registers, Boolean functions, and the secret keystream. We will also introduce the period, the randomness and linear complexity of linear feedback shift register. Next, we will introduce the basic properties of Boolean function: balancedness, correlation-immunity, nonlinearity, algebraic degree, propagation characteristics and algebraic immunity, and explore their relations with Walsh transformation.

In Chapter 3, we introduce various attacks on stream cipher. They include correlation attacks which are to use relations between inputs and outputs of the Boolean function to attack, the best affine attack which is to use a single linear feedback shift register to approximate the secret keystream which a generator produces, and the algebraic attack which is to find the key by solving an overdefined system of algebraic equations. At last we also introduce other attacks for the filter generator.

In Chapter 4, we will propose various kinds of stream ciphers and implement them with software on sensors. We hope to achieve a balance between security and

all constrains on sensors. By comparing various stream ciphers we may find the fittest one on Wireless Sensor Networks. We also try to implement it with hardware.

Then we analyze the security of the stream cipher we choose.

Finally, the conclusion is given in Chapter 5.



Chapter 2

Stream Cipher

The main purpose of the cryptographic research theory is how to deliver the secret information securely and reliably to another end on a network. Except for people possessing a particular secret key can extract the real information from the information delivered, no one can understand the contents in it. A cryptosystem approximately contains three main parts : encoding function E_k , decoding function D_k , and the key k respectively. The practical method of encryption and decryption in the cryptosystem is illustrated as Figure 1. Alice wants to deliver a message m to Bob, but does not want other ones to know except Bob. So Alice and Bob should in advance decide a key k which is suitable for some cryptosystem. Then Alice uses the encoding function together with the key k to encrypt message m to get ciphertext C and deliver it to Bob. After Bob receives ciphertext C , he uses the decoding function together with the key k to decrypt this ciphertext to get correct original message m .

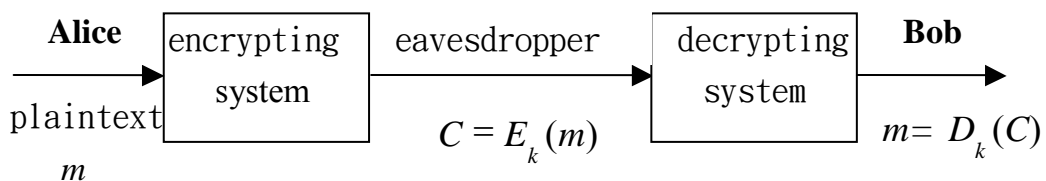


Figure 2.1: Cryptosystem

Cryptosystems are classified into two classes, the public key cryptosystem and the secret key cryptosystem, according to different forms of key. Secret key systems are approximately classified into the block cipher and the stream cipher. As far as the block cipher system is concerned, this cryptosystem is to divide the plaintext into

several blocks and then to encrypt each block one by one. For example, let $m = m_1m_2\dots m_{2L}$ be the plaintext to be encrypted. The block cipher usually divides the plaintext m into L fixed length blocks and encrypts each block respectively. Taking the plaintext m for example, m will be divided into $M_1 = m_1m_2\dots m_L$ and $M_2 = m_{L+1}m_{L+2}\dots m_{2L}$ two blocks. Block cipher will encrypt these two blocks one by one to get two ciphertexts $C_1 = c_1c_2\dots c_L$ and $C_2 = c_{L+1}c_{L+2}\dots c_{2L}$.

$$C_i = E_k(M_i) \quad i = 1, 2 \quad (1)$$

We can find that encoding function E_k does not vary with time in block ciphers. That is when M_1 is equal to M_2 , C_1 will be equal to C_2 . In general, to protect the security of block ciphers L is usually large.

The method of encryption in stream cipher system is quite different. The stream cipher contains a keystream generator which produces a pseudorandom sequence, called the keystream $Z = z_1z_2\dots$. The stream cipher uses this keystream to encrypt every digit in the plaintext one by one to get the ciphertext:

$$c_i = E_{z_i}(m_i) \quad i \geq 1 \quad (2)$$

c_i , z_i and m_i represent i -th digit in the ciphertext, the keystream and the plaintext respectively. We can see obviously that the encoding function E_{z_i} in the stream cipher varies with time.

According to the different architectures of the keystream generators, we can classify stream ciphers into two groups, synchronous stream cipher and self-synchronous stream cipher [5].

Definition 2.1: A synchronous stream cipher is one in which the keystream is generated independently of the plaintext message and of the ciphertext.

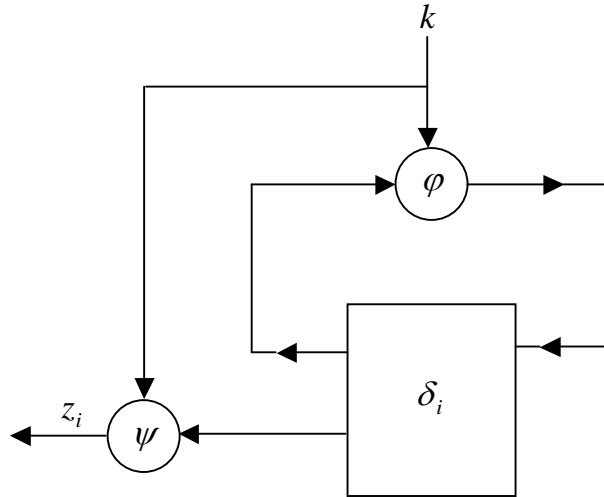


Figure 2.2: Finite state machine of synchronous stream cipher

A synchronous stream cipher can be represented by the finite state machine, shown in Figure 2.2. If the state of the keystream generator is δ_i in i -th time, the state of the keystream generator in $i+1$ -th time can be written by

$$\delta_{i+1} = \varphi(k, \delta_i) \quad (3)$$

And the keystream of i -th time z_i can be produced as follows:

$$z_i = \psi(k, \delta_i) \quad (4)$$

where k is a secret key of the stream cipher cryptosystem and δ_0 is the initial state of the keystream generator. In the synchronous stream cipher, the keystream generator of the transmitter and the receiver must be synchronous. What is so-called synchronization is to have the same secret key, initial state σ_0 and clock. Otherwise, if these two generators are not synchronous, the decrypting will fail and at this time the system need provide some auxiliary method to help to resynchronize.

Next we see the other kind of stream cipher.

Definition 2.2: A self-synchronous or asynchronous stream cipher is one in which the keystream is generated as a function of the key and a fixed number of previous

ciphertext digits.

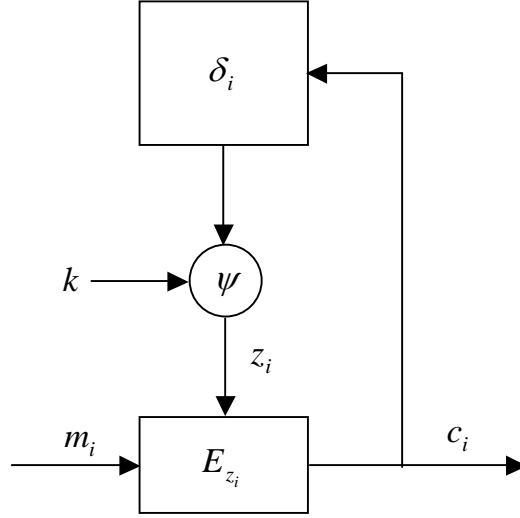


Figure 2.3: Finite state machine of self-synchronous stream cipher

A self-synchronous stream cipher can also be represented by a finite state machine, as shown in Figure 2.3. And the state of the keystream generator in i -th time is $\delta_i = (c_{i-1}, c_{i-2}, \dots, c_{i-t})$ and the keystream of i -th time z_i can be represented by:

$$z_j = \psi(k, \delta_i) \quad (5)$$

From the above equation we can see that we must define an initial state $\sigma_0 = (c_{-1}, c_{-2}, \dots, c_{-t})$, so this initial state is public. Such stream cipher is capable of re-establishing proper decryption automatically after loss of synchronization, only with a fixed number of plaintext characters unrecoverable.

In this thesis, the stream cipher is discussed in $GF(2)$. We usually use XOR to be the encrypting and decrypting function in the stream cipher with respect to $GF(2)$.

$$c_j = E_{z_i}(m_i) = m_i \oplus z_i \quad (6)$$

$$m_j = D_{z_i}(c_i) = c_i \oplus z_i \quad (7)$$

Such stream cipher is called a binary additive stream cipher. A simple binary additive stream cipher architecture is given as Figure 2.4 illustrates.

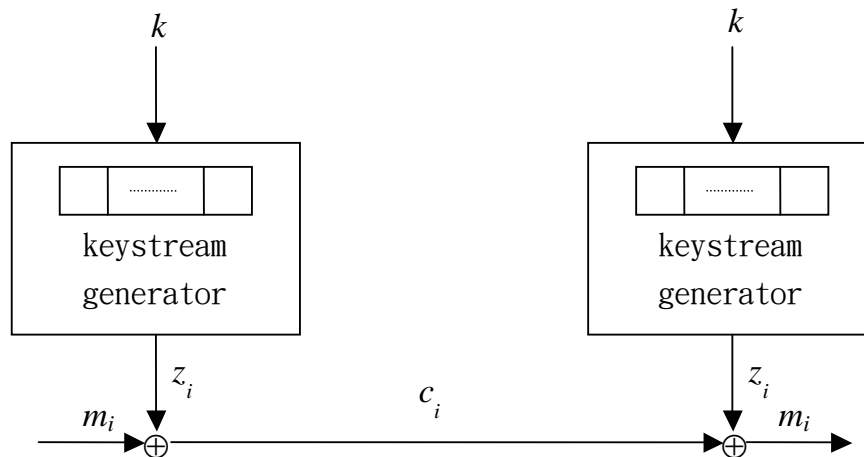


Figure 2.4: Binary additive stream cipher architecture

For a binary additive stream cipher architecture, its security is totally based on the keystream generator. There have been a lot of relative research [4,6,7] on this aspect. In summary the keystream of a secure keystream generator must meet the following conditions.

- (1) The keystream needs to have the very large period and usually this period is not smaller than 3×10^{16} or 2^{55} .
- (2) The distribution of 0 and 1 between the keystream sequence must be random enough.
- (3) The linear complexity of the stream cipher generator must be big enough. What is so-called high or low in the linear complexity is to point whether we can use a single LFSR only to produce the same keystream sequence of the stream cipher generator. With respect to the linear complexity and LFSR, we will explore in detail in section 2.2.
- (4) No matter what statistic test we use to compute does not get any information of the secret key from the keystream.

2.1 Linear feedback shift registers

The component which the stream cipher uses most frequently is the feedback shift register. The feedback shift register can very fast produce binary sequences.

The method of production is as Figure 2.5 illustrates.

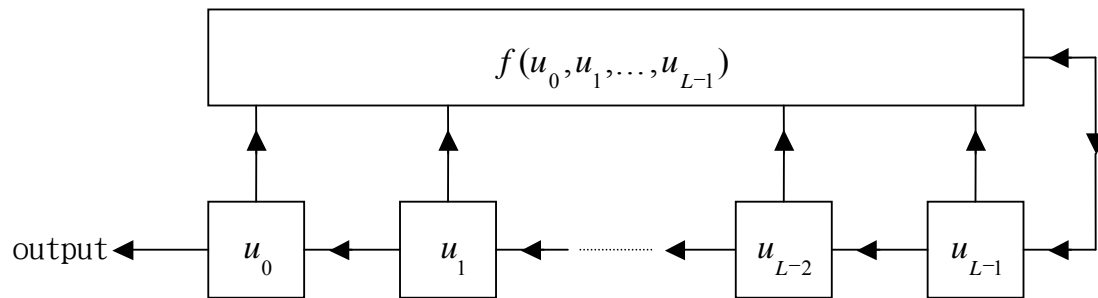


Figure 2.5 Feedback shift register

The feedback shift registers of the length L contains L stages which compose the states of the registers. Each stage is called one degree of the register and L is called degree or length of registers. The function f is called the feedback function or the connection function of feedback shift register and the method of its operation is as follows: The user can assign the first initial state $(u_0, u_1, \dots, u_{L-1})$ and the first output is u_0 and then the state changes (u_1, u_2, \dots, u_L) where $u_L = f(u_0, u_1, \dots, u_{L-1})$. This is to say when the state makes a shift and u_L will be into the end of the register. When j -th shift clock impulse comes, its output is u_j and the state of the feedback shift register $(u_j, u_{j+1}, \dots, u_{j+L-1})$ changes into $(u_{j+1}, u_{j+2}, \dots, u_{j+L})$ where $u_{j+L} = f(u_j, u_{j+1}, \dots, u_{j+L-1})$. For the same reason it will produce infinite sequences $u = \{u_i\}_{i \geq 0}$.

For infinite sequences u , if the positive integer $T > 0$ exists, $u_{i+T} = u_i$ for all $i \geq 0$ will be true and then this sequence is called the periodic sequence and the smallest value of all T is called the period of this sequence. Because the feedback shift register has L stages, at most 2^L different states exist. So when the output sequence

is very long, the state will appear repeatedly. Therefore the output sequence of feedback shift register must be a periodic sequence. Because the result is zero by computing feedback function in the state of all 0 in L stages, the state will be all zero. So the state of all zero is taken off and there are 2^L-1 different states. The period of the generated infinite sequence u , T , is smaller than or equal to 2^L-1 .

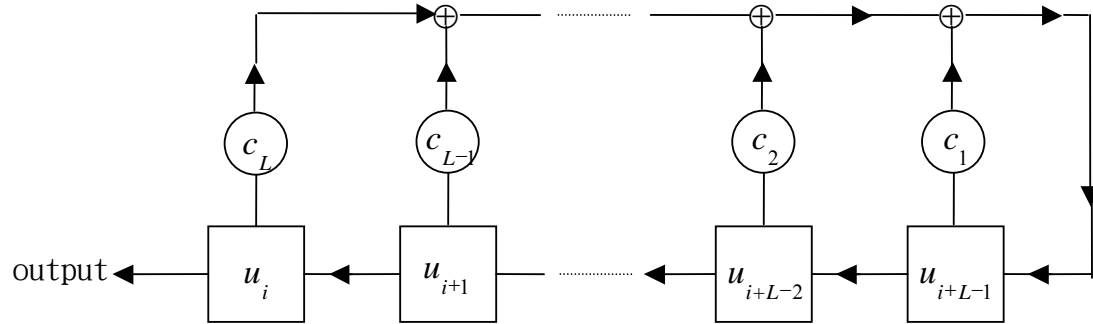


Figure 2.6: Linear feedback shift register

In general, the feedback function often takes the linear function as the architecture. At this time, we call this the linear feedback shift register and the feedback function can be represented by:

$$f(u_0, u_1, \dots, u_{L-1}) = c_0 u_0 + c_1 u_1 + c_2 u_2 + \dots + c_{L-1} u_{L-1} \quad (8)$$

where $c_i \in \{0,1\}$ ($0 \leq i \leq L-1$) and these additive operations are module 2. We can represent the output sequence $u = \{u_i\}_{i \geq 0}$ by the recursive relative equation:

$$u_j = \sum_{i=1}^L c_i u_{j-i} \quad , \quad j \geq L. \quad (9)$$

Figure 2.6 is the general structure of the linear feedback shift register of degree L . We call coefficient c_i feedback coefficient and from feedback coefficient we can define the feedback polynomial of LFSR $g(x)$:

$$g(x) = 1 + c_1 x + c_2 x^2 + \dots + c_{L-1} x^{L-1} + c_L x^L \quad , \quad c_L = 1 \quad (10)$$

where L is called the degree of the feedback polynomial, and the number of

feedback coefficient c_i ($0 \leq i \leq L-1$) which is not zero is called tap of the feedback polynomial.

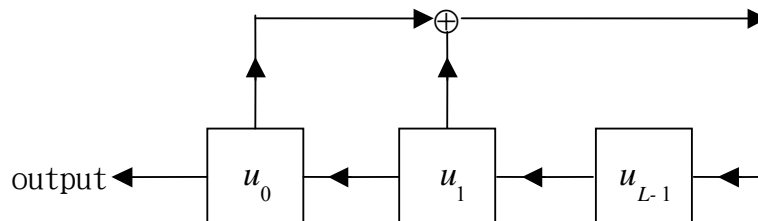
Next, we talk about the relation between the feedback polynomial and the period in LFSR. Let the length of LFSR be L and its output sequence is 2^L-1 . We call this sequence m-sequence and call this LFSR maximal-length LFSR.

Definition 2.3: For a polynomial of degree L $g(x)$, if two polynomials of degree smaller than L $a(x)$, $b(x)$ exist and $g(x) = a(x) \cdot b(x)$ is true, then we call this polynomial reducible polynomial; reversely, call it irreducible polynomial.

Definition 2.4: Let x be a root of a irreducible polynomial of degree L $g(x)$. If x is a generator in multiplicative group Z_{2^L} , we call $g(x)$ a primitive polynomial.

Theorem 2.1: If a feedback polynomial of LFSR of length L $g(x)$ is a primitive polynomial, the sequence which LFSR produces will be m-sequence of the period 2^L-1 . Such LFSR is a linear feedback shift register of the longest period.

Example 2.1: Let $L=3$, $c_1=0$, $c_2=1$, $c_3=1$ and the initial state be $(1, 0, 0)$. Then the structural chart and the continuous change of LFSR are as follows.



Time	0	1	0	0
	1	0	0	1
	2	0	1	0
	3	1	0	1
	4	0	1	1
	5	1	1	1
	6	1	1	0
	7	1	0	0

The feedback polynomial of LFSR is $g(x) = 1+x^2+x^3$ and its tap is two. We observe that the state of the register in $t = 7$ is the same with that in $t = 0$. This represents the recursive state of the output sequence has happened. Therefore, we know the period of output sequence is 7, that is, 2^3-1 and this sequence achieves the largest period of LFSR of degree 3. So, the feedback polynomial of this LFSR is a primitive one.

In section 2.1, we have proposed the conditions of the good keystream, one of which must be random enough, that is, the keystream sequence is highly unpredictable. We can not find in previous research a complete theory to judge whether a keystream is random. We just use the statistic tests to explain whether the distribution between 0 and 1 of the keystream sequence is close to some probabilistic distribution, that is, it achieves enough randomness. The usual statistic test methods have frequency test, serial test, poker test, run test and autocorrelation test, etc. If a binary sequence can satisfy these testes, we can say approximately that this sequence is unpredictable. Until 1967 Golomb proposed the three standards of testing randomness of the binary periodic sequence [6] with respect to whether it is enough random or not. A sequence which satisfies these three standards is called the pseudo-random sequence, or the P-N sequence. We can briefly prove that an m-sequence which is generated by the linear feedback shift register of the longest period is a P-N sequence.

Another condition which the keystream must satisfy is the linear complexity

and this must be large enough. This concept was presented by A Lempel and J. Ziv [8] in 1976. The linear complexity of a periodic sequence means that it possesses higher unpredictability. The size of the linear complexity of the keystream sequence is an important guideline for the secure strength of the stream cipher system.

Definition 2.5: The linear complexity of a periodic sequence m is denoted by $\Lambda(m)$ which is defined by the smallest degree of LFSR which can produce the sequence m . When $m = 0$, define $\Lambda(m) = 0$.

Therefore the size of the linear complexity of the keystream is very important for a stream cipher. We can get the linear complexity of a sequence easily by the Berlekamp-Massey algorithm [9]. Given a sequence of the length n , we can get its linear complexity by B-M algorithm within $O(n^2)$.

Through the above description, a good LFSR must satisfy three conditions: the large period, the unpredictability and the large linear complexity. When we construct the keystream generator of the stream cipher, we must find LFSR that satisfies these three conditions to produce the secret keystream. But usually we use a nonlinear Boolean function combining several LFSRs in order to strengthen the randomness and the linear complexity of the keystream sequence. We will introduce Boolean functions in detail in the next section.

2.2 Boolean functions

First, let $GF(2) = V$ and $x = (X_1, X_2, \dots, X_n) \in V^n$ is a vector of n elements, where each $X_i \in V^n$ ($1 \leq i \leq n$). The definition of a Boolean function $f(x)$ is the function from V^n to V , that is, $f(x) | V^n \rightarrow V$. We denote all Boolean functions of n variables by Ω_n or $\Omega_V(n)$. Any Boolean function of Ω_n $f(x)$ can be represented

uniquely by the algebraic normal form, called ANF for short, and this representation is as follows:

$$f(x) = a_0 + a_1 X_1 + \dots + a_n X_n + a_{12} X_1 X_2 + a_{13} X_1 X_3 + \dots + a_{12\dots n} X_1 X_2 \dots X_n \quad (11)$$

where operations of the addition and multiplication in the function are all based on \mathbb{V} . The outputs of Boolean function $f(x)$ form the $(0, 1)$ sequence called truth table, denoted by f or T_f .

Let S be a finite set and the number of elements in set S is denoted by $\#S$. Let $f(x)$ and $g(x)$ be two Boolean functions of n variables and the hamming weight of $f(x)$ is defined by the number of one in the truth table of $f(x)$ and is denoted by $wt(f(x))$ or $w(f)$, that is, $wt(f) = \#\{x \mid f(x) = 1\}$. The hamming distance of two Boolean functions is defined by the different numbers between the truth table of them, that is, $d(f, g) = \#\{x \mid f(x) \neq g(x)\} = wt(f + g)$, denoted by $d(f(x), g(x))$ or $d(f, g)$. Next define two important properties of a Boolean function: balancedness and algebraic degree.

Definition 2.6: If $wt(f)$ is 2^{n-1} for a Boolean function of n variables $f(x)$, then $f(x)$ possesses balancedness.

Definition 2.7: The algebraic degree of a Boolean function of n variables $f(x)$ is denoted by $deg(f)$, that is, the number of the biggest term in function.

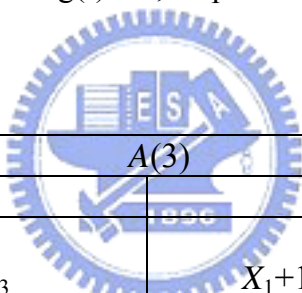
Example 2.2: Let $f(x), g(x) \in \Omega_2$, $f(x) = X_1 + X_2$ and $g(x) = X_1 X_2$ and the truth tables of $f(x)$ and $g(x)$ are $T_f = (0,1,1,0)$ and $T_g = (0,0,0,1)$ respectively. Algebraic degree and hamming distance of these two functions are $deg(f) = 1$, $deg(g) = 2$ and $wt(f) = 2$, $wt(g) = 1$ respectively. From the definition of balancedness, we can know $f(x)$ is balancedness, but $g(x)$ is not. Besides, hamming distance between two functions is $d(f, g) = 3$.

Let $w = (W_1, W_2, \dots, W_n)$, $x = (X_1, X_2, \dots, X_n) \in V^n$ and we define one operation: $w \bullet x = W_1X_1 + W_2X_2 + \dots + W_nX_n$ are inner product of two vectors of length n . Next we define an affine function and a linear function.

Definition 2.8: A Boolean function of n variables and $\deg(f) \leq 1$ can be represented by $f(x) = W_0 + W_1X_1 + W_2X_2 + \dots + W_nX_n$, where $W_i \in V$ ($0 \leq i \leq n$). We call such form of Boolean function an affine function. If $W_0 = 0$, then such function is a linear function. The set which all affine functions form is denoted by $A(n)$ or $A_V(n)$; The set which all linear functions form is denoted by $L_n(x)$ or $L_V(n)$.

A linear function of n variables $f(x)$ can be represented by $f(x) = w \bullet x$.

Example 2.3: Let $f(x) \in \Omega_2$ and $\deg(f) \leq 1$, all possible $f(x)$ are represented as follows:



$A(3)$	
$L(3)$	
X_1, X_2, X_3	$0, 1$
$X_1+X_2, X_1+X_3, X_1+X_3$	X_1+1, X_2+1, X_3+1
$X_1+X_2+X_3$	$X_1+X_2+1, X_1+X_3+1, X_1+X_3+1$
	$X_1+X_2+X_3+1$

Table 1: Affine functions of three variables

The Walsh transform of a Boolean function is defined as follows:

Definition 2.9: The Walsh transform of a Boolean function of n variables is denoted by $F_f(w)$ or $F_{f(x)}(w)$ and is defined by

$$F_f(w) = \sum_{x \in [GF(2)]^n} (-1)^{f(x)+w \bullet x} \quad (13)$$

Besides, let $F_{(f)}(w) = \frac{1}{2^n} \cdot F_f(w)$

We regard $w \bullet x$ as a linear function and the meaning of the Walsh transform of the Boolean function of n variables $f(x)$ is regarded as $F_f(w) = \#\{x \mid f(x) = w \bullet x\} - \#\{x \mid f(x) \neq w \bullet x\}$, that is, the number of x in $f(x)$ equal to $w \bullet x$ subtracts the number of x in $f(x)$ not equal to $w \bullet x$. Therefore we can easily infer the relation between the Walsh transform and $d(f, w \bullet x)$.

$$\begin{aligned}
 F_f(w) &= \#\{x \mid f(x) = w \bullet x\} - \#\{x \mid f(x) \neq w \bullet x\} \\
 &= 2^n - 2 \cdot \#\{x \mid f(x) \neq w \bullet x\} \\
 &= 2^n - 2 \cdot d(f, w \bullet x) \\
 \Leftrightarrow d(f, w \bullet x) &= 2^{n-1} - F_f(w)/2
 \end{aligned} \tag{14}$$

Definition 2.10: Nonlinearity of a Boolean function of n variables is denoted by N_f or $N_{f(x)}$ and is defined by the minimum distance between $f(x)$ and all affine functions of n variables. That is

$$N_f = \min_{w \bullet x \in A(n)} \{d(f, w \bullet x)\} \tag{15}$$

Besides, an affine function $w \bullet x$ which is minimally distant from a Boolean function $f(x)$ is called a best affine function of $f(x)$.

From the equation (14) and the definition of nonlinearity, we can infer the relation between the Walsh transform of $f(x)$ and N_f .

Theorem 2.2: Nonlinearity of a Boolean function $f(x)$ of n variables is denoted by

$$N_f = 2^{n-1} - \frac{1}{2} \cdot \max_{w \in [GF(2)]^n} |F_f(w)| \tag{16}$$

Proof: From (14) we know

$$\begin{aligned}
 d(f, w \bullet x) &= 2^{n-1} - F_f(w)/2 \\
 \Rightarrow \min_{w \bullet x \in A(n)} \{d(f, w \bullet x)\} &= \min_{w \in [GF(2)]^n} \{2^{n-1} - F_f(w)/2\}
 \end{aligned}$$

$$\Rightarrow N_f = 2^{n-1} - \max_{w \in [GF(2)]^n} |F_f(w)| \quad \square$$

Example 2.4: $f(x) \in \Omega_3$ and the truth table of $f(x)$ $T_f = \{0, 0, 1, 1, 0, 1, 1, 0\}$

function	truth table ($c=0$)								$d(f, wx)$	
	(0,0,0)	(0,0,1)	(0,1,0)	(0,1,1)	(1,0,0)	(1,0,1)	(1,1,0)	(1,1,1)	$c=0$	$c=1$
f	0	0	1	1	0	1	1	0		
c	0	0	0	0	0	0	0	0	4	4
x_1+c	0	0	0	0	1	1	1	1	4	4
x_2+c	0	0	1	1	0	0	1	1	2	6
x_3+c	0	1	0	1	0	1	0	1	4	4
x_1+x_2+c	0	0	1	1	1	1	0	0	2	6
x_1+x_3+c	0	1	0	1	1	0	1	0	4	4
x_2+x_3+c	0	1	1	0	0	1	1	0	2	2
$x_1+x_2+x_3+c$	0	1	1	0	1	0	0	1	6	6

Table 2: Hamming distance between $f(x)$ and affine functions

From the above Table 2, we know $N_f = 2$.

Balancedness of a Boolean function can also be represented by Walsh transform of the Boolean function.

Theorem 2.3: If the Boolean function of n variables $f(x)$ possesses balancedness then if and only if $F_f(0) = 0$ will be true.

Proof: From the definition of the Walsh transform of the Boolean function $f(x)$ we

know as $w = 0$

$$F_f(0) = \sum_{x \in \mathbb{Z}_2^n} (-1)^{f(x)} = \#\{x \mid f(x) = 0\} - \#\{x \mid f(x) = 1\}$$

(1) \Leftrightarrow :

Because $f(x)$ possesses balancedness, $\#\{x \mid f(x) = 0\} = \#\{x \mid f(x) = 1\} = 2^{n-1}$. So

we prove $F_f(x) = 2^{n-1} - 2^{n-1} = 0$.

(2) \Leftarrow :

From $F_{f(x)} = 0$ so we know from (17)

$$\#\{x \mid f(x) = 0\} - \#\{x \mid f(x) = 1\} = 0$$

$$\Rightarrow \#\{x \mid f(x) = 0\} = \#\{x \mid f(x) = 1\}$$

Because $\#\{x \mid f(x) = 0\} + \#\{x \mid f(x) = 1\} = 2^n$, we can infer from (18)

$$\#\{x \mid f(x) = 0\} = \#\{x \mid f(x) = 1\} = 2^{n-1}$$

So we prove $f(x)$ possesses balancedness. \square

Next we define the relations between inputs and outputs of a Boolean function: correlation immunity, propagation characteristics and correlations respectively.

Definition 2.11: If we take randomly m variables from in a Boolean function of n variables $f(x)$ to make $\Pr[f(x) = 1 \mid X_{i_1} = a_1, \dots, X_{i_m} = a_m] = \Pr[f(x) = 1]$ true, where $a_1, a_2, \dots, a_m \in V$ and then call $f(x)$ the function of m -th correlation immunity, abbreviated the (m, n) CI function.

If a Boolean function is (m, n) CI, then in brief we can fix m variables in the Boolean function of n variables, its output distribution keeps the same original proportion. In other words, when we analyze the relation between inputs and outputs of the Boolean function, even if we fix below m variables, we can not get any hidden information from output data. If $f(x)$ is (m, n) CI and possesses balancedness, we call $f(x)$ the resilient function of order m , denoted by the (m, n) resilient function.

Theorem 2.4: if $f(x)$ is (m, n) CI, then $f(x)$ is also (t, n) CI, where $1 \leq t \leq m$.

Proof: Assume that every input variable of the Boolean function is statistic independent. As $t = m-1$, from the Baye's theorem, we know

$$\begin{aligned}
& \Pr[f(x) = 1 \mid X_{i_1} = a_1, \dots, X_{i_{m-1}} = a_{m-1}] \\
&= \Pr[X_{i_m} = 1] \Pr[f(x) = 1 \mid X_{i_1} = a_1, \dots, X_{i_{m-1}} = a_{m-1}, X_{i_m} = 1] \\
&\quad + \Pr[X_{i_m} = 0] \Pr[f(x) = 1 \mid X_{i_1} = a_1, \dots, X_{i_{m-1}} = a_{m-1}, X_{i_m} = 0] \tag{19}
\end{aligned}$$

Because $f(x)$ is (m, n) CI, (19) can be simplified to

$$\begin{aligned}
& \Pr[f(x) = 1 \mid X_{i_1} = a_1, \dots, X_{i_{m-1}} = a_{m-1}] \\
&= \Pr[X_m = 1] \cdot \Pr[f(x) = 1] + \Pr[X_m = 0] \cdot \Pr[f(x) = 1] \tag{20}
\end{aligned}$$

So from $\Pr[f(x) = 1 \mid X_{i_1} = a_1, \dots, X_{i_{m-1}} = a_{m-1}] = \Pr[f(x) = 1]$ we prove $f(x)$ is $(m-1, n)$ CI.

As $t = m - 2$, similarly from the Baye's theorem, we know

$$\begin{aligned}
& \Pr[f(x) = 1 \mid X_{i_1} = a_1, \dots, X_{i_{m-2}} = a_{m-2}] \\
&= \Pr[X_{i_{m-1}} = 1] \Pr[f(x) = 1 \mid X_{i_1} = a_1, \dots, X_{i_{m-2}} = a_{m-2}, X_{i_{m-1}} = 1] \\
&\quad + \Pr[X_{i_{m-1}} = 0] \Pr[f(x) = 1 \mid X_{i_1} = a_1, \dots, X_{i_{m-2}} = a_{m-2}, X_{i_{m-1}} = 0] \tag{21}
\end{aligned}$$

Because $f(x)$ is $(m-1, n)$ CI, (21) can be simplified to

$$\begin{aligned}
& \Pr[f(x) = 1 \mid X_{i_1} = a_1, \dots, X_{i_{m-2}} = a_{m-2}] \\
&= \Pr[X_{m-1} = 1] \cdot \Pr[f(x) = 1] + \Pr[X_{m-1} = 0] \cdot \Pr[f(x) = 1] \tag{22}
\end{aligned}$$

So from $\Pr[f(x) = 1 \mid X_{i_1} = a_1, \dots, X_{i_{m-2}} = a_{m-2}] = \Pr[f(x) = 1]$ we prove $f(x)$ is $(m-2, n)$ CI.

We use the method like above to infer and will prove $f(x)$ is (t, n) CI for all t , where $1 \leq t \leq m - 2$. □

Theorem 2.5: If $f(x)$ is (m, n) CI, $\deg(f) \leq n - m$. If $f(x)$ is (m, n) resilient, then $\deg(f) \leq n - m - 1$ where $m = n - 1$ is excluded.

Theorem 2.6: $f(x)$ is (m, n) CI for all $w \in V^n$ where $1 \leq wt(w) \leq m$ if and only if $F_f(w) = 0$.

Theorem 2.5 explains the relation between correlation immunity and algebraic degree in the function and between resilient and algebraic degree in the function.

Theorem 2.6 explains the relation between correlation immunity and Walsh transform in the function. The proofs of these two theorems are in [10] and [11].

Next we define propagation characteristics of the function.

Definition 2.12: Given an n -variable function and $\alpha \in V^n$, if $f(x) + f(x + \alpha)$ possesses balancedness, then $f(x)$ satisfies propagation characteristics for α .

Definition 2.13: Given an n -variable Boolean function and $\alpha \in V^n$:

- (1) If $f(x) + f(x + \alpha)$ possesses balancedness for all α where $wt(\alpha) = 1$, $f(x)$ satisfies strict avalanche criteria, called SAC for short.
- (2) If we fix any k input variables of $f(x)$ and $f(x)$ still satisfies SAC, $f(x)$ satisfies SAC of order k , called SAC(k) for short.
- (3) If $f(x) + f(x + \alpha)$ possesses balancedness for all α where $1 \leq wt(\alpha) \leq \alpha$, $f(x)$ satisfies propagation characteristics of degree 1, called PC(1) for short.
- (4) If we fix any k input variables of the function, $f(x)$ still satisfies PC(1), $f(x)$ satisfies PC(1) of order k , called PC _{k} (1) for short.

From (1) and (3) in definition 2.13, we can obviously see PC(1) is the same with SAC. If an n -variable function Boolean function $f(x)$ satisfies PC(n), this function possesses perfect nonlinear.

Definition 2.14: Given two binary sequences $u = \{u_i\}_{1 \leq i \leq N}$ and $s = \{s_i\}_{1 \leq i \leq N}$, the correlation between these two sequences is defined by:

$$\alpha = \frac{\#\{i | u_i = s_i\} - \#\{i | u_i \neq s_i\}}{N} \quad (23)$$

Definition 2.15: Let $x = (X_1, X_2, \dots, X_n) \in V^n$ be a vector of n elements and $f(x) \in \Omega_n$. The probabilistic relation between input variable X_i and output in $f(x)$ is defined by $\alpha_i = \Pr[X_i = f(x)] - \Pr[X_i \neq f(x)]$, where $i \in \{1, 2, \dots, n\}$. If at least one of the probabilities for all α_i are not zero, $f(x)$ possesses the correlation.

From the above definition we know the value of α_i is between 1 and -1. We take an example to explain the correlation of a Boolean function.

Example 2.5: Let $f(X_1, X_2) = X_1X_2 \in \Omega_2$, and the truth table of $f(x)$ is showed in Table 3. Obviously, for all i , $\Pr[X_i = f(x)] = 0.75$, $\Pr[X_i \neq f(x)] = 0.25$. Therefore, $\alpha_1 = \alpha_2 = 0.75 - 0.25 \neq 0$, and $f(x)$ possesses the correlation.

X_1	X_2	$f(x)$
0	0	0
0	1	0
1	0	0
1	1	1

Table 3: Truth table of $f(X_1, X_2) = X_1X_2$

After seeing some properties of the Boolean function we define a special function called a Bent function.

Definition 2.16: Let $f(x) \in \Omega_n$ and n is even. For all $w \in V^n$ if

$$2^{-\frac{n}{2}} \sum_{x \in [GF(2)]^n} (-1)^{f(x) + w \cdot x} = \pm 1 \quad (24)$$

We call $f(x)$ a Bent function.

Theorem 2.7: Let $f(x) \in \Omega_n$ and n is even. The following properties explain the same thing:

(1) $f(x)$ is a Bent function.

(2) $N_f = 2^{n-1} - 2^{\frac{n}{2}-1}$, and N_f is maximal nonlinearity in all n -variable Boolean function.

(3) $\#\{x \mid f(x) = 1\} = 2^{n-1} \pm 2^{\frac{n}{2}-1}$, $\#\{x \mid f(x) = 0\} = 2^{n-1} \mp 2^{\frac{n}{2}-1}$

(4) $f(x)$ possesses perfect nonlinearity.

From (3) in the above theorem we know $f(x)$ does not possess balancedness. The proof of theorem 2.7 is in [12]

Example 2.6: Let $f(X_1, X_2) = X_1X_2 \in \Omega_2$, and this is a 2-variable Bent function.

Nonlinearity of $f(x)$ is $N_f = 2^1 - 2^0 = 1$; From Table 3 we know $\#\{x \mid f(x) = 1\} = 1 = 2^{2-1} - 2^{1-1}$ and $\#\{x \mid f(x) = 0\} = 3 = 2^{2-1} + 2^{1-1}$. We discuss propagation characteristics of $f(x)$ in two cases:

Let $\alpha \in V^2$

case1: $\text{wt}(\alpha) = 1$

case 1-1: $\alpha = (0, 1)$

$f(x) + f(x + \alpha) = X_1X_2 + X_1(X_2 + 1) = X_1$, which possesses balancedness.

case1-2: $\alpha = (1, 0)$

$f(x) + f(x+\alpha) = X_1X_2 + X_2(X_1 + 1) = X_2$, which possesses balancedness.

case2: $\text{wt}(\alpha) = 1$

$f(x) + f(x + \alpha) = X_1X_2 + (X_1 + 1)(X_2 + 1) = X_1 + X_2 + 1$, which possesses balancedness.

From case1 and case2, we know $f(x)$ satisfies PC(2) and because $f(x)$ is a 2-variable function, $f(x)$ possesses perfect nonlinearity. \square

Next, we define a new property of the Boolean function, algebraic immunity [13, 16].

Definition 2.17:

1. Take $f, g, h \in \Omega_n$. Assume that there exists a nonzero function g of low degree such that $f * g = h$ or $(1+f) * g = h$, where h is a nonzero function of low degree and without loss of generality, $\deg(g) \leq \deg(h)$. Among all such h 's we denote the lowest degree h (may be more than one and we take any one of them) by $LDG_n(f)$.
2. Assume there exists a nonzero function g of low degree such that $f * g = 0$ or $(1+f) * g = 0$. Among all such g 's we denote the lowest degree g (may be more than one and then we take any one of them) by $LDA_n(f)$.

Definition 2.18: We define algebraic immunity of an n -variable Boolean f as

$$AI_n(f) = \deg(LDG_n(f))$$



In 1 of above definition 2.17, if $\deg(g) > \deg(h)$, then $f * h = f * f * g = f * g = h$, so one can use h in place of g , that is, we always assume $\deg(g) \leq \deg(h)$.

Let $f * g = h$. When $\deg(g) < \deg(h)$, $(1 + f) * h = h + f * h = h + f * f * g = h + f * g = h + h = 0$. And when $\deg(g) = \deg(h)$, $f * (g + h) = f * g + f * h = f * g + f * f * h = f * g + f * g = 0$. So $\deg(LDG_n(f)) = \deg(LDA_n(f))$

Example 2.7: Let $f(X_1, X_2, X_3) = X_1X_2X_3 + X_1X_2 + X_1X_3 + X_3 \in \Omega_3$. We choose $g(x) = X_1 + 1$ and

$$f * g = X_1X_2X_3 + X_1X_2 + X_1X_3 + X_3 + X_1X_2X_3 + X_1X_2 + X_1X_3 + X_1X_3 = X_1X_3 + X_3.$$

$LDG_n(f)$ is $X_1X_3 + X_3$.

We choose $g(x) = X_1X_3 + X_3$.

$$(1+f) * g = (1 + X_1X_2X_3 + X_1X_2 + X_1X_3 + X_3) * (X_1X_3 + X_3) = X_1X_3 + X_3 + X_1X_2X_3 + X_1X_2X_3 + X_1X_3 + X_1X_3 + X_1X_2X_3 + X_1X_2X_3 + X_1X_3 + X_3 = 0.$$

$LDA_n(f)$ is $g(x) = X_1X_3 + X_3$.

$$\deg(LDG_n(f)) = \deg(LDA_n(f)) = 2. \quad \square$$

2.3 LFSR-based keystream generators

Whether a stream cipher possesses high cryptographic strength or not is decided by the design of the keystream generator. A generator can be regarded as a finite state machine [14] as Figure 2.2 illustrates. It is composed of the output sequence set $\{z_i\}$, the state set $\{\delta_i\}$, two functions, φ and ψ , and an initial state δ_0 . The function of the state change $\varphi | \delta_i \rightarrow \delta_{i+1}$ is to transform the current state δ_i into next one δ_{i+1} . The output function $\psi | \delta_i \rightarrow z_i$ is to changes δ_i into z_i .

The main purpose of the design of such keystream generator is to seek appropriate φ and ψ to make the output sequence $\{z_i\}$ have good randomness and achieve several basic requests in the end of section 2.1. A general stream cipher usually uses several LFSRs and a nonlinear Boolean function to form a keystream generator. Based on combinative methods of the LFSR and the Boolean function we classify the keystream generators into three categories, the filter generator, the combination generator, the clock-control generator.

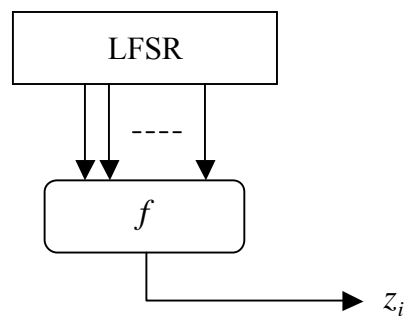


Figure 2.7: The filter generator

The structure of a filter generator is to use a single LFSR and then a nonlinear Boolean function to filter its state as Figure 2.7 shows, where a nonlinear Boolean function $f(x)$ is called the filter function. In recent research on the stream cipher, the most famous filter generator in the stream ciphers is SNOW [15].

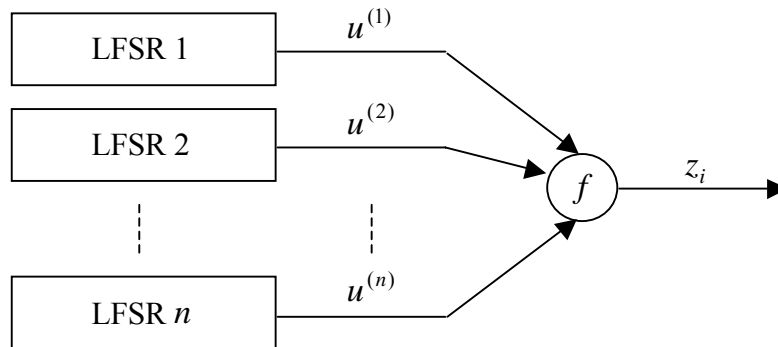


Figure 2.8: The combination generator

A combination generator is to take the outputs of the several LFSRs as inputs of a nonlinear Boolean function to produce the keystream as Figure 2.8 illustrates, where $f(x)$ is called a combining function. The application of the combination generator is very universal, such as E0 [1] in Bluetooth whose keystream generator is the combination generator.

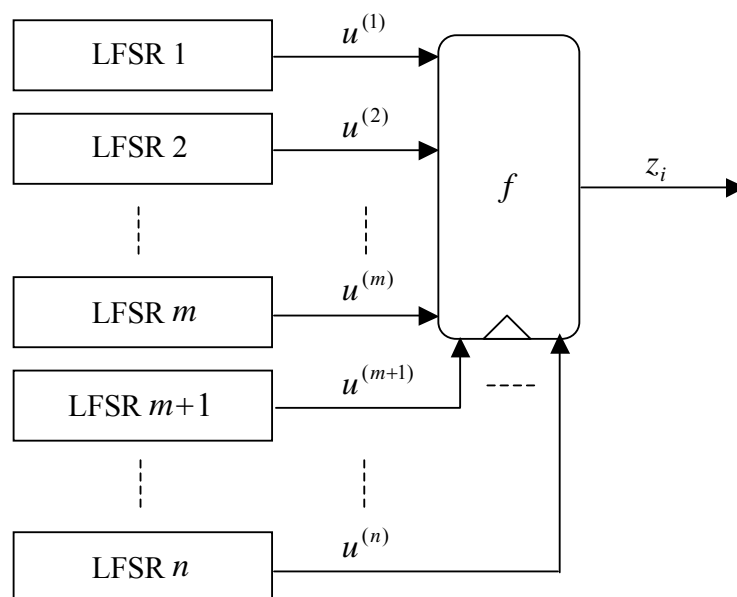


Figure 2.9: The clock-control generator

A filter generator is a special case of the combination generator, where all the combined sequences are produced by the same LFSR.

The third keystream generator is a clock-control generator. The main difference between this and the above two is that this generator uses a single or several LFSRs as the controller to produce the keystream as in Figure 2.9, where $f(x)$ is called the control function.

A5 [2,3] in GSM and SOBER [42] in microprocessor all use clock-control generators as the main structure of the stream cipher cryptosystem. A simple example of the clock-control generator is as follows:

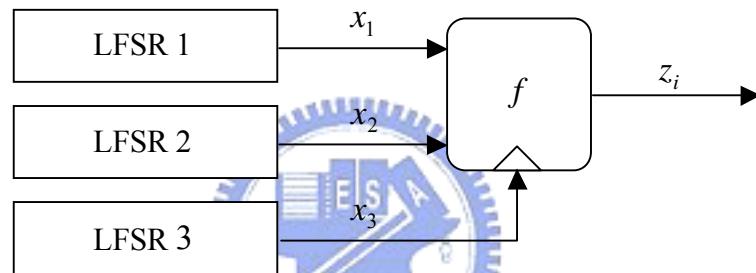


Figure 2.10 Geffe clock-control generator

Example 2.8: The Geffe clock-control generator is the keystream generator which consists of three LFSRs as in Figure 2.10, where the control function is

$$f(x) = x_1x_3 + x_2\overline{x_3} .$$

We see obviously that when the output of LFSR3 is one, the keystream is from LFSR1; On the other hand, when the output of LFSR3 is zero, the keystream is from LFSR2. Such clock-control generator is to use a single LFSR as the controller to produce the keystream.

Chapter 3

Cryptanalysis on Stream Ciphers

The keystream generator produces the pseudo-random keystream by inputting the secret key. The method of encrypting in the stream cipher is to xor the plaintext with the keystream and the method of decrypting is also xor the ciphertext with the same keystream. Therefore, if we know some pairs (plaintext, ciphertext), we know the corresponding keystream by xoring ciphertexts with plaintexts. Most attacks usually know some keystream and want to find out the secret key. A first attack is always the exhaustive search attack. When an adversary knows the keystream and the combining function, he can guess the feedback polynomial of every LFSR and its secret key. This complexity is $\prod_{j=1}^n L_j (2^{l_j} - 1)$, where n is the number of LFSRs, l_j is the degree of j -th feedback polynomial, and L_j is the number of all primitive polynomials of degree l_j . Other attacks are to improve this complexity. Because a filter generator is a special case of the combination generator, an attack on the combination generator is also able to attack the corresponding filter generator. In the following section, we will introduce all kinds of attacks. Some attacks are aimed at a combination generator, but can also attack a corresponding filter. Some attacks are aimed at both generators. And others specially attack a filter generator. From these attacks, we can construct a more secure stream cipher against all kinds of attacks.

3.1 The divide and conquer attack

The divide and conquer attack is a kind of ciphertext only attack [17] and it is to attack a combination generator. From Figure 3-1 we note C , Z , and Y have the correlation and Z and x^j have the correlation so C must contain information of the

output sequence of the LFSR x^j . Let the inputs $x_i^1, x_i^2, \dots, x_i^n$ of the function f in Figure 3-1 be generated by independent and identically distributed (i.i.d) random variables (r.v.) X_i^j with probability distribution P_x such that $P[X_i^j = 0] = p[X_i^j = 1] = 0.5$ for all i and j . The function f generates i.i.d. r.v. $Z_i = f(x_i^1, x_i^2, \dots, x_i^n)$ with probability distribution P_z where $P[Z_i = 0] = P[Z_i = 1]$ and $P[Z_i = x_i^j] = q_i$. $P[Y_i = 0] = P_0$.

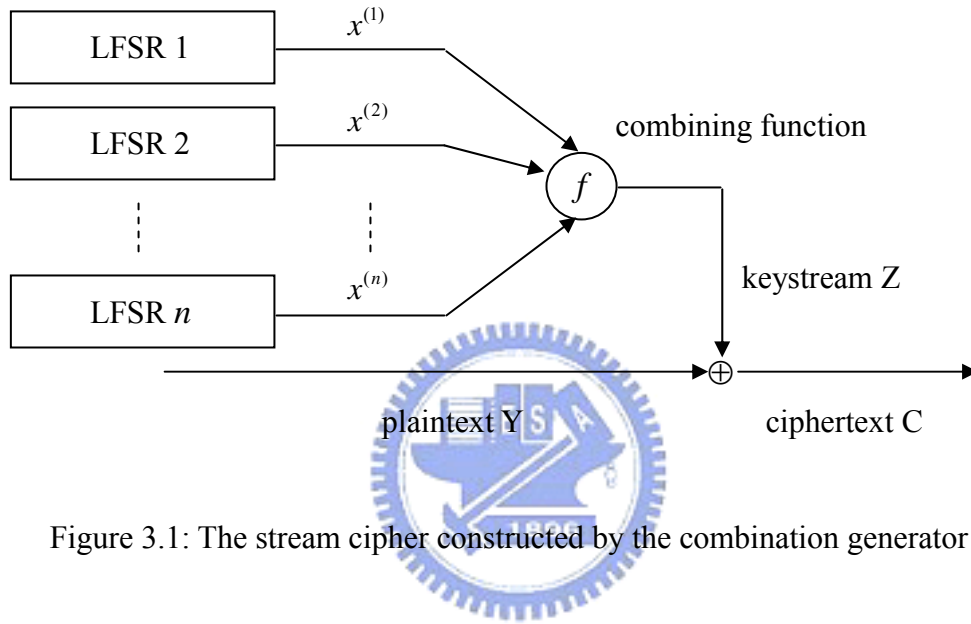


Figure 3.1: The stream cipher constructed by the combination generator

The r.v. α as a measure for the correlation between C_i and X_i^j is defined as

$$\alpha = \sum_{i=1}^N (1 - 2 \cdot (C_i \oplus X_i^j)) / N = 1 - 2 \cdot \sum_{i=1}^N (C_i \oplus X_i^j) / N \quad (j \in \{0, 1, \dots, n\}) \quad (25)$$

The probability $P[C_i \oplus X_i^j = 0] = p_e$ can be determined

$$\begin{aligned} p_e &= P(C_i = X_i^j) = P(C_i \oplus X_i^j = 0) \\ &= P(Y_i = 0) \cdot P(Z_i = X_i^j) + P(Y_i = 1) \cdot P(Z_i \neq X_i^j) \\ &= P_0 \cdot q_j + (1 - P_0) \cdot (1 - q_j) = 1 - (P_0 + q_j) + 2P_0 \cdot q_j \end{aligned} \quad (26)$$

The random variable $\beta = \sum_{i=1}^N (C_i \oplus X_i^j)$ is binomially distributed with mean value m_β and variance σ_β^2

$$m_\beta = E(\beta) = \sum_{i=1}^N E(C_i \oplus X_i^j) = N \cdot (1 - p_e)$$

$$\delta_\beta^2 = \text{Var}(\beta) = \sum_{i=1}^N \text{Var}(C_i \oplus X_i^j) = N \cdot p_e \cdot (1 - p_e)$$

The expected value and variance of α , m_β and σ_β^2 will be

$$m_\alpha = E(\alpha) = 1 - 2 \cdot E\left(\sum_{n=1}^N (C_n \oplus X_n^i)\right) / N = 2 \cdot p_e - 1 \quad (27)$$

$$\delta_\alpha^2 = \text{Var}(1 - 2 \cdot \beta / N) = 2^2 \cdot \delta_\beta^2 / N^2 = 4 p_e (1 - p_e) / N \quad (28)$$

For large N , the r.v. α can be assumed to be normally distributed with parameter m_β and σ_β^2 due to the central limit theorem.

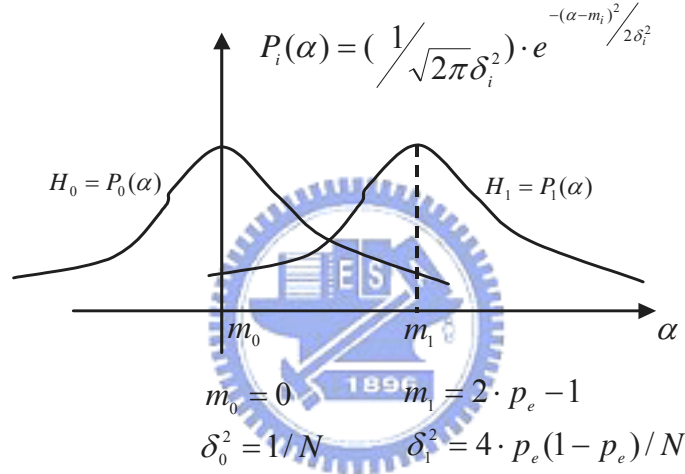


Figure 3.2: Probability density function for H_0 and H_1

Because of the independence of Z_i and X_i^0 and because of the statistics of X_i^0 we have for $j = 0$: $q_0 = 0.5$ and $p_e = 0.5$ and with (27) and (28) $m_\beta = 0$, $\sigma_\beta^2 = N$. The above situation will lead to the failure of the divide and conquer attack. In an attack an actual value α_0 for α is determined from N ciphertext digits and N digits generated by a LFSR of length l_i with an arbitrary initial state and an arbitrary of the L_i sets of feedback coefficients. There are two hypotheses to be considered.

H_1 : The $N > l_i$ digits of the LFSR of length l_i coincide with N digits generated by the LFSR i . This case corresponds to α being the correlation of C_i and X_i^j .

H_0 : The $N > l_i$ digits of the LFSR of length l_i do not coincide with N digits generated

by the LFSR i . Therefore C_i and X_i^j are independent.

Figure 3-2 shows the normally distributed probability density function for H_0 and H_1 .

The value T shall be the decision threshold for the two hypotheses H_0 and H_1 .

(1) for $\alpha_0 < T$, H_0 is accepted.

(2) for $\alpha_0 \geq T$, H_1 is accepted.

For $p_e = 0.5$ (i.e., $q_i = 0.5$ or/and $P_0 = 0.5$) the two probability density function are identical and therefore no decision can be made. The computational effort depends on the number of wrong decision, i.e., on the number of values α_0 exceeding the threshold T . Therefore, the probability P_f for a “false alarm” $P(\alpha \geq T | H_0)$ is of primary interest. To determine the decision threshold, however, the probability P_m for “missing the event” $P(\alpha < T | H_1)$ must also be taken into account

$$p_f = P(\alpha_0 \geq T | H_0) = \int_T^{\infty} P_{\alpha|H_0}(x) dx$$

$$p_m = P(\alpha_0 < T | H_1) = \int_{-\infty}^T P_{\alpha|H_1}(x) dx$$

With

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^{\infty} e^{-\frac{y^2}{2}} dy$$

$$p_f = Q\left(\frac{T}{\sqrt{N}}\right)$$

and

$$p_m = Q\left(\frac{N(2p_e - 1) - T}{2\sqrt{N}\sqrt{p_e(1 - p_e)}}\right)$$

Instead of the threshold T we use for convenience

$$\gamma_0 = \frac{N(2p_e - 1) - T}{2\sqrt{N}\sqrt{p_e(1 - p_e)}}$$

and $\frac{T}{\sqrt{N}} = \sqrt{N}(2p_e - 1) - 2\gamma_0\sqrt{p_e(1-p_e)}$

which give the final expressions for P_m and P_f

$$p_m = Q(|\gamma_0|) \tag{29}$$

$$p_f = Q(|\sqrt{N}(2p_e - 1) - 2\gamma_0\sqrt{p_e(1-p_e)}|) \tag{30}$$

The number of “false alarm” ($\alpha \geq T \mid H_0$) and consequently the number of tests necessary depend on the number N of cipher digits used. If we choose N_1 such that

$p_f = \frac{1}{L_j 2^{l_j}}$, we get

$$\frac{1}{L_j 2^{l_j}} = Q(|\sqrt{N}(2p_e - 1) - 2\gamma_0\sqrt{p_e(1-p_e)}|)$$

If we use $Q(x) < \frac{1}{2} e^{-\frac{x^2}{2}}$ ($x \geq 0$) to compute N_1 , we can get the upper bound of N_1 :

$$\begin{aligned} \frac{1}{L_j 2^{l_j}} &= Q(|\sqrt{N}(2p_e - 1) - 2\gamma_0\sqrt{p_e(1-p_e)}|) < \frac{1}{2} e^{-\frac{1}{2}(\sqrt{N_1}(2p_e - 1) - 2\gamma_0\sqrt{p_e(1-p_e)})^2} \\ \Rightarrow N_1 &< \left[\frac{\frac{1}{\sqrt{2}} \sqrt{\ln(L_j 2^{l_j})} + \gamma_0 \sqrt{p_e(1-p_e)}}{p_e - 1/2} \right]^2 \end{aligned} \tag{31}$$

This upper bound (31) can be used to roughly estimate the number of ciphertext digits to perform an attack on a system as given in Figure 3.1.

All steps of the attack are as follows: Let degree of the j -th LFSR be l_j .

Step 1: The probabilities q_i are determined from function f . The probability P_0 is known from the code (e.g. ASCII) and the language of the plaintext. Using (26) we calculate the probability p_e .

Step 2: For a chosen value P_m the parameter γ_0 of (29) is a constant and from (30) the probability for “false alarm” $P(\alpha \geq T \mid H_0)$ can be determined as a function of the N plaintext digits used in (25).

Step 3: To find the LFSR i part of the key, we choose one out of the L_i possible

primitive feedback polynomials and generate the corresponding maximal length $\{s_i\}$ of period $2^i - 1$.

Step 4: For each of $2^i - 1$ possible positions of $\{s_i\}$ and the N ciphertext digits, the correlation α is computed. For each event $\alpha \geq T$ it is assumed that the correct feedback polynomial and the correct position is used, hence the LFSR i part of the key is known. Because the event $\alpha \geq T \mid H_0$ occurs with probability P_f , our decision may be wrong. Therefore, additional tests with new ciphertext segments have to be performed at all positions with $(\alpha \geq T)$. If H_1 is rejected for all of the $2^i - 1$ positions, return to step 3 and choose another new primitive feedback polynomial to go on.

In the worst case, all of the $2^i - 1$ positions of all the possible L_i feedback polynomials have to be tested. The complexity of this attack is approximately $\sum_{j=1}^n L_j (2^{L_j-1})$. This attack is to use the correlation between inputs and outputs in the Boolean function. Therefore seeking the Boolean function without the correlation between inputs and outputs in it is the best method to prevent the divide and conquer attack. The property of correlation immune in the Boolean function is used to resist this attack. If f is the correlation-immunity of one order Boolean function, then this attack will not work except using two LFSR simultaneously to attack. Hence the higher correlation-immunity of Boolean function, the securer the stream cipher constructed by this function. The concept of correlation attacks of Siegenthaler on LFSR-based keystream generators was improved by the basic fast correlation attack of Meier and Staffelbach [18, 19]. Recently, e.g. [20-26], more advanced decoding techniques have been proposed to mount a fast correlation attack. Their common method is to find low weight parity check polynomials of LFSR and /or to apply an iterative decoding procedure to realize the attack. These fast correlation attacks can

also be applied with minor modifications to the filter generator [27-30].

3.2 The best affine approximation attack

This section will introduce another attack, which is called the best affine approximation attack, abbreviated BAA, and we will analyze the method of its attack and discuss how to prevent this kind of attack.

The concept of the best affine approximation was first proposed by Rueppel [31] in 1986. This kind of attack is different from the correlation attack. It does not find the initial state of LFSRs in the stream cipher, but to construct a new LFSR to approximate the original cryptosystem from known information. Such attack is to use the sequence of low complexity to approximate one of high complexity in nature and then it produces the similar keystream with the original cryptosystem. At last we can get the approximate plaintext from the ciphertext.

The BAA attack can attack the combination generator and the filter generator. We take the combination generator as Figure 2.8 to explain. BAA of the combining function $f(x)$ is denoted by $w \bullet x$, where $w = (w_1, w_2, \dots, w_n)$ and $x = (x_1, x_2, \dots, x_n)$. The relation between $\Pr[f(x) = w \bullet x] = F_{(f)}(w)$ is :

$$F_{(f)}(w) = \frac{1}{2^n} (\#\{x \mid f(x) = w \cdot x\} - \#\{x \mid f(x) \neq w \cdot x\})$$

$$\Rightarrow F_{(f)}(w) = \frac{1}{2^n} (\#\{x \mid f(x) = w \cdot x\} \cdot 2 - 2^n)$$

$$\Rightarrow F_{(f)}(w) = 2 \cdot \Pr[f(x) = w \cdot x] - 1$$

$$\Rightarrow \Pr[f(x) = w \cdot x] = \frac{1}{2} + \frac{F_{(f)}(w)}{2}$$

Theorem 3.1: Let $a = \max_w \{F_{(f)}(w)\}$

(1) $w \bullet x$ is the BAA of $f(x)$ and the probability that $f(x)$ and $w \bullet x$ are equal is:

$$\Pr[f(x) = w \cdot x] = \frac{1}{2} + \frac{1}{2}a$$

(2) $w \bullet x + 1$ is the BAA of $f(x)$ and the probability that $f(x)$ and $w \bullet x + 1$ are equal is:

$$\Pr[f(x) = w \cdot x + 1] = \frac{1}{2} + \frac{1}{2}a$$

The BAA attack is the known plaintext attack. It first computes $a = \max_w |F_{(f)}(w)|$

from $f(x)$. Let k of w_1, w_2, \dots, w_n be 1 ($w_{i_1} = w_{i_2} = \dots = w_{i_k} = 1$) and others are all

zero. BAA of $f(x)$ is $L(x) = x_{i_1} + x_{i_2} + \dots + x_{i_k} + c$ and if $F_{(f)}(w) \geq 0, c = 0$; else

$c = 1$. We know from the above equation that $z' = u^{(i_1)} + u^{(i_2)} + \dots + u^{(i_k)} + \{c \cdot 1\}_{j \geq 0}$

has the highest similarity with the keystream z in all linear combinations of LFSRs.

Because we know the degree of all LFSRs' feedback polynomials, from the theorem

of the linear complexity [32] we can easily infer the linear complexity of the

sequence z' is $\Lambda(z') \leq l_{i_1} + l_{i_2} + \dots + l_{i_k}$, where l_{i_j} is the degree of the i_j -th LFSR's

feedback polynomial. Furthermore, we can construct a new LFSR to make the

similarity of its output sequence with the keystream sequence of the original stream

cipher to be $\frac{1}{2} + \frac{1}{2}a$ and this LFSR can replace the combination generator in the

original stream cipher. We take an example to explain.

Example 3.1: The stream cipher constructed by the combination generator consists of five LFSRs and each degree of their feedback polynomial is $l_1 = 3, l_2 = 4, l_3 = 5, l_4 = 6, l_5 = 7$ respectively. The truth table of the combining function $f(x)$ is

$$T_f = [00\ 01\ 00\ 11\ 11\ 00\ 11\ 00\ 00\ 11\ 00\ 11\ 11\ 00\ 11\ 00]$$

When $w = (0, 1, 0, 1, 0)$, $f(x)$ calculates $a = 15/16$. Therefore the BAA function is x_2

+ x_4 and $\Lambda(z') \leq 10$. This BAA function generates the sequence z' whose similarity

with the original keystream z is $31/32$. So we can generate a LFSR whose degree of

the feedback polynomial are at most 10 to approximate the original sequence and these two sequences have the 96 percent similarity. \square

Because the BAA attack decides the similarity between the keystream sequence and the sequence which BAA of the combining function approximates according to a , seeking the Boolean function $f(x)$ possessing minimum a as the combining function is the best method to resist prevents the BAA attack. Because the definition of a in the combining function is $a = \max_w \{F_{(f)}(w)\}$, its relation with the nonlinearity of the Boolean function is as follows:

$$N_f = 2^{n-1} - \frac{1}{2} \cdot \max_{w \in Z_2^n} \{F_f(w)\} = 2^{n-1} - 2^{n-1} \cdot a$$

From the above equation we know the smaller a is, the bigger N_f is. Therefore, seeking the Boolean function possessing minimum a is equal to seeking one possessing the maximal nonlinearity and this one can be used as the combining function in the stream cipher to resist the BAA attack. The filter function also possesses the maximal nonlinearity to resist the BAA attack.

3.3 The algebraic attack

The algebraic attack was presented in 2003 [33, 34]. Before discussing the algebraic attack, we first introduce the XL algorithm. In [35] the XL algorithm was first presented to solve overdefined quadratic systems. Instead of solving a system of m multivariate quadratic equations with n variables of degree $d = 2$ as in [35], we consider also higher degree equations, i.e. study the general case $d \geq 2$ [36]. Let D be the parameter of the XL algorithm. Let $l_i(x_0, \dots, x_{n-1}) = 0$ be the initial m equations, $i = 1 \dots m$ with n variables $x_i \in GF(2)$. The XL algorithm consists of multiplying both sides of these equations by products of variables:

1. *Multiply*: Generate all the products $\prod_{j=1}^k x_{i_j} \bullet l_i$ with $k \leq D - d$, so that the total degree in the x_i of these equations is $\leq D$.
2. *Linearize*: Consider each monomial in the x_i of degree $\leq D$ as a new variable and perform Gaussian elimination on the equations obtained in 1. The ordering on the monomials must be such that all the terms containing one variable (say x_1) are eliminated last.
3. *Get a Simpler Equation*: Assume that step 2 yields at least one univariate equation in the power of x_1 . Solve this equation over the finite field.
4. *Final step*: It should not be necessary to repeat the whole process. Once the value of x_1 is known, we expect that all the other variables will be obtained from the same linear system.

The XL algorithm consists of multiplying the initial m equations l_i by all possible monomials of degree up to $D - d$, so that the total degree of resulting equations is D .

Let R be the number of equations generated in XL, and T be the number of all monomials. We have

$$R = m \bullet \left(\sum_{i=0}^{D-d} \binom{n}{i} \right) \approx m \bullet \binom{n}{D-d}$$

$$T = \sum_{i=0}^D \binom{n}{D-i} \approx \binom{n}{D}$$

The main problem in the XL algorithm is that in practice not all the equations generated are independent. Let *Free* be the exact number of equations that are linearly independent in XL. When $Free \geq T - D$, it is possible by Gaussian elimination, to obtain one equation in only one variable, and XL will work.

Otherwise, we need a bigger D , or an improved algorithm. [36] has several tables to show the relation between all parameters ($d, n, m, D, R, T, Free$) in XL.

The complexity of XL is mainly in the Gaussian reduction. The fastest practical

algorithm we are aware of is Strassen's algorithm that requires about $7 \cdot T^{\log_2 7}$ operations.

Then we introduce the higher order correlation attack [36] that can affect both a filter generator and a combination generator. We assume the connection function L is public and only the state is secret. We also assume that function f that computes the output bit from the state is public and does not depend on the secret key of the cipher. We take the filter generator as in Figure 2.7 as an example and f is the filter function. Let (k_0, \dots, k_{n-1}) be the initial state, then the output of the cipher (i.e. the keystream) is given by:

$$\begin{aligned} b_0 &= f(k_0, \dots, k_{n-1}) \\ b_1 &= f(L(k_0, \dots, k_{n-1})) \\ b_2 &= f(L^2(k_0, \dots, k_{n-1})) \\ &\dots \end{aligned}$$



The problem we consider is to recover (k_0, \dots, k_{n-1}) given some b_i . In this attack we assume that we have some m bits of the keystream at some known positions: $\{(t_1, b_{t_1}), \dots, (t_m, b_{t_m})\}$ and want to solve a system of multivariate equations that is overdetermined (much more equations than unknowns). This attack works in two cases:

- S1** When the Boolean function f has a low algebraic degree d .
- S2** When f can be approximated with good probability by a function g that has a low algebraic degree d .

More precisely, we assume that:

- $f(x_0, \dots, x_{n-1}) = g(x_0, \dots, x_{n-1})$ holds: 1. with probability $\geq 1 - \epsilon$
- 2. and with g of degree d .

In the first scenario S1, when f has just a low algebraic degree, it is known that

the system can be easily broken given $\binom{n}{d}$ keystream bits. So if f has a high algebraic degree, this stream cipher will be hard to break. Since in S2, we do not need for the function to have a low algebraic degree (S1), successful attacks can be mounted given much less keystream bits, and with much smaller complexities. If we choose m such that $(1 - \varepsilon)^m \geq 1/2$, we may assume that all these equations are true and we have to find a solution to our system of m multivariate equations of degree d with n variables. We take the Boolean function in Toyocrypt [37] as an example. The Boolean function is as follows:

$$f(x_0, \dots, x_{127}) = x_{127} + \sum_{i=0}^{62} x_i x_{\alpha_i} + x_{10} x_{23} x_{32} x_{42} + x_1 x_2 x_9 x_{12} x_{18} x_{20} x_{23} x_{25} x_{26} x_{28} x_{33} x_{38} x_{41} x_{42} x_{51} x_{53} x_{59} + \prod_{i=0}^{62} x_i \quad (32)$$

with $\{\alpha_0, \dots, \alpha_{62}\}$ being some permutation of the set of $\{63, \dots, 125\}$.

We put: $g(x_0, \dots, x_{127}) = \sum_{i=0}^{62} x_i x_{\alpha_i}$. Then $f(x) = g(x)$ holds with probability about $1 - 2^{-4}$. That is $d = 2$ and $\varepsilon = 2^{-4}$. And if we put: $g(x_0, \dots, x_{127}) = \sum_{i=0}^{62} x_i x_{\alpha_i} + x_{10} x_{23} x_{32} x_{42}$

Then $f(x) = g(x)$ holds with probability very close to $1 - 2^{-17}$. That is $d = 4$ and $\varepsilon = 2^{-17}$. We can choose m to $(1 - \varepsilon)^m \geq 1/2$ and apply XL to solve the initial x_0, \dots, x_{127} .

If $(1 - \varepsilon)^m < 1/2$, the attack still works; if we repeat it about $(1 - \varepsilon)^{-m}$ times, each time for a different subset of m keystream bits until it succeeds. This complexity is as

$$\text{follows: } WF = T^\omega (1 - \varepsilon)^{-m} \approx \left(\frac{n}{n/m^{1/d}} \right)^\omega (1 - \varepsilon)^{-m}$$

To summarize, $n = 128$, $m = 1.3 \cdot 2^{16}$, $d = 4$, $\varepsilon = 2^{-17}$, and $(1 - \varepsilon)^m = 0.52 \geq 1/2$. We

have $T = \left(\sum_{i=0}^D \binom{n}{i} \right)$, $R = m \left(\sum_{i=0}^{D-4} \binom{n}{i} \right)$. As $D = 9$, XL works. The complexity of

the attack is basically the complexity of solving a linear system $T \times T$. With Strassen's algorithm, we get

$$WF = 7 \cdot T^{\log_2 7} = 2^{128}$$

In conclusion, we can reduce the cryptanalysis of a stream cipher to solving a system of multivariate equations that is overdefined. In order to resist the higher order correlation attack, we must find the Boolean function which possesses the very high algebraic degree and is approximated with very low probability by a function that has a low algebraic degree.

The algebraic attack is to improve the higher order correlation attack to break a stream cipher. The algebraic attack lowers the degree of these multivariate equations by multiplying them by well-chosen multivariate polynomials.

At the time t , the current keystream bit gives an equation $f(x) = b_t$ with x being the current state. The main new idea consists of multiplying $f(x)$, that is usually of high degree, by a well chosen multivariate polynomial $g(x)$, such that fg is of substantially lower degree, denoted by d . Then for example if $b_t = 0$, we get an equation of low degree $f(x)g(x) = 0$. This in turn gives a multivariate equation of low degree d on the initial state bits k_i . If we get one such equation for each of sufficiently many keystream bits, we obtain a very overdefined system of multivariate equations that can be solved efficiently.

Except S1 and S2 in [36], the algebraic attack [33] presents two new scenarios as follows:

- S3** The multivariate polynomial f has some multiple fg of low degree d , with g being some non-zero multivariate polynomial.
- S4** It is also possible to imagine attacks in which f has some multiple fg , such that fg can be approximated by a function of low degree with some probability $(1 - \varepsilon)$

In scenarios S3 and S4, for each known keystream bit $b_t = f(x)$ at position t , we get:

$$f(x) \bullet g(x) = b_t \bullet g(x)$$

and, since the state at time t is $x = L^t(x_0, \dots, x_{n-1})$, it boils down to :

$$f(L^t(x_0, \dots, x_{n-1})) \bullet g(L^t(x_0, \dots, x_{n-1})) = b_t \bullet g(L^t(x_0, \dots, x_{n-1}))$$

This is the equation we are going to use in our attack. We get one multivariate equation for each keystream bit. This equation may be of very low degree, without f being of low degree, and without f having an approximation of low degree.

In the basic version of this attack S3, we also require that g is of low degree.

There are other possibilities. In the basic version of the attack S3, that may be called S3a, we use the equation written above and we require $fg \neq 0$ and fg is of low degree, and also we need g of low degree. There is another variant, in which we may admit that for all x such that $f(x)g(x) = 0$, and the equation can still be used when $b_t \neq 0$.

This is called the scenario S3b. Another variant, called S3c, allows to relax the degree condition on g : when $b_t = 0$, we can still use the equation, whatever is the degree of g , provided that $fg \neq 0$ and is of low degree. All the 3 sub-cases of the S3 attack scenario are summarized in the following Table 4.

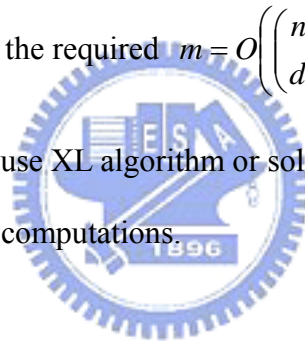
Attack scenario considered	Degree of			Use the equation	Only when	Number of equations for m keystream bits
	f	g	fg			
S1 and S2	low	$g = 1$	low	$f(x) = b_t$	always	m
S3a and S4a	high	low, $g \neq 0$	low, $fg \neq 0$	$f(x) \bullet g(x) = b_t \bullet g(x)$	always	m
S3b and S4b	high	low, $g \neq 0$	$fg = 0$	$g(x) = 0$	$b_t \neq 0$	$m/2$
S3c and S4c	high	high	low, $fg \neq 0$	$f(x) \bullet g(x) = 0$	$b_t = 0$	$m/2$

Table 4: Different methods to obtain low degree equations from keystream bits

In this attack, given m keystream bits, let R be the number of multivariate equations of degree d , and with n variables x_i . With one equation, and in scenario S3a, we have $R = m$, but we may also combine several scenarios and several different g for the same f , and get, for example, $R = 14 \bullet m$. We solve them as follows.

Linearization Method: There are about $T \approx \binom{n}{d}$ monomials of degree $\leq d$ in the n variables x_i (assuming $d \leq n/2$). We consider each of these monomials as a new variable V_j . Given $R \geq \binom{n}{d}$ equations, we get a system of $R \geq T$ linear equations with $T = \binom{n}{d}$ variables V_i that can be easily solved by Gaussian elimination on a linear system of size T .

XL Method: When as many as the required $m = O\left(\binom{n}{d}\right)$ keystream bits are not available, it is still possible to use XL algorithm or solve the system with less keystream bits, but with more computations.



Therefore the complexity of this algebraic attack is equal to the complexity of the higher order correlation attack, which is $7 \bullet T^{\log_2 7}$.

The method of this attack is by factoring multivariate polynomials. We consider the terms of high degree in $f(x)$ (regardless the lower degree terms) and look if they are divisible by a common low degree factor $g'(x)$. Then (for polynomials over $GF(2)$) we observe that $f(x) \bullet g(x)$ with $g(x) = g'(x) - 1$ is of low degree. Take (32) as an example. We observe that the combination of the parts of degree 4, 17 and 63, is divisible by a common factor $x_{23}x_{42}$. Let $f(x) = b_t$, and multiply both sides of it by $g(x) = (x_{23} - 1)$. Then we get $f(x)x_{23} - f(x) = b_t(x_{23} - 1)$. The monomials divisible by x_{23} in f will cancel out, and what remains is an equation of degree 3 true with probability 1. We repeat the same trick for x_{42} , i.e. we put $g(x) = (x_{42} - 1)$. From this,

we have a simple linearization attack following the scenario S3a. For each keystream bit, we obtain 2 equations of degree 3 in the x_i and thus 2 equations of degree 3 in the k_i . The linearization will work as soon as $R > T$. We have

$T \approx \binom{128}{3} = 2^{18.4}$ monomials and $R = 2m$ with $m = T/2 = 2^{17.4}$ keystream bits. This

attack is in $7 \cdot T^{\log_2 7} = 2^{55}$ CPU clocks, requiring 16 Gigabytes of memory and only about 20 kilobytes of keystream.

There are many interesting cases in which this attack will work as follows:

- 1 either f uses a small subset of state bits,
- 2 or is very sparse,
- 3 or can be factored with a low degree factor,
- 4 or can be approximated by one of the above,
- 5 or its part of high degree is one of the above

We conclude that, in a stream cipher with linear feedback, the filter function should use many state bits and should not be too sparse, so it has also many terms of high degree. Moreover the part of high degree should not have a low degree factor, and should itself also use many state bits. Then no approximation of the part of high degree should have a low degree factor, or should use a small number of state bits. The algebraic immunity in the definition 2.18 of section 2-3 is used to resist the algebraic attack. In recent years, there have been many attacks which improve the algebraic attack [34, 38].

3.4 Other attacks on filter generators

In [51] the author introduces some other attacks on the filter generator as shown in Figure 3.3. These attacks are specially targeted on the filter generator. We

just generally introduce these attacks and will not discuss the details. Note that k is the degree of the connection polynomial and n is the number of inputs in f .

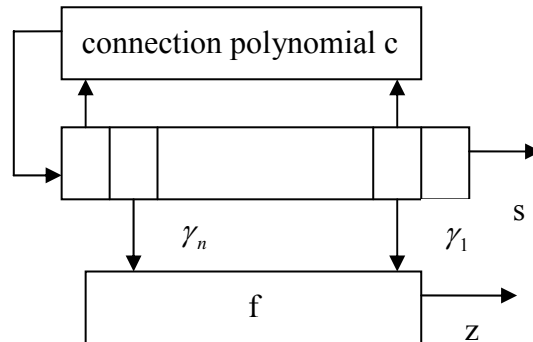


Figure 3.3: The filter generator

In [39-41] the special and the general inversion attacks were published and analyzed. The first one only works for filter functions f which are linear-separable in the first one or last variable, i.e. $f(x_1, \dots, x_n) = x_1 + g(x_2, \dots, x_n)$ or $f(x_1, \dots, x_n) = g((x_1, \dots, x_{n-1}) + x_n$, where $g: \text{GF}(2)^{n-1} \rightarrow \text{GF}(2)$ is an arbitrary Boolean function. The general inversion attack is applicable to any filter function. Both attacks have time complexity of $O(2^{M-1})$ on average, where M is $\gamma_n - \gamma_1$ and are successful for highly nonlinear filter functions and less known keystream N . The inversion attack was improved for certain filter generator configurations in [42] to $O(2^{k-r-1})$, where r is the largest gap between LFSR cells, which have taps to the filter function or the connection polynomial c . The filter generator can be made resistant against the inversion attack if one chooses $\gamma_1 = 0$, $\gamma_n = k - 1$ and $\text{gcd}(\gamma_1, \dots, \gamma_n) = 1$. In [43] ideas from the inversion attack and the conditional correlation attack are used to form a trellis based decoding procedure. Like the inversion attack, it has a time complexity of $O(2^{M-1})$ and is conceptually the same as the basic generalized inversion attack from [40, 41].

The set of exponents of the connection polynomial with non-zero coefficients is

called the *LFSR tapset*, denoted $T = \{0, 4, 15, 17\}$. Note that the LFSR tapset contains the indices for inputs to the linear recurrence, combined with the register length. The *filter generator tapset* $\Gamma = \{0, 1, 6, 13, 16\}$, contains the inputs to the filter generator. The values in a tapset are called *taps*. Given a tapset S , we define the positive difference set $\Delta(S)$ to be the set of positive differences between the taps in that set. Consequently, $\Delta(T) = \{2, 4, 11, 13, 15, 17\}$, while $\Delta(\Gamma) = \{1, 3, 5, 6, 7, 10, 12, 13, 15, 16\}$. Each of the tapsets is *full positive difference set* (FPDS), meaning that no positive difference is repeated. Full positive difference sets are highly recommended for LFSR-based ciphers [39, 44].

In [45, 46] the decimation attack is proposed for LFSR based keystream generators. The idea is to consider a decimated sequence $z[d]$, with $z[d] = z_0, z_d, z_{2d}, \dots$ of the observed keystream sequence $z = z_0, z_1, z_2, \dots$. For the generator filter the decimation attack is applicable to a d with $1 \leq d \leq g$, $d \mid g$ and $g = \gcd(\gamma_1, \dots, \gamma_n)$ [39]. For such a d , the decimated keystream sequence $z[d]$ can now be written as

$$z_{dt} = f(s_{dt+\gamma_1}, \dots, s_{dt+\gamma_n}) = f(s_{t+\gamma_1/d}^2, \dots, s_{t+\gamma_n/d}^2)$$

for all $t \geq 0$. Thus, the decimated sequence $z[d]$ can be generated from the decimated LFSR sequence $s[d]$. If the decimated sequence $s[d]$ can be generated by a smaller LFSR with length $k' < k$, then all known attacks against the filter generator can be applied to this smaller filter generator. Properties of decimated sequences have been developed in [31]. If k is chosen as prime or $1 \leq k \leq 89$, then it always holds that $k' = k$ and the decimation attack provides no further advantages.

In [47] $N = 2k$ keystream symbols are used to build an equation system with $2k$ nonlinear equations of the form $z_t = f(s_{t+\gamma_1}, \dots, s_{t+\gamma_n})$ for $0 \leq t \leq 2^k - 1$ and $k + \gamma_n$ linear equations for the variables $s_k, s_{k+1}, \dots, s_{2k+\gamma_n-1}$ from the linear recurrence relation of the LFSR. For any nonlinear equation the solution set is computed, i.e. the set of all tuples fulfil the nonlinear equation. Then the solution sets of two

nonlinear equations with overlapping variables are iteratively merged and common values are removed from the merged set and substituted into the other equations. This process is called local reduction technique and is iterated until k independent variables from $\{s_0, s_1, \dots, s_{2k+\gamma n-1}\}$ have a solution or no further merging and substituting is possible. In the latter case, a tree-based search is done over the unsolved variables. The attack is only feasible for small values of n and there must be enough overlapping in the solution sets of the nonlinear equations.

In [48] tradeoff attacks (Time/Memory/Data tradeoffs) are developed and analyzed for synchronous stream cipher systems. Two main variants of a tradeoff attack are discovered, which differ in the generation of special states: Rivest and BSW sampling. Special states generate output prefixes of a keystream generator with a predefined bit pattern of l bit length. In the case of BSW sampling the special states of the keystream generator can be enumerated in an efficient way, i.e. in polynomial time. Both variants have a tradeoff relationship given by

$$TS^2N^2 = Z^2,$$

where T is time complexity in the realtime phase of the attack (i.e. one time unit equals the generation of $O(\log_2(Z))$ bit keystream), S represents the storage requirement (typically access on a hard disk), N is the amount of keystream, and Z is the size of the state space of the stream cipher, i.e. $Z = 2^k$ in the case of the filter generator. The time for preprocessing is $P = Z/N$ and the number of disk operations in the realtime phase is then given by $T_{\text{disk}} = (T)^{1/2}$ in the case of Rivest sampling and $T_{\text{disk}} = (T)^{1/2}2^{-1}$ for BSW sampling. In the case of Rivest sampling $D^2 \leq T \leq N$ is allowed and $(2^{-1}D)2 \leq T \leq N$ for BSW sampling. Such if a keystream generator allows efficient BSW sampling for an appropriate $l > 0$ the number of disk operations and the lower bound on T can be further reduced. Typical values could be $P = Z^{2/3}$, $T = Z^{2/3}$, $S = Z^{1/3}$, $N = Z^{1/3}$.

[49, 50] presented an $k^{O(1)}2^{(1-\alpha)/(1+\alpha)k}$ time bounded attack, the FBDD-attack, against LFSR-based generators, which computes the secret initial state $x \in \{0, 1\}^k$ from bn consecutive keystream bits, where α denotes the rate of information, which B reveals about the internal bitstream, and b denotes some small constant. The algorithm uses Free Binary Decision Diagrams (FBDDs), a data structure for minimizing and manipulating Boolean function. Let k be the secret key length and n be n LFSR. This attack can be applied to the combination generator. This attack computes the secret initial state x from the first k bits of $f(x_1, \dots, x_n)$ in the combination generator in time $k^{O(1)}2^{\frac{n-1}{n+1}k}$.



Chapter 4

Design of Stream Ciphers on Sensors

This Chapter will discuss how to design the secure stream cipher on Wireless Network Sensors. On Wireless Sensor Networks there are several severe challenges – these sensors have limited processing power, storage, bandwidth, and energy. So we must choose a fast and low-storage cryptosystem. The stream cipher is very fast and can lower power consumption. Since the stream cipher encrypts each character under a time varying function of the key, it prevents deletion, insertion or replay of ciphertext, as well as ciphertext searching. One may say that a stream cipher is inherently more secure than a block cipher because of the additional dimension offered by the use of memory. This advantage is important on Wireless Sensor Networks because we will not need extra storage or computing for resisting replaying attack. Saving storage and power consumption is very important for sensors so that they can live longer. Then we will design a stream cipher to be suitable for Wireless Network Sensors - that has small enough storage to be put on sensors. Of course, this stream cipher must be secure. We will design the stream cipher to resist all the attacks introduced in Chapter 3. A stream cipher is to use the generator to produce the pseudo-random keystream. The generator has the filter generator and combination generator. We will discuss which generator is more suitable for Wireless Network Sensors. There are two main components to construct the generator: one is LFSR and the other is a Boolean function.

First we talk about Boolean functions. They must satisfy some conditions to resist the attacks introduced in Chapter 3. They must be of high resilient, nonlinearity, algebraic degree, algebraic immunity and balancedness. We will use

the method of [52] to construct the Boolean function we want. [52] defines linear and quasilinear variables as follows:

Definition 4.1: If a variable x_i is linear for a function f we can represent f in the form: $f(x_1, \dots, x_{i-1}, x_i, x_{i+1}, \dots, x_n) = g(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) \oplus x_i$.

Other equivalent definition of a linear variable is that a variable x_i is linear for a function f if $f(\delta_1) \neq f(\delta_2)$ for any two vectors δ_1 and δ_2 that differ only in i -th component.

Definition 4.2: We say that a Boolean function $f = f(x_1, \dots, x_n)$ depends on a pair of its variables (x_i, x_j) quasilinearly if $f(\delta_1) \neq f(\delta_2)$ for any two vectors δ_1 and δ_2 of length n that differ only in i -th and j -th components. A pair (x_i, x_j) in this case is called a pair of quasilinear variables in f .

Theorem 4.1: Let n be a positive integer. Let $f_1(x_1, \dots, x_n)$ and $f_2(y_1, \dots, y_n)$ be m -resilient Boolean functions on V^n such that $N_{f_1} \geq N_0$, $N_{f_2} \geq N_0$. Moreover, there exist two variables x_i and x_j such that f_1 depends on the variables x_i and x_j linearly, and f_2 depends on a pair of the variables (x_i, x_j) quasilinearly. Then the function

$$f_1'(x_1, \dots, x_n, x_{n+1}) = (x_{n+1})f_1(x_1, \dots, x_n) \oplus x_{n+1}f_2(x_1, \dots, x_n) \quad (33)$$

is an m -resilient Boolean function on V^{n+1} with nonlinearity $N_{f_1'} \geq 2^{n-1} + N_0$, and the function

$$f_2'(x_1, \dots, x_n, x_{n+1}, x_{n+2}) = (x_{n+1} \oplus x_{n+2} \oplus 1)f_1(x_1, \dots, x_n) \oplus (x_{n+1} \oplus x_{n+2})f_2(x_1, \dots, x_n) \oplus x_{n+1} \quad (34)$$

is an $(m+1)$ -resilient Boolean function on V^{n+2} with nonlinearity $N_{f_2'} \geq 2^n + 2N_0$.

Moreover, f_2' depends on a pair of the variables (x_{n+1}, x_{n+2}) quasilinearly.

proof: see [52]

Theorem 4.2: $nl_{\max}(n, m) = 2^{n-1} - 2^{m+1}$ for $(2n-7)/3 \leq m \leq n - 2$, where $nl_{\max}(n, m)$ denotes the maximal possible nonlinearity of m -resilient Boolean function on V^n .

proof: see[52]

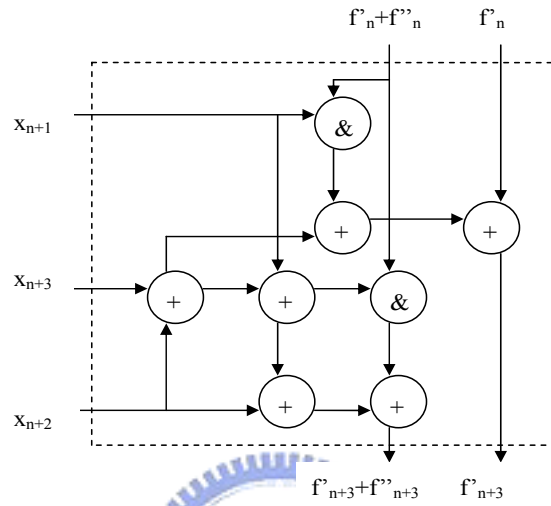


Figure 4.1: Scheme of block B

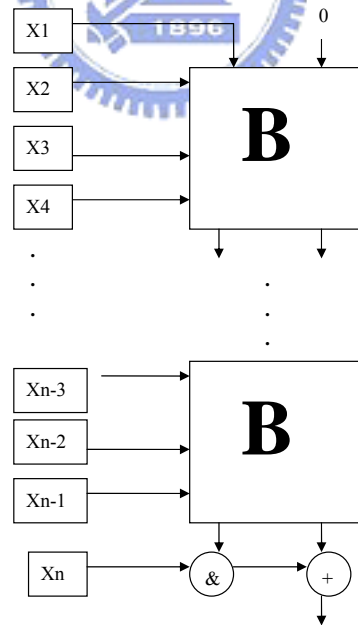


Figure 4.2: f_n

$$f'_{n+3} = (x_{n+1} \oplus 1)f'_n \oplus x_{n+1}f''_n \oplus x_{n+2} \oplus x_{n+3} \quad (35)$$

$$f''_{n+3} = (x_{n+2} \oplus x_{n+3} \oplus 1)f'_n \oplus (x_{n+2} \oplus x_{n+3})f''_n \oplus x_{n+1} \oplus x_{n+2} \quad (36)$$

By theorem 4.1 if f'_n and f''_n are m -resilient Boolean functions on V^n with maximal possible nonlinearity $(2^{n-1} - 2^{m+1})$, f'_n depends on its last two variables linearly and f''_n depends on a pair of its variables quasilinearly; then f'_{n+3} and f''_{n+3} are $(m+2)$ -resilient Boolean functions on V^{n+3} with maximal possible nonlinearity $(2^{n+2} - 2^{m+3})$, f'_{n+3} depending on its last two variables linearly and f''_{n+3} depending on a pair of its last variables quasilinearly.

It is a little more convenient to rewrite the relations (35), (36) in the form

$$f'_{n+3} = x_{n+1}(f'_n \oplus f''_n) \oplus f'_n \oplus x_{n+2} \oplus x_{n+3} \quad (37)$$

$$f'_{n+3} \oplus f''_{n+3} = (x_{n+1} \oplus x_{n+2} \oplus x_{n+3})(f'_n \oplus f''_n) \oplus x_{n+1} \oplus x_{n+3} \quad (38)$$

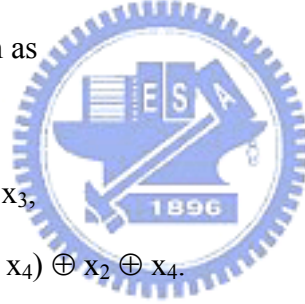
The relations (37), (38) allow to realize f'_{n+3} and $f'_{n+3} \oplus f''_{n+3}$ as two functions of five values $f'_n, f'_n \oplus f''_n, x_{n+1}, x_{n+2}, x_{n+3}$ by means of the block B (see Figure 4.1).

Initial functions can be chosen as

$$f'_4 = x_1x_2 \oplus x_3 \oplus x_4,$$

$$f''_4 = x_2 \oplus x_1(x_3 \oplus x_4) \oplus x_3,$$

$$f'_4 \oplus f''_4 = x_1(x_2 \oplus x_3 \oplus x_4) \oplus x_2 \oplus x_4.$$



Comparison with (37), (38) shows that we can take $f'_1 = 0, f''_1 = x_1$. Finally, we put

$$f'_n = x_n (f'_{n-1} \oplus f''_{n-1}) \oplus f'_{n-1}, \quad n \equiv 2 \pmod{3} \quad (39)$$

f'_n can be represented as Figure 4.2. From theorem 4.2 the function f'_n is

$(2n-7)/3$ -resilient function on $V^n, n \equiv 2 \pmod{3}$, with the nonlinearity $2^{n-1} - 2^{(2n-4)/3}$

and an algebraic degree of each variable in f'_n is $(n+4)/3$. The scheme of the function

f'_n contains $2n - 4$ XOR and $(2n - 1)/3$ AND. It is linear on n . The number of XOR

and AND in other functions constructed by usual methods, in general, is exponential

on n . Hence, our construction has a big advantage on sensors. It uses less storage

and operations and is faster. The Boolean function constructed by this method is $(n,$

$(2n-7)/3, (n+4)/3, 2^{n-1} - 2^{(2n-4)/3}$), that is $(n, \text{resilient, algebraic degree, nonlinearity})$.

If we choose n to be 11, it is $(11, 5, 5, 960)$ and from [13] we know the algebraic

immunity of this Boolean function is 4. We expect these values are enough to be a secure stream cipher.

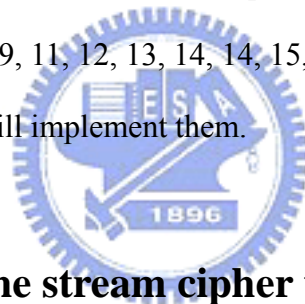
Then we talk about LFSR. On sensors, the key length is not too big, so we choose it to be 128. In the filter generator, we only need to find one primitive polynomial of degree 128. Its period is $2^{128} - 1$, and we believe it is long enough. But if we want to resist the inversion attack and the conditional correlation attack, the LFSR tapset must be FPDS. Therefore we choose

$$c = x^{128} + x^{99} + x^{59} + x^{31} + x^9 + x^7 + 1$$

$T = \{7, 9, 31, 59, 99, 128\}$ is FPDS. And the filter generator tapset Γ also need to be FPDS. Therefore, we choose $\Gamma = \{0, 1, 3, 7, 12, 20, 30, 44, 66, 82, 127\}$, and it is FPDS.

In the combination generator, we must choose 11 primitive polynomials and sum of their degree is 128, e.g. $\{6, 8, 9, 11, 12, 13, 14, 14, 15, 16, 17\}$.

In the next section, we will implement them.



4.1 Implementing the stream cipher with software

In this section, we will discuss how to implement the filter generator and the combination generator and decide which one is better for Wireless Sensor Networks.

The hardware specification of the sensor we use is as follows:

CPU	8051, 8-bit, 12MHz
Storage	512 bytes RAM
	16k bytes ROM
	64k flash RAM
OS	MicroC OSII
compiler	Keil C

Table 5: Characteristics of prototype sensors

We will implement the stream cipher on this hardware structure. We first write c program and compile it with Keil C and load the hex file Keil C produces into ROM

of sensors. Because the memory of sensors is small, we expect the code size of our stream cipher is smaller.

We first choose the filter generator to implement. Let n be 11 and the Boolean function is (39) in section 4.1. It is as follows:

$$\begin{aligned}
 f = & x11\{(x8+x9+x10) [(x5+x6+x7)(x1(x2+ x3+x4)+x2+x4)+x5+x7]+x8+x10\} \\
 & + x8 [(x5+x6+x7) (x1(x2+x3+x4)+x2+x4)+x5+x7] +x5(x1(x2+x3+x4) + \\
 & x2+x4) +x1x2+x3+x4+x6+x7+x9+x10 \tag{40}
 \end{aligned}$$

The connection polynomial of LFSR is $x^{128} + x^{99} + x^{59} + x^{31} + x^9 + x^7 + 1$. And the filter generator tapset is $\Gamma = \{0 1 3 7 12 20 30 44 66 82 127\}$. Let this stream cipher be *StreamCipher1* as in Figure 3.3.

Then we use “Pointer and circular buffer” to *StreamCipher1* to become *StreamCipher2*. “Pointer and circular buffer” [53] is as follows:

Pointer and circular buffer: is based on the idea of having a pointer pointing at the beginning of the LFSR in memory. When we clock the LFSR once we do not shift all the values one step in memory, but rather, we only move the pointer one position. This gives a compact code description of the LFSR sequence generation, and is faster than *StreamCipher1*.

Next we implement the combination generator. The Boolean function is the same as *StreamCipher1*. And we need 11 connection polynomials of LFSR as follows:

$$X^6 + X + 1$$

$$X^8 + X^5 + X^4 + X^3 + 1$$

$$X^9 + X^4 + 1$$

$$X^{10} + X^3 + 1$$

$$X^{11} + X^2 + 1$$

$$X^{12} + X^7 + X^4 + X^3 + 1$$

$$X^{13} + X^4 + X^3 + X^1 + 1$$

$$X^{14} + X^{12} + X^{11} + X + 1$$

$$X^{14} + X^5 + X^3 + X + 1$$

$$X^{15} + X + 1$$

$$X^{16} + X^5 + X^3 + X^2 + 1$$

We use these LFSRs and the Boolean function f to construct the **StreamCipher3** as Figure 2.8.

	StreamCipher1	StreamCipher2	StreamCipher3
Code Size	3.56k bytes	2.56k bytes	5.77k bytes
time	0.050614 s	0.050409 s	0.167840 s

Table 6: Code size of three stream cipher

Note that because we use int type to store LFSR, we choose the length of LFSRs to be less than 16. If we use the length of LFSRs which is larger than 16, we must use long type to store it, which will cost more code size and more time. Therefore, we use one int type to store one LFSR.

In Table 6 it is obvious that the filter generator is smaller than the combination generator. And the filter generator using “Pointer and circular buffer” is the smallest. Therefore, we choose the filter generator as our generator in the stream cipher. At last we implement the filter generator with 8051 assembly code and optimize it to be the smallest by using reuse and loop and so on. It will produce code size of 799 bytes. Key setup and running 128-bit keystream totally approximately cost 0.031426 seconds. The data rate is $128 / 0.031426 = 4073$ bits/s. We only XOR the plaintext with the keystream to complete encrypting.

4.2 Implementing the stream cipher with hardware

We devise these sensors to last as long as possible on Wireless Sensor Networks. We want to lower power consumption when sensors encrypt with the stream cipher. In the same CPU clock rate the algorithm of the faster encrypting consumes the lower power. Therefore, we want to make our encryption algorithm faster to lower power consumption. Of course, we also want our algorithm to use less memory. In order to save power consumption and memory we may change the hardware specification to better fit our stream cipher.

First we write the stream cipher algorithm with 8051 assembly code. We use the filter generator as our stream cipher generator. This filter generator consists of one LFSR of length 128 and one 11-variable Boolean function. LFSR of length 128 needs 16 addresses to be stored (one address is 8bits), that is from KEY0 to KEYF. The Boolean function needs 11 inputs from LFSR and one input is one bit. Therefore we often extract one bit from some address and it needs to do 11 times. Doing one time needs many operations as follows.

```
MOV    A, 30H
ANL    A, #08H
RR     A
RR     A
RR     A
MOV    R2, A
```

Doing these is only to move fourth bit of the value in address 30H to register 2. The connection polynomial of LFSR also needs to do these operations to compute the next state. So doing these operations in StreamCipher2 needs totally 17 times. If we can increase one new instruction to replace these operations, the code size will

reduce much and the speed of encryption will be faster.

After observing the stream cipher program with assembly code, we find that increasing this instruction, MOV Rn, ADDRESS.m, is a good idea. This instruction means to move m-th bit of the value in address ADDRESS to the register n; that is, this instruction can replace the above all instructions.

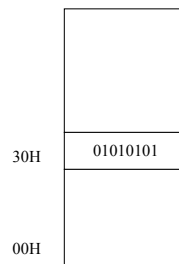


Figure 4.3: Memory

For example MOV R1, 30H.0 => R1 = 1, MOV R2, 30H.4 => R2 = 0 in Figure 4.3.

By using this instruction we will reduce code size and increase the speed of the stream cipher. How do we increase this new instruction in 8051? We first find out the source code of 8051 and modify it to increase this instruction. The source code of 8051 is VHDL or may be Verilog. But we only simulate it and do not implement it in reality. So we find a simulation of 8051 written by C++ and modify it to increase this instruction. How do we modify the simulation of 8051? If we can find out opcode which is not used in 8051, then we use this opcode as the opcode of our new instruction. The easier method is to modify the instruction which we do not use in the stream cipher algorithm to become our new instruction. In this case, we modify MOV DIRECT, DIRECT as Figure 4.4 below.

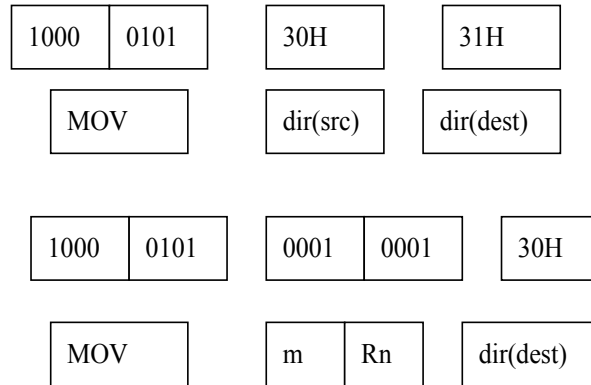


Figure 4.4: New instruction

In this Figure, the original instruction MOV dir, dir, is to move the value in address 30H to the value in address 31H. The modified instruction, MOV m, Rn, dir, is to move m-th bit of the value in address dir to register n, Rn. In Figure 4.4, MOV 11H, 30H is to move second bit (begin from 0) of the value in address 30H to R1. We take Figure 4.3 as an example, that is, R1 is equal to 0. By using this instruction we can largely decrease the code size of the stream cipher and increase the speed of the stream cipher. Table 9 compares the stream cipher not using the modified instruction and one using the modified instruction with regard to the code size and execution time. This modification improves by $799 - 578 = 221$ (bytes) and $0.031426 - 0.024642 = 0.006784$ (s). The improved rate of code size is $221/799 = 27.7\%$ and the improved rate of execution time is $0.016784/0.031426 = 21.6\%$.

	original	modified
code size (bytes)	799	578
execution time (s)	0.031426	0.024642

Table 7: Code size and execution time of the stream cipher

4.3 Analyzing security and conclusions

In the previous section, we obviously know the code size of the filter generator is smaller than one of the combination generator. On sensors memory is very critical. Because the filter generator is a special case of the combination generator and they share the same Boolean function f in (40), that is, they have the same nonlinearity, resilient, algebraic immunity, so they have the same power to resist some attacks, such as the BAA attack, the correlation attack, and the algebraic attack. Therefore we will choose the filter generator as our cryptosystem on sensors.

The structure of StreamCipher2 is shown in Figure 3.3. Because the connection polynomial c is a primitive polynomial so the period of the sequence s and z are $2^k - 1$ if f is balanced [55]. In StreamCipher2 the period of the keystream is $2^{128} - 1$. We believe it is long enough.

In Chapter 2 we hope the keystream generator produces possesses the randomness. While it is impossible to give a mathematical proof that a generator is indeed a random bit generator, the tests described below help detect certain kinds of weakness the generator may have. This is accomplished by taking a sample output sequence of the generator and subjecting it to various statistical tests. Each statistical test determines whether the sequence possesses a certain attribute that a truly random sequence would be likely to exhibit; the conclusion of each test is not definite, but rather probabilistic. If a sequence passes all five tests, there is no guarantee that it is indeed produced by a random bit generator [54].

(i) Frequency test (monobit test)

The purpose of this test is to determine whether the number of 0's and 1's in a sequence s are approximately the same, as would be expected for a random sequence. Let n_0, n_1 denote the number of 0's and 1's in s , respectively. The

statistic used is

$$X_1 = \frac{(n_0 - n_1)^2}{n}$$

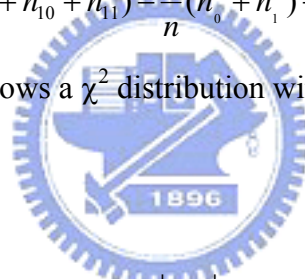
which approximately follows a χ^2 distribution with 1 degree of freedom if $n \geq 10$.

(ii) Serial test (two-bit test)

The purpose of this test is to determine whether the number of occurrences of 00, 01, 10, and 11 as subsequences of s are approximately the same, as would be expected for a random sequence. Let n_0, n_1 denote the number of 0's and 1's in s , respectively. Note that $n_{00} + n_{01} + n_{10} + n_{11} = (n - 1)$ since the subsequence is allowed to overlap. The statistic used is

$$X_2 = \frac{4}{n-1}(n_{00}^2 + n_{01}^2 + n_{10}^2 + n_{11}^2) - \frac{2}{n}(n_0^2 + n_1^2) + 1$$

which approximately follows a χ^2 distribution with 2 degree of freedom if $n \geq 21$.



(iii) Poker test

Let m be a positive integer such that $\lfloor \frac{n}{m} \rfloor \geq 5 \cdot (2^m)$, and $k = \lfloor \frac{n}{m} \rfloor$. Divide the sequence s into k non-overlapping parts each of length m , and let n_i be the number of occurrences of the i^{th} type of sequence of length m , $1 \leq i \leq 2^m$. The poker test determines whether the sequences of length m each appear approximately the same number of times in s , as would be expected for a random sequence. The statistic used is

$$X_3 = \frac{2^m}{k} \left(\sum_{i=1}^{2^m} n_i^2 \right) - k$$

which approximately follows a χ^2 distribution with $2^m - 1$ degree of freedom.

Note that the poker test is a generalization of the frequency test: setting $m = 1$ in the poker test yields the frequency test.

(iv) **Runs test**

The purpose of the run test is to determine whether the number of runs of various lengths in the sequence s is as expected for a random sequence. The expected number of gaps (or blocks) of length i in a random sequence of length n is $e_i = (n - i + 3)/2^{i+2}$, where a run of 0's is called a gap, while a run of 1's is called a block. Let k be equal to the largest integer i for which $e_i \geq 5$. Let B_i, G_i be the number of blocks and gaps, respectively, of length i in s for each $i, 1 \leq i \leq k$. The statistic used is

$$X = \sum_{i=1}^k \frac{(B_i - e_i)^2}{e_i} + \sum_{i=1}^k \frac{(G_i - e_i)^2}{e_i}$$

which approximately follows a χ^2 distribution with $2k - 2$ degrees of freedom.

(v) **Autocorrelation test**

The purpose of this test is to check for correlations between the sequence s and (non-cyclic) shifted version of it. Let d be a fixed integer, $1 \leq d \leq \lfloor n/2 \rfloor$. the

number of bits in s not equal to their d -shifts is $A(d) = \sum_{i=0}^{n-d-1} s_i \oplus s_{i+d}$. The

statistic used is

$$X = 2 \cdot (A(d) - \frac{n-d}{2}) / \sqrt{n-d}$$

which approximately follows an $N(0, 1)$ distribution if $n - d \geq 10$. Since small values of $A(d)$ are as unexpected as large values of $A(d)$, a two-sided test should be used.

Let a significance level of $\alpha = 0.5$ and the length of keystream is 10000 bits. Table 7 shows the results of StreamCipher2.

test	degree of freedom	passing range	other parameter	results	
frequency test	1	-3.84 ~ 3.83	no	pass	X1=1
serial test	2	-5.99 ~ 5.99	no	pass	X2=1
poker test	7	-14.067 ~ 14.067	m = 3	pass	X3=4
runs test	14	-23.685 ~ 23.685	k = 8	pass	X4=11.4
autocorrelation	no	-1.96 ~ 1.96	d = 500	pass	X5=1

Table 8: Statistic test table of StreamCipher2

The keystream StreamCipher2 produces is very highly probable to be random.

Then we hope StreamCipher2 can resist all kinds of attacks. The Boolean function f in StreamCipher2 is (11, 5, 5, 960) and its AI is 4. We believe 5-resilient is big enough to resist all correlation attacks. Nonlinearity is equal to 960 and in the BAA attack $a = 0.0625$ and the sequence the BAA attack generates is similar with the original keystream with probability of 0.53125. This value is low enough to resist the BAA attack. The generator filter tapset and the LFSR tapset are FPDS to resist the inversion attack and the conditional correlation attack.

attack	algebraic	BDD	Investion	tradeoff
complexity	$O(2^{65})$	$O(2^{114})$	$O(2^{82})$	$O(2^{85})$

Table 9: Complexity of attacks

Table 8 shows the complexity of other attacks. Let CPU clock rate be 4G, and it computes at most 2^{48} instructions in one day. Therefore if all complexity is larger than 2^{64} , we say the stream cipher is secure. So StreamCipher2 is secure.

Compared with RC5 in [56] the filter generator uses less code size and is faster. RC5 was used as the cryptosystem on Wireless Sensor Network in [56]. The faster the operations of encrypting are in the same clock rate, the less power consumption

is. This is also very important on sensors and this makes sensors survive longer. At last, we compare the filter generators, RC5 and A5. RC 5 and A5 are implemented by 8051 assembly code. The filter generator is implemented by our modified 8051 assembly code. The result is as follows.

	filter generator	RC5	A5
code size (bytes)	578	1789	1071
data rate (bits/s)	5194	600	3318

Table 10: Comparison among the filter generator, RC5, and A5

Obviously, our filter generator is faster than RC5 and A5 and uses less code size.



Chapter 5

Conclusion and Future Research

On Wireless Sensor Networks we aim to develop a secure, lower power-consumption, and lower code-size cryptosystem. The stream cipher meets these properties. The stream cipher is divided into the combination generator and the filter generator. The filter generator needs less code size than the combination generator. The filter generator is also faster than the combination generator. Therefore, we choose the filter generator as our cryptosystem on Wireless Sensor Networks. To resist all kinds of attacks we must find good properties of the LFSR and the Boolean function. The LFSR must be primitive and have large period. The Boolean function must have high correlation immunity, nonlinearity, algebraic degree and algebraic immunity. The LFSR tapset and the filter generator tapset must be FPDSs. If the Boolean function can be computed fast, it is the best. We use one LFSR of length 128 and one (11, 5, 5, 960) Boolean function of algebraic immunity 4 to compose the filter generator. This generator can resist most attacks. It is the secure cryptosystem we apply on Wireless Sensor Networks. The code size of this stream cipher is 799 bytes and the time when it produces 128-bit keystream is 0.031426 seconds in 12MHz 8051 CPU. If we increase a new instruction to original 8051 CPU, we can reduce code size to 578 bytes and time to 0.024642 seconds. The performance is greatly improved.

Furthermore, we can increase the secret key length to strengthen the security of our stream cipher. For example, we can use LFSR of length 196 or LFSR of length 256 and so on to strengthen the security. But we will need more code size and memory to do this. We can also increase correlation immunity, nonlinearity, algebraic degree and algebraic immunity of the Boolean function to have more power to resist all kinds of attacks. The easiest way is to increase variables of the Boolean function. For example, we can choose $n = 14$ in (39) and f as (14, 7, 6, 7936) the Boolean function of algebraic immunity 5. Of course, this will take more code size, memory and time. This is trade off. We may figure out a new method of construction to increase algebraic immunity in the same variables, nonlinearity,

correlation immunity and algebraic degree of the Boolean function. A lot of previous research is on how to construct the Boolean function to reach the highest nonlinearity, correlation immunity and algebraic degree, but few consider algebraic immunity, so we may modify these methods of construction to increase algebraic degree and do not change nonlinearity, correlation immunity and algebraic degree. If we can do so, we will save much code size, memory and computing time while achieving good security.

The stream cipher is applied to the sensors. If we want to improve the performance of sensors, we may develop system on a chip on the sensors. In addition, we may integrate all the components, e.g. RF, sensors, cryptosystems and so on, into one chip. This will greatly improve power consumption, the most important factor on sensors' performance.



Bibliography

- [1] M. Jakobsson and S. Wetzel, "Security Weaknesses in bluetooth," 1996.
Avaliable: <http://www.bluetooth.com>.
- [2] A. Biryukov, A. Shamir, and D. Wagner, "Real time cryptanalysis of A5 on a PC," in Proceeding Fast Software Encryption 2000, New York:Springer-Verlag, 2000, Vol. 1978, pp.1-18.
- [3] P. Ekdahl and T. Johansson, "Another attack on A5," in Proceedings of 2001 IEEE International Symposium on Information Theory, 2001, pp. 160-167.
- [4] G. Rose and P. Hawkes, "The t-class of SOBER stream ciphers,"
Avaliable:www.home.aone.net.au/qualcomm.
- [5] H. Beker and F. Piper, "Cipher systems: the protection of communication," John Wiley & Sons, New Work, 1982.
- [6] S. W. Golomb, "Shift register sequences," Holden-Day, San Francisco Calif., 1967.
- [7] J. L. Massey, "A self-synchronizing digital scrambler for cryptographic protection of data," 84 International Zurich Seminar on Digital Communications Applications of Source Coding, Channel Coding & Secrecy Coding, 1984.
- [8] A. Lempel and J. Ziv, "On the complexit of finite sequences," IEEE Transaction Information Theory, January 1969 IT-15, pp. 122-127.
- [9] J. L. Massey, "Shift-register synthesis and BCH decoding," IEEE Transaction on Information Theory, January 1976, IT-22.
- [10] T. Siegenthaler, "Correlation immunity of non-linear combining functions for cryptographic applications," IEEE Transaction on Information Theory, 1984, IT-30, pp. 776-780.
- [11] X. Guo-Zhen and J. Massey, "A spectral characterization of

- correlation-immune combining functions,” IEEE Transaction on Computers, 1988, Vol. C-34, pp. 81-85.
- [12] J. F. Dillon, “A survey of bent functions,” NSA Mathematical Meeting, 1972, pp. 191-215.
- [13] Anne Canteaut, Kapaleeswaran Viswanathan, “Results on Algebraic Immunity for Cryptographically Significant Boolean Functions,” Progress in Cryptology - INDOCRYPT 2004: 5th International Conference on Cryptology in India, Chennai, India, December 20-22, 2004. Proceedings.
- [14] J. L. Massy, “Cryptography and system theory,” Proceeding 24th Allerton Conference Communication , Control, Comput., Oct. 1-3, 1986.
- [15] P. Ekdahl and T. Hohansson, “SNOW-a new stream cipher,” in Proceedings of First Open NESSIE Workshop, KU-Leuven, 2000.
- [16] C. Carlet, “Improving the algebraic immunity of resilient and nonlinear functions,” Technical report, Cryptology ePrint Archive of the IACR, 2004. <http://eprint.iacr.org>
- [17] T. Siegenthaler, “Decrypting a class of stream ciphers using ciphertext only,” IEEE Transaction on Computers, 1985, C-34, pp. 81-85.
- [18] W. Meier and O. Staffelbach, “Fast correlation attacks on stream ciphers,” in Advances in Cryptology, EUROCRYPT’88, Springer-Verlag 1988, Vol. 330, pp. 301-314.
- [19] W. Meier and O. Staffelbach, “Fast correlation attacks on certain stream ciphers,” Journal of Cryptology, 1989, Vol. 1, pp. 159-176.
- [20] T. Johansson and F. Jonsson, “Improved fast correlation attacks on stream ciphers via convolutional codes,” in Advances in Cryptology, EUROCRYPT’99, Springer-Verlag, 1999, pp. 347-362.
- [21] Vladimor Chepyzhov, Thomas Johansson, and Bernard Smeets, “A simple

- algorithm for fast correlation attacks on stream ciphers,” In Bruce Schneier, editor, Fast Software Encryption (FSE 2000), Proceedings, Lecture Notes in Computer Science 1978, pages 181-195. Springer-Verlag, 2001.
- [22] Fredrik Jonsson and Thomas Johansson, “Theoretical analysis of a correlation attack based on convolutional codes,” In Ezio Biglieri and Sergio Verdu, editors, IEEE International Symposium on Information Theory 2000, page 212, 2000.
- [23] Anne Canteaut and Michael Trabbia, “Improved fast correlation attacks using parity-check equations of weight 4 and 5,” In Bart Preneel, editor, Advances in Cryptology, EUROCRYPT’00, LNCS 1807, pages 573-588. Springer-Verlag, 2000.
- [24] Miodrag J. Mihajevic, Marc P. C. Fossorier, and Hideki Imai, “fast correlation attack algorithm with list decoding and an application,” In Hideki Imai, editor, Fast Software Encryption (FSE2001), LNCS 2355, pages 196-210. Springer-Verlag, 2001.
- [25] Phillippe Chose, Antoine Joux, and Michel Mitton, “Fast correlation attacks: An algorithmic point of view,” In Lars Kundsén, editor, Advances in Cryptology, EUROCRYPT’02, LNCS 2332, pages 209-221. Springer-Verlag, 2002.
- [26] T. Johansson and F. Jönsson, “Fast Correlation Attacks Based on Turbo Code Techniques,” Advances in Cryptology, Crypt’99, Springer-Verlag, 2000, Berlin, 181-197.
- [27] Rejane Forre, “A fast correlation attack on nonlinearity feedforward filtered shift-register sequences,” In Jean-Jacques Quisquater and Joos Vandewalle, editors, Advances in Cryptology, EUROCRYPT’89, LNCS 434, pages 586-595. Springer-Verlag, 1990.

- [28] Jovan Dj. Golic, Mahmoud Salmasizadeh, Leonie Ruth Simpson, and Ed Dawson, "Fast correlation attacks on nonlinear filter generators," *Information Processing Letters*, 64(1):37-42, October 1997.
- [29] Fredrik Jonsson and Thomas Johansson, "A fast correlation attack on LILI-128," *Information Processing Letters*, 81(3): 127-132, February 2002.
- [30] Bernhard Lohlein, "Attacks based on conditions against the Nonlinear Filter Generator," Technical report, Cryptology ePrint Archive of the IACR, 2003. <http://eprint.iacr.org>.
- [31] R. A. Rueppel, "Analysis and design of stream cipher," Springer-Verlag, Berlin etc., 1986. [43]
- [32] 丁存生, 蕭國鎮, 流密碼學及其應用, 國防工業出版社, 1993。
- [33] N. Courtois and W. Weier, "Algebraic attacks on stream ciphers with linear feedback," In *Advances in Cryptology – EUROCRYPT 2003*, volume LNCS 2656, pages 346-359. Springer-Verlag, 2003.
- [34] N. Courtois, "Fast algebraic attacks on stream ciphers with linear feedback," In *Advances in Cryptology- CRYPTO 2003*, volume LNCS 2729, pages 176 – 194. Springer-Verlag, 2003.
- [35] Adi Shamir, Jacques Patarin, Nicolas Courtois, Alexander Klimov, "Efficient Algorithms for solving Overdefined Systems of Multivariate Polynomial Equations," *Eurocrypt'2000*, LNCS 1807, Springer, pp. 392-407.
- [36] Nicolas Courtois, "Higher order correlation attacks, XL algorithm and cryptanalysis of Toyocrypt," *ICISC 2002*, November 2002, LNCS 2501, pp.267-287, Springer, A preprint with a different version of the attack is available at <http://eprint.iacr.org/2002/87/>.
- [37] M. Mihaljevic, H. Imai, "Cryptanalysis of Toyocrypt-HS1 stream cipher," *IEICE Transactions on Fundamentals*, vol. E85-A, pp.66-73, Jan. 2002.

Available at <http://www.csl.sony.co.jp/ATL/papers/IEICEjan02.pdf>.

- [38] W. Meier, E. Pasalic, C. Carlet, “Algebraic attacks and Decomposition of Boolean Functions,” *Advances in Cryptology - EUROCRYPT 2004*, Interlaken, Proceedings, LNCS 3027, pp. 474 – 491.
- [39] Jovan Dj. Golic, ”On the security of nonlinear filter generators,” In Dieter Gollmann, editor, *Fast Software Encryption (FSE 1996)*, LNCS 1039, pages 173-187. Springer-Verlag, 1996.
- [40] Jovan Dj. Golic, Andrew Clark, and Ed Dawson, “Inversion attack and branching,” In Josef Pieprzyk, Reihaneh Safavi-Naini, and Jennifer Seberry, editors, *Information Security and Privacy, Fourth Australasian Conference, ACISP’99*, LNCS 1587, pages 88-102. Springer-Verlag, 1999.
- [41] Jovan Dj. Golic, Andrew Clark, and Ed Dawson, “Generalized inversion attack on nonlinear filter generators,” *IEEE Transactions on Computers*, 49(10): 1100-1109, October 2000.
- [42] A. Gorska and K. Gorski, “Improved inversion attacks on nonlinear filter generators,” *Information Processing Letters*, 38(16): 870-871, August 2002.
- [43] Sabine Leveiller, Joseph Boutros, Philippe Guillot, and Gilles Zemor, “Cryptanalysis of nonlinear filter generators with $\{0, 1\}$ -metric Viterbi decoding,” In Bahram Honary, editor, *Cryptography and Coding VIII*, LNCS 2260, pages 402-414. Springer-Verlag, 2001
- [44] B. Löhlein, “Analysis of modifications of the conditional correlation attack,” 1999. Accepted at 3rd IEEE/ITG Conference on Source and Channel Coding, 17-19 Jan. 2000, Munich.
- [45] Richard A. Games and Joseph J. Rushanan, “Blind synchronization of m-sequences with even span,” In Tor Helleseth, editor, *Advances in Cryptology, EUROCRYPT’93*, Lecture Notes in Computer Science 765, pages 168-180.

- Springer-Verlag, 1994.
- [46] Eric Filiol, “Decimation attack on stream ciphers,” In Bimal Roy and Eiji Okamoto, editors, Progress in Cryptology, INDOCRYPT 2000, LNCS 1977, pages 31-42. Springer-Verlag, 2000.
- [47] S.S. Bedi and N.R. Pilai, “Cryptanalysis of the nonlinear feedforward generator,” In Bimal Roy and Eiji Okamoto, editors, Progress in Cryptology, INDOCRYPT 2000, LNCS 1977, pages 188-194. Springer-Verlag, 2000.
- [48] Alex Biryukov and Adi Shamir, “Cryptanalytic time/memory/data tradeoffs for stream ciphers,” In Tsutomu Matsumoto, editor, Advances in Cryptology, ASIACRYPT’00, Lecture Notes in Computer Science 1976, pages 1-13. Springer-Verlag, 2000.
- [49] Matthias Krause, “BDD-based cryptanalysis of keystream generators,” Technical report, Cryptology ePrint Archive of the IACR, 2001. TR-2001-092, <http://eprint.iacr.org>.
- [50] Matthias Krause, “BDD-based cryptanalysis of keystream generators,” In Lars Knudsen, editor, Advances in Cryptology, EUROCRYPT’2002, Lecture Notes in Computer Science 2332, pages 222-237. Springer-Verlag, 2002.
- [51] B. Löhlein, “Attacks based on conditional correlations against the nonlinear filter generator,” Technical report, Cryptology ePrint Archive of the IACR, 2003. <http://eprint.iacr.org>.
- [52] Turiy Tarannikov, “On resilient Boolean functions with maximal possible,” Crypto ePrint Archive, <http://eprint.iacr.org>, No. 2000/005.
- [53] Patrik Ekdahl, Tomas Johanson, “SNOW – a new stream cipher,” Proceeding of first NESSIE Workshop, Heverlee, Belgium, 2000.
- [54] A. J. Menezes, P. C. V. Oorschot, and S. A. Vanstone, Handbook of Applied Cryptography, pp.169-190, 1996.

- [55] Markus Schneider, “Methods of generating binary pseudo-random sequences for stream cipher encryption (in German),” PhD thesis, Faculty of Electrical Engineering, University of Hagen, Germany, September 1999. Berichte aus der Kommunikationstechnik, Band 4, Shaker Verlag.
- [56] Perrig, R. Szewczyk, V. Wen, D. Culler, and J. Tygar, “SPINS: Security Protocols for Sensor Networks,” In Seventh Annual ACM International Conference on Mobile Computing and Networks (Mobicom 2001), Rome Italy, July 2001.

