A NEW REAL-TIME TIME-MEMORY TRADE-OFF ATTACK ON A5/1 STREAM CIPHERS USING "PECULIAR EVENTS"

by

Onur Yazgan

B.S., Electrical and Electronics Engineering, Boğaziçi University, 2002

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Electrical and Electronics Engineering
Boğaziçi University

2005

Dedicated to my mother,
whose endless motivation and encouragement
made me complete this work

ACKNOWLEDGMENTS

I would like to take this opportunity to sincerely thank all the valuable faculty members in electrical engineering department for their continuous support during my master's studies and especially to Prof. Emin Anarim for his enormous help, guidance, patience and trust. It has been an honor for me to work with him and closely observe his genius at engineering and interpersonal relations.

ABSTRACT

A NEW REAL-TIME TIME-MEMORY TRADE-OFF ATTACK ON A5/1 STREAM CIPHERS USING "PECULIAR EVENTS"

In this paper, we describe a new real-time attack on A5/1 stream ciphers that achieves superior performance in terms of lower computation time and calculation complexity compared to most of the previous attacks of similar type. We introduce the concept of "peculiar events" that can be applied to any array of mutually clock-controlled stop/go shift registers but works exceptionally well in A5/1 due to a set of subtle flaws in its design, particularly the position of its clocking tabs. The proposed attack is an advancement of the "peculiar events" concept by embedding it into an improved version of M. Hellman's cryptanalytic time-memory tradeoff attack.

We show that the proposed technique is suitable for real-time attacks on commercially available computers with a pre-computational complexity of $2^{41.33}$ clockings, a memory requirement of $2^{40.21}$ bits (<160 GB), real-time complexity of 2^{25} unit computation durations and ~3 seconds of known conversation.

Finally, we compare our attack's performance with other widely accepted techniques and conclude that ours achieves acceptable success rates at comparably lower memory, data and calculation complexity requirements.

ÖZET

A5/1 AKIM ŞİFRELEYİCİLERİNE "YABANSI OLAYLARA" DAYALI YENİ BİR GERÇEK-ZAMANLI, ZAMAN-BELLEK ÖDÜNLEŞİMLİ SALDIRI ÖNERİSİ

Bu çalışmada, şimdiye kadar A5/1 akım şifreleyicilerine gerçekleştirilmesi önerilmiş önceki saldırılara kıyasla daha az hesaplama karmaşıklığı ve daha kısa işletme süresi gereksinimine sahip, bu sayede de daha yüksek başarımlı, yeni bir gerçek-zamanlı zaman-bellek ödünleşimli saldırı yöntemi önerilmektedir.

Çalışmanın ana taşı, herhangi bir karşılıklı saat güdümlü dur/git kayan sarkacına da uygulanabilecek ancak A5/1 akım şifreleyicisinin tasarımındaki bazı kusurlar (özellikle de saat güdülme sekmelerinin yeri) yüzünden A5/1'e bilhassa kolay ve başarılı bir biçimde uygulanabilecek "yabansı olaylar" kavramının ortaya atılmasıdır. Önerilen saldırı yöntemi, "yabansı olay" kavramının M. Hellmann'ın kripto analitik zaman-bellek ödünleşimli saldırısının gelişmiş bir sürümüne gömülmesi ile sağlanmaktadır.

Çalışmada, önerilen yöntemin ticari olarak elde edilebilecek kişisel bilgisayarlar üzerinden gerçek-zamanlı olarak yapılmasına olanak sağlayacak şekilde, $2^{41.33}$ saat vurumu kadar ön hesaplama karmaşıklığına, $2^{40.21}$ bitlik (<160 GB) bellek gereksinimine, 2^{25} birim hesaplama süresi kadar gerçek zaman karmaşıklığına ve yaklaşık üç saniyelik bilenen görüşme gereksinimine sahip olduğu gösterilmektedir.

Sonuç olarak, önerilen saldırı yönteminin başarımı, yaygın olarak kabul edilmiş diğer saldırı yöntemleriyle kıyaslanmakta ve önerilen yöntemin kabul edilebilir bir başarı oranını, nispi olarak daha az hesaplama karmaşıklığı, daha düşük bellek ve veri gereksinimi ile sağladığı ispatlanmaktadır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS

C1, C2, C3	Clocking tab of each register
C_n	Cipher text frames
D	Amount of real-time data available to the attacker
fi()	i^{th} mask function
F_n	Frame counter
K	Secret session key
ki	i^{th} secret session key
Ker	Kernel function
LFSR	Linear Feedback Shift Register
M	Random access memory requirement
mi	i^{th} bitwise operator of the mask function
N	Search space (Number of possible secret session keys)
P	Time required in the pre-processing phase of the attack
Q	Set of the linear equation of clock controlling bits
R1, R2, R3	Linear Feedback Shift Registers in A5/1
R	Reduction function
T	Time required in the real-time phase of the attack
xi	Random states of the registers
yi	64-bit output prefixes
ζt	Clock control value at time t

1. INTRODUCTION

The security of the over-the-air GSM voice and data communication is based on usage of the A5 stream ciphers. A5 stream ciphers are a common type of key stream generators for additive stream cipher applications consisting of a number of irregularly clocked linear feedback shift registers (LFSRs) that are combined by a function with or without memory. Standard cryptographic criteria such as a large period, a high linear complexity, and good statistical properties are thus relatively easily satisfied in these types of stream ciphers [1,2].

Although intended to be kept secret, the approximate design of A5/1 was leaked in 1994, and the exact design was reverse engineered from an actual GSM telephone in 1999 [3].

Since then, various attacks have been proposed on A5/1, in which the attacker is assumed to know some pseudo random bits generated by A5/1 in some of the frames. This is a standard assumption in cryptanalysis and we will not consider in this work the way to obtain this information in the open-field. To further focus on the core issue, it will be assumed that the attacker has complete knowledge on the input bits during some initial period of the communication and his goal is to find the secret session key in order to decrypt the remaining of the conversation.

1.1. Key Observations and Considerations

The new attack presented in this work is based on:

- Use of internal states to recover the initial state of known plain and cipher text pairs through partial implementation of sub-branching [4],
- Golic's application of Hellman's Time/Memory Tradeoff attack [5] on A5/1, (extended by the "distinguished points" concept by Borst, Preneel and Vandewalle [6]),
- Use and efficient sampling of special states as proposed by Biryukov, Shamir and

Wagner [7],

- Utilizing state transitions instead of actual content to store each register's content to enable an efficient software implementation; i.e. fast computation with reasonably low disk storage capacity,
- Incorporating Biham's and Dunkelman's idea of "constant registers for consecutive clock rounds", i.e. waiting until an event which leaks a large amount of information about the key occurs and then exploiting the attack [8],
- Use of wide-belt internal states in finding the candidate initial state, i.e. benefiting from the "biased birthday attack" concept,
- Improvements to Golic's and Hellman's time-memory tradeoffs through new methods of calculating and storing pre-computed data [9] with rainbow chain models and recent advancements of this method by Erguler and Anarim [10].

1.2. Purpose of this Work

Each of the seven key ideas stated above has already been investigated individually in some way or the other. However, any comprehensive effort to combine a subset of them to achieve superior decryption performance either failed or was not extensively published yet.

In this work, by further analyzing the cryptanalytic flaws of the A5 stream cipher, we aim at shedding more light on ways of coupling several of these key ideas. The work is driven by the introduction of the "peculiar events" concept and a new time-memory trade-off attack is presented that enables a faster implementation in software.

We summarize the major flaws considered of special importance for the derivation of the new real-time attack as follows:

- Enter the clocking: Each one of the three shift registers has an irreducible characteristic polynomial and since they are very small in size (19,22,23 bits respectively) it is possible to pre-compute all of their successive states and keep them as three cyclic arrays in memory, where successive locations in each array

correspond to successive states of the according shift register. In fact, since at most only a single bit differs (LSB) between the consecutive states, there is no need to store the register content in the memory but just to save the difference. Thus, a state can be viewed as a triplet of indices (a,b,c) and the computation of internal states/output sequences can follow through simple use of state-transition operations.

- A second major flaw back of the A5/1 cipher is that each register has a single "clocking tap" and that these taps are positioned at cryptanalyticly "weak" locations (the 8th ,10th and 10th bits, respectively). The reason for these locations being "weak" is that the majority function is used as the clock generator and output sequences with a desired prefix up to 16-bits can be calculated without trying and discarding approximately more than 264-16=248 states.
- A third flaw worth to further investigate is the weight distribution of initial candidate states, i.e. Biryukov, Shamir and Wagner claim that about 85 per cent of the candidate internal states have zero weight because their trees die out before reaching the critical depth of 100 clocks. (We provide detailed information on tree structure and on the importance of exceeding a depth of 100 clocks in the following sections.)

It is obvious that such a coupled stream cipher would in principle be vulnerable to various "divide-and-conquer attacks" on the secret key. Typically, the attacks require an exhaustive search over the initial states of a subset of the LFSR's and are hence feasible only if the effective secret key controlling these initial states is short. In some cases, faster cryptanalytic attacks which may work for long regularly clocked LFSR's have also been proposed.

In this paper, we give a solid background on these core issues and their potential application to implementing the proposed new attack in software. We present other flaws crucial to further improve implementation performance of the new attack.

This paper is organized as follows: In Section Two, we start with an overview of cryptology and cryptanalysis. In Section Three, we give a short description of A5/1's architecture. Section Four considers with the key ideas behind major previous attacks. Section Five introduces the concept of "peculiar events". In Section Six, we present P. Oechslin's improvement to time/memory tradeoff attacks. In Section Seven, we describe

our new real-time attack. In Section Eight, we investigate the performance of our attack. Finally, in Section Nine we mention some possible improvement ideas to our proposed attack and state concluding remarks for future research.

2. OVERVIEW OF CRYPTOLOGY AND CRYPTANALYSIS

2.1. Introduction to Cryptology

The history of cryptology is long and there are several fascinating stories to tell [11]. Different systems for encrypting messages have been used by governments and the military to prevent national or military secrets to be revealed by enemies. One of the most famous cipher systems is Enigma, used by Germany in the World War II. It has been argued that when Enigma was broken by the allied, the outcome of the war changed. Two books that describes classical cryptosystems in a non-technical fashion are The Codebreakers by Kahn [12] and The Code Book by Singh [13].

The open scientific study of cryptology started with the World War II. One of the first and most celebrated scientific papers in the area of cryptology was written by Claude E. Shannon [14]. There, Shannon uses information theory to give a theoretical foundation of cryptology. From this pioneering work, cryptology as a science has grown. Cryptology uses ideas not only from information theory but also from other fields such as, computer science, probability theory, number theory and abstract algebra.

The traditional way of using cryptography is for confidentiality. The cryptographic model can be described in the following way. Alice wants to send a private message m to Bob. The message is taken from a set of possible messages or (plaintexts), denoted by M. Since the eavesdropper, here called Eve, can listen to the communication, Alice needs to encrypt the message to keep it private. The two users, Alice and Bob, share a secret key K, where K is taken from the set of possible keys, K, called the keyspace. For each K 2 K, there is an encryption function, denoted by E_K, that maps a message m 2 M into a cipher text c 2 C, where C is the set of possible cipher texts, i.e., $c = E_K(m)$. The cipher text c is then transmitted over the insecure channel. The key K also specifies a decryption function D_K , where the decryption function is chosen such that $m = D_K(E_K(m))$ for all possible keys K and messages m. Bob, who has the same key as Alice, can retrieve the plaintext m after receiving the cipher text c by applying the decryption function $m = D_K(c)$.

Since Eve does not know the secret key she cannot use the correct decryption function to get the plaintext m. What she can do, however, is to mount an attack and try to estimate the plaintext without knowing the key. It is important to note that we assume that Eve knows all properties of the cryptosystem except the actual key, K, that was used to encrypt the message.

As an example of a classical cryptosystem we can describe the Vigenère cipher. The message m = m_0, m_1,… and the cipher text c = c_0, c_1, …are both sequences of letters from the English alphabet, i.e., m_i, c_i ε {A, B, …, Z}.The key K is a sequence of letters from the English alphabet of length k, K = K_1, K_2, …, K_k. To encrypt a message, the message and the key are transformed to integer sequences by the transformation, A ↔ 0; B ↔ 1; …. ; Z ↔ 25. Denote the integer sequence corresponding to the message and the key by m_0 and K_0, respectively. The sequences m_0 and K_0 are then used to calculate a new sequence c'$_1$, c'$_2$, …, where

$$c_i' = m_i' + K_{i \bmod k}' \bmod 26 \text{ , i = 0, 1, 2, …} \qquad (2.1)$$

From the sequence c' the cipher text c is given by the same transformation that was used to transform the message to the integer sequence m'.

Example: We use a Vigenère cipher to encrypt the message m = TOBEORNOTTOBE. Let the key be K = HAMLET. First, the message is transformed into the sequence m_0 = 19; 14; 1; 4; 14; 17; 13; 14; 19; 19; 14; 1; 4, and the key is transformed into K_0 = 7; 0; 12; 11; 4; 19. The sequence c 0 is then calculated, and we get c_0 = 0; 14; 13; 15; 18; 10; 20; 14; 5; 4; 18; 20; 11. Finally, we get the cipher text c = AONPSKUOFESUL.

In the classical model of cryptography, Alice and Bob share the same key. This is usually called symmetric-key encryption. In [15] W. Diffie and M. E. Hellman proposed another model for encryption using two separate keys, one for Alice and one for Bob. The elegant property of this model is that the encryption key used by Alice does not have to be kept secret. Since the encryption key can be publicly known, such a system is called a

public-key cryptosystem. The decryption key, used by Bob, still has to be private and known only to Bob.

The security of public-key cryptosystems relies on the existence of one-way functions and trapdoor one-way functions.

Definition: A function f is called a one-way function if it is computationally easy to compute $y = f(x)$ for all x but for almost all y it is computationally infeasible to compute the inverse $x = f^{-1}(y)$.

Formally, one should also provide definitions of the concepts computationally easy and computationally infeasible, mentioned in the definition. However, for our treatment the intuitive meaning is enough. It is not possible to use a one-way function directly for encryption. The receiver Bob has to have an efficient decryption function. Thus we need a trapdoor one-way function.

Definition: A function f is called a trapdoor one-way function if it is a one-way function with the additional property that the inverse is easy to compute if we have knowledge of some extra information, called the trapdoor.

A public-key cryptosystem can be described in the following way. Assume that Bob wants to set up a public-key cryptosystem such that Alice can send a private message to Bob. Bob chooses a trapdoor one-way function, denoted by E_e , and transmits it to Alice over the insecure channel. Since Bob knows the trapdoor, he can also find the inverse function, denoted by D_d , that is kept private by Bob. The function E_e is then used as the encryption function. As before, let M be the set of possible plaintexts and C be the set of possible cipher texts. Alice can then encrypt a message $m \varepsilon M$ as $c = E_e(m)$. Bob knows the inverse function and can decrypt the cipher text c as $m = D_d(c)$. Due to the one-way property of E_e, an eavesdropper cannot calculate the plaintext m, even if both E_e and c were observed.

The first public-key cryptosystem was the RSA cryptosystem by Rivest, Shamir, and Adleman [16]. The trapdoor one-way function used in the RSA cryptosystem is based on

the difficulty of factoring large integers. Since then, several other public-key cryptosystems have been proposed. However, all public-key cryptosystems used in practice today are much slower than modern symmetric-key cryptosystems, which make them less suitable for encrypting large amounts of plaintext.

The description of cryptology in this section has focused on privacy, i.e., methods to make sure that a message can only be read by authorized users. There are also a lot of other subjects in cryptology: Assume that we have received a message that is claimed to have been sent from Alice. One question we can ask, is if this message is correct and has not been tampered with by an opponent during the transmission. Cryptographic solutions to make sure that a message has not been altered during transmission is usually called data integrity. Another question is how we can be sure that it actually was Alice that transmitted this message. This is called data origin authentication. There may also be a need for entity authentication, where an entity, or user, can identify itself to other users.

For a more thorough treatment of different subjects in cryptology, different textbooks are available, see for instance [17, 18, 19].

2.2. Introduction to Cryptanalysis

In the previous section different cryptosystems were discussed. A natural question that follows is how secure these systems are. If a weakness has been found in a cryptosystem, we say that the cryptosystem is broken. It is hard to give a precise definition for when a cryptosystem is broken. The definition of broken will depend on what kind of cryptosystem that has been used, and also on the method used to break it. A definition that often is used for symmetric-key cryptosystems is the cryptosystem is considered to be broken if we can find the key or the plaintext faster than by an exhaustive key search.

However, other definitions can also be considered. This discussion leads us into cryptanalysis, the study of the security of cryptographic primitives and functions. When discussing the security of cryptosystems we usually distinguish between unconditionally and conditionally secure cryptosystems. An unconditionally secure cryptosystem cannot be

broken, even with infinite computational resources. The security of a conditionally secure cryptosystem relies on some assumption of the opponent's computing power.

Ideally, we would like to have cryptosystems that are unconditionally secure. In [14] Shannon uses information theory to analyze unconditional security of cryptosystems. One example of an unconditionally secure cryptosystem is the one-time pad.

Definition: The one-time pad is a cryptosystem where M = C = K = F_2^n. The key K = K_1,K_2, …., K_n is randomly chosen and may never be reused. A message m = m_1, m_2, …,m_n is encrypted into the cipher text c = c_1, c_2, …,c_n by calculating

$$c_i = m_i + k_i, \; 1 < i < n \tag{2.2}$$

To set up a one-time pad cryptosystem, we need to transmit a key that is at least as long as the plaintext over a secure channel. Instead we can transmit the plaintext directly over the secure channel. However, such a system might be useful in the case where we have access to a secure channel only in a small period of time. Then we can use this time period to send a key over the channel. This transmitted key can then be used at a later time to communicate over an insecure channel.

From the discussion above we conclude that in most cases an unconditionally secure cryptosystem is impractical. Thus, one must rely on cryptosystems that are conditionally secure. For some cryptosystems one can prove that breaking the cipher is equivalent to solving a computational problem that is supposed to be hard. Examples of such problems are factoring large integers and solving the discrete logarithm problem. This kind of arguments for the security of a cryptosystem is called provable security.

For most of the symmetric-key cryptosystems one must rely on a much weaker definition of conditionally secure cryptosystems. The security of a cipher is given as the complexity, or number of operations, of the currently best known attack. For ciphers with such security assumptions there is much research ongoing for improving the best known attacks. A typical example of such research is improvements of fast correlation attacks on stream ciphers.

The different attacks that can be mounted on a cryptosystem can be divided into different classes depending on the power of the attacker. Four of the different classes of attacks are the following:

- Cipher text-only: This is the worst case for the cryptanalyst. In cipher text-only attacks the cryptanalyst tries to recover the plaintext or the key, when he observes only the cipher text.
- Known-plaintext: The aim of a known plaintext attack is to deduce the key, when the cryptanalyst knows some plaintext and the corresponding cipher text.
- Chosen-plaintext: Like a known plaintext attack, the aim of a chosen plaintext attack is to recover the key. In this case the cryptanalyst can choose any plaintext and gets the corresponding cipher text.
- Chosen-cipher text: This case has similarities with chosen plaintext attacks. The difference is that we assume that the cryptanalyst have access to the decryption unit and can decrypt any cipher text.

We will have more to say about each of attack in the following sections.

3. A BRIEF DESCRIPTION OF A5/1

3.1. Introduction to Stream Ciphers

A key stream generator is a pseudo-random sequence generator or running key generator. The stream cipher encryption and decryption can be formulated as follows: Let k_1, k_2, k_3,..., k_i denote the sequence that key stream generator outputs and p_1, p_2, p_3,..., p_i denote the plaintext bits. Then, if c_1, c_2, c_3,..., c_i represents the corresponding cipher text bits, encryption and decryption are realized according to the equations below respectively [21].

$$c_i = p_i \oplus k_i \tag{3.1}$$

$$c_i \oplus k_i = p_i \oplus k_i \oplus k_i = p_i \tag{3.2}$$

Stream ciphers can be classified as synchronous or self-synchronizing stream ciphers according to the relation between key stream generation and plaintext [22].

A synchronous stream cipher is one in which the key stream is generated independently of the plaintext message and of the cipher text [23]. On the encryption side, a key stream generator outputs the key stream bits, one after the other. On the decryption side, another key stream generator produces the identical key stream bits, one after the other. To avoid false decryption in communication, the two key stream generators must be synchronized. In case of losing synchronization during transmission, every cipher text character after the error will be decrypted incorrectly. To solve this problem, the sender and receiver must resynchronize their key stream generators before continuing communication. Techniques for re-synchronization can be re-initialization or placement of special markers at regular intervals in the cipher text. An advantage of the synchronous stream cipher is that synchronous ciphers do not propagate transmission errors. If a single bit or multiple bits alter during transmission following occurs, then only error bits will be decrypted incorrectly, all preceding and subsequent bits will be unaffected. Most of the stream ciphers are binary additive stream ciphers that are synchronous stream ciphers in

which the keystream, plaintext, and cipher text digits are binary digits, and the output function is the XOR of plaintext and key stream sequence.

A self-synchronising stream cipher is a finite state machine for which the key stream is generated as a function of the key and a fixed number of the previous cipher text symbols [24]. In other words, for this type of stream ciphers each key stream bit is produced within a function of a fixed number of previous cipher text bits. Since the key stream depends on a fixed number of the previous cipher text symbols say v, the cipher will resynchronize after v symbols if there is a transmission error. In this case, the next v symbols will be erroneous and the error propagation is thus worse than for a synchronous stream cipher. However, if some cipher text symbols are deleted or inserted during transmission, the self-synchronizing cipher were to recover after v correct cipher text symbols, whereas the synchronous ciphers will never regain synchronization.

3.2. Linear Feedback Shift Registers

An FSR (Feedback Shift Register) is a device made up by registers that produces binary sequences or symbols from a field F_q where $q=2^k$ and k is the symbol size (for our case and most of the stream ciphers q is chosen as 2). These registers are the main components of many key stream generators and they are used both in coding and cryptography.

A feedback shift register is made up of two parts; a shift register *s* and a feedback function *f*. If the shift register *s* has a length of *n* bits or consists of *n* stages as s_1, s_2, ..., s_n which contain a single bit 0 or 1, it is called an *n*-bit shift register. The feedback function maps the state of the shift register according to its bits content.

When the register is clocked at a time interval, all of the bits in the shift register are shifted one bit to the right. The new value of the left-most bit is computed by applying the feedback function to the contents of the register before clocking. At each clock, the right most bit of the register can be concerned as its output. The period of a shift register is the length of the output sequence before it starts repeating itself.

The simplest kind of feedback shift register is a linear feedback shift register. In that case, the feedback function can be written as $c_1 s_1 \oplus c_2 s_2 \oplus c_3 s_3 \ldots \oplus c_n s_n$, where s values are the contents of the register at time t and c values are the feedback coefficients.

As can be seen the feedback function is linear and simply the XOR of the appropriate bits in the register according to whether or not c_i is equal to 1 or 0; the list of the bits that have feedback coefficient value as 1 is called a tap sequence. Since the feedback function is linear and simple, many mathematical theories have been applied to analyzing LFSRs. The mathematical expression for the period of the shift register depends on its characteristic feedback function. If the feedback function is a primitive polynomial, then the period of the register becomes 2^n-1, where n is the length of the register.

An irreducible polynomial $f(x) \in F_q[x]$ of degree l is said do be primitive if the root of $f(x)$ in the splitting field F_{q^l} is a generator of multiplicative group $F^*_{q^l}$; where a polynomial $g(x) \in F_q[x]$ is defined as irreducible polynomial over F_q, if it can not be factored into polynomials of smaller positive degrees in the ring of polynomials $F_q[x]$. For our binary case, we can restrict the definition of irreducible polynomial and primitive polynomial as: A polynomial $f(x)$ over GF(2) is said to be an irreducible polynomial over GF(2) if the only polynomials over GF(2) which divide $f(x)$ are 1 and itself. An irreducible polynomial $f(x)$ of degree n, which is also the length of the shift register, over GF(2) is said to be a primitive polynomial, if $(2^n -1)$ is the least positive integer p such that $f(x)$ divides $(1 + x^p)$ over GF(2).

If we start with a non-zero state as the initial state of the LFSR and the register have a primitive feedback polynomial, then all possible states except the all-zero state will appear during a period and the length of the period will be 2^n-1 as stated before. An LFSR with a primitive feedback polynomial is also called a maximum-length LFSR, and the sequence generated is called a maximum-length sequence. Notice that to say the sequence is maximum length, the initial state of the register must be non-zero and hereafter it is assumed that the starting state is as such. For example if the register has a length of 3 bits and a primitive feedback polynomial then the period of the register will be 2^3-1=7.

Most of the practical stream ciphers use LFSRs in their designs. There are several reasons for this: Firstly, LFSRs are well suited for hardware implementation, because an LFSR is nothing more than an array of bit memories and its feedback function is just use of a series of XOR gates. Therefore, within a few logic gates an LFSR based stream cipher can be realized. Second reason is LFSRs can generate sequences with large period. An L-bit maximal length LFSR can produce a sequence of 2^L-1, so as L increases the length of the period becomes incredibly large. The third is the fact that LFSRs produce sequences with good statistical properties. That is, they can produce random-looking key stream sequences.

However, LSFRs can easily be analyzed using algebraic techniques, due to their linear structure. The Berlekamp-Massey algorithm can generate sequence of an *n*-bit LFSR after using only 2*n* bits of the key stream. Thus, if an attacker gets 2*n* bits of key stream he can break the stream cipher which is based on a pure single *n* bit LFSR. Considering Berlekamp-Massey algorithm, the strength of an LFSR stream cipher against such an attack can be evaluated by using the metric linear complexity or linear span. The linear complexity of a sequence say *s,* denoted by L(*s*), is the length of the shortest LFSR that generates the same sequence. Linear complexity is very important, since the Berlekamp-Massey algorithm, can generate the sequence of a stream cipher with a linear complexity *n*, after examining only 2*n* bits of the key stream.

To eradicate linear complexity problems of the LFSRs and keeping their strong encryption characteristics, different approaches that will be discussed in the following sections.

3.3. Description of A5/1

In GSM systems, the conversation is sent between transmitting and receiving parties as a sequence of 4.6 miliseconds long frames. Each frame consists of 228 bits in which the first 114 bits conveys the message from A to B while the second half from B to A. Each session is encrypted with a secret session key K. For each frame to be sent, the session key

K is mixed with a publicly known frame counter denoted by F_n, and the result serves as the initial state of the shift registers in the A5/1 generator. It then generates 228 bits of running key for the current frame that is binary-XOR'ed with the plaintext to produce the cipher text [25].

A5/1 consists of three liner feedback shift registers of lengths 19, 22 and 23 that will be denoted R1, R2, and R3, respectively. The three LFSRs all have primitive feedback polynomials. The running key of A5/1 is given as the XOR of the output of the three LFSRs. (See Figure 3.1)

The LFSRs are clocked in an irregular fashion. It is a type of stop/go clocking with a majority rule as follows: Each register has a certain clocking tap, denoted C1, C2, C3, respectively. Each time the LFSRs are clocked, the three clocking taps C1, C2, C3 determine which of the LFSRs are to be clocked. R1 and R2 are clocked, but not R3, if C1 = C2 ≠ C3; R1 and R3 are clocked, but not R2, if C2 ≠ C1 = C3; R2 and R3 are clocked, but not R1, if C1 ≠ C2 = C3; finally, R1, R2 and R3 are all clocked, if C1 = C2 = C3. It should be noted that at each step at least two LFSRs are clocked and that the probability for an individual LFSR being clocked is 3/4 [26].

After the initialization procedure for the LFSRs, 228 bits of running key are produced, using irregular clocking. In each step one bit of the running key is calculated as the binary XOR of the current output bits from the LFSRs. The initialization process uses the session key K and the known frame counter F_n. First the LFSRs are initialized to zero. They are then clocked 64 times, ignoring the irregular clocking, and the key bits of K are consecutively XORed in parallel to the feedback of each of the registers. In the second step the LFSRs are clocked 22 times, ignoring the irregular clocking, and the successive bits of F_n are again XORed in parallel to the feedback of the LFSRs. The state of LFSRs at this time is generally called the initial state of the frame. In the third step the LFSRs are clocked 100 times with irregular clocking, but ignoring outputs. Then, the LFSRs are clocked 228 times with the irregular clocking, producing 228 bits of the running key. For a more detailed description of A5/1 refer to [27].

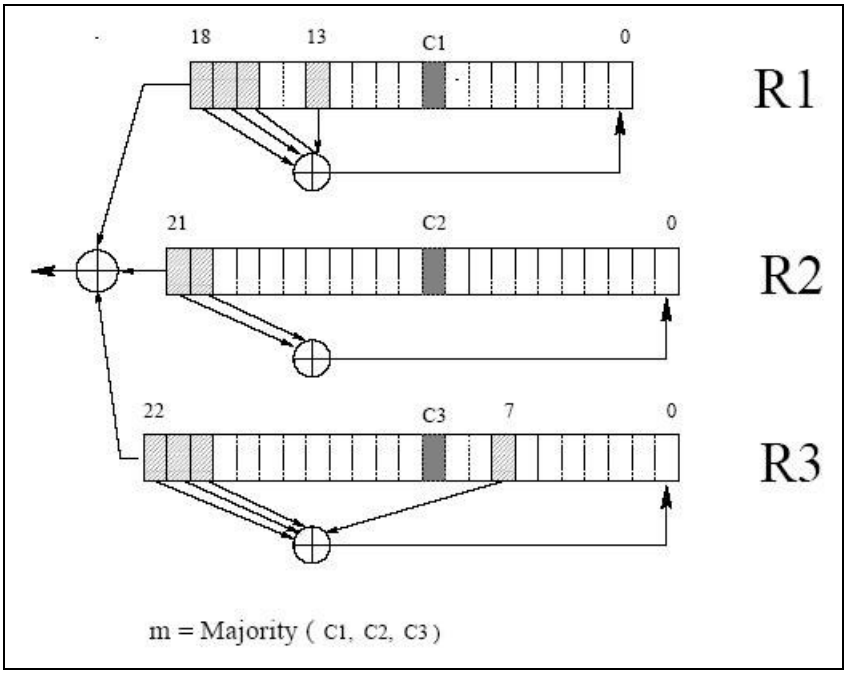

Figure 3.1. Design of the A5/1 stream cipher

3.4. Description of A5/2

A5/2 consists of four LFSRs of lengths 19, 22, 23, 17 bits denoted by R1, R2, R3 and R4 respectively. Each register is updated by its own primitive feedback polynomial. The feedback functions of R1, R2 and R3 of A5/2 are same as those of R1, R2, R3 of A5/1. Also the initialization part is very similar to that of A5/1. In A5/2, clocking mechanism of R1, R2 and R3 is controlled by R4. The bits, R4[3], R4[7] and R4[10], are given as input to clock controlling unit and this unit provides a binary result according to majority rule mentioned in previous part. If the result and R4[3] are equal, then R2 is clocked, if the result and R4[7] are equal, then R3 is clocked and if the result and R4[10] are equal then R1 is clocked. After these clockings R4 is clocked. As can be seen, the clock control mechanism of R1, R2 and R3 are equal in both A5/1 and A5/2. However inputs to clocking control unit are from R4 in case of A5/2, while in A5/1 inputs are from R1, R2 and R3.

After the clocking is performed the output bit is ready. The output bit is computed as follows: in each register majority of two bits and complementary of a third bit is calculated; the results of all the majorities and the right most bit from each register are XORed to form the output bit. (See Figure 3.2)

A5/2 mechanism runs in three steps, i.e. discarding first 99 produced bits part and producing 228 bits of key stream sequence part. In the initialization step, 64 bit secret session key K_c XOR'ed (ignoring majority rule) in parallel into the LSBs of the four registers which are all zeroed at the beginning of the process. Then 22 bit frame number F_n XOR'ed (ignoring majority rule) in parallel into the LSBs of the three registers. Then the bits R1[15], R2[16], R3[18] and R4[10] are forced to be 1. In second step, A5/2 is run are for 99 clock cycles according to clocking rule, however produced 99 bits are discarded. Finally, in the last step A5/2 is run according to clocking mechanism to produce 228 bits of key stream sequence. The first 114 bits of 228 bit-key stream sequence is used in encryption of the link from base station to the user and remained 114-bit sequence is used in encryption of the link from user to the base station [28].

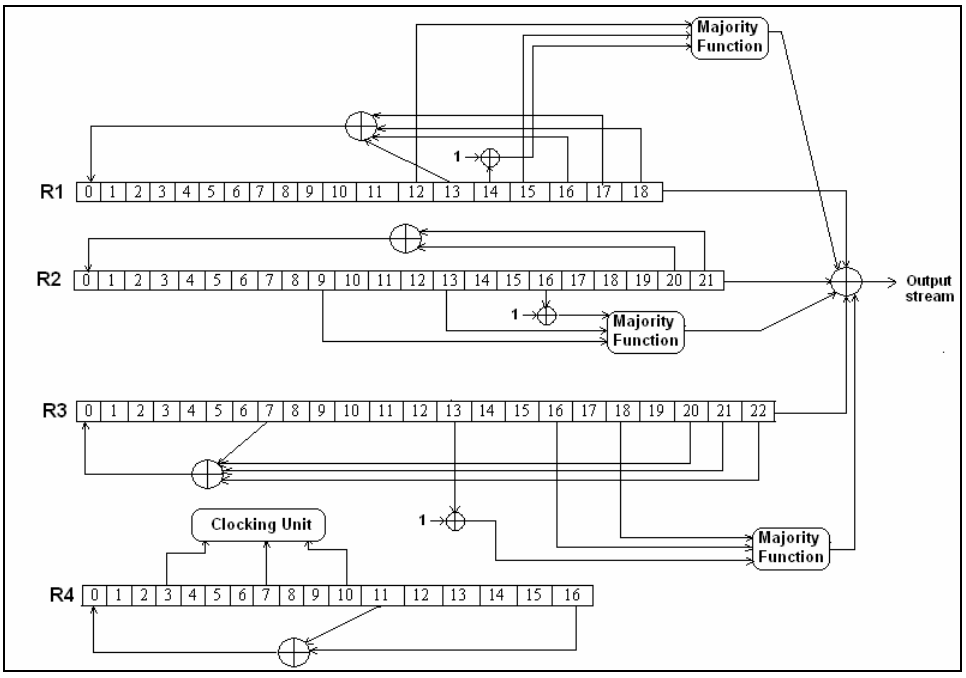

Figure 3.2. Design of the A5/2 stream cipher [24]

3.5. Other Stream Ciphers Proposed for A5/1

In their paper, Erguler and Anarim propose a new stream cipher named A5/H that is built of four LFSR's of lengths 19, 22, 23, 17 bits denoted by R1, R2, R3 and R4

respectively. In fact these LFSR's are same as those of A5/2 and their feedback functions same as A5/2's feedback functions. (See Figure 3.3)

In this attack, the authors designed a clocking mechanism that reduces these security flaws. According to the A5/H, clocking unit takes inputs from all of the four registers. The clocking mechanism is as follows:

- Tap position control unit takes 14^{th} ,15^{th} and 16^{th} bits of R4 as input.
- This unit evaluates 16^{th} bit as most significant bit and 14^{th} bit as least significant bit and computes a result as "i" which is decimal value of this three bit sequence.
- This result determines which bits to be clocked as $C1_i$, $C2_i$ and $C3_i$.
- Clocking mechanism of each LFSR is determined by majority rule result of the clocking taps $C1_i$, $C2_i$ and $C3_i$ respectively.
- Ten the registers whose clocking tap bits agree with the majority are clocked as in the case of A5/1.
- After these clockings, R4 is clocked.

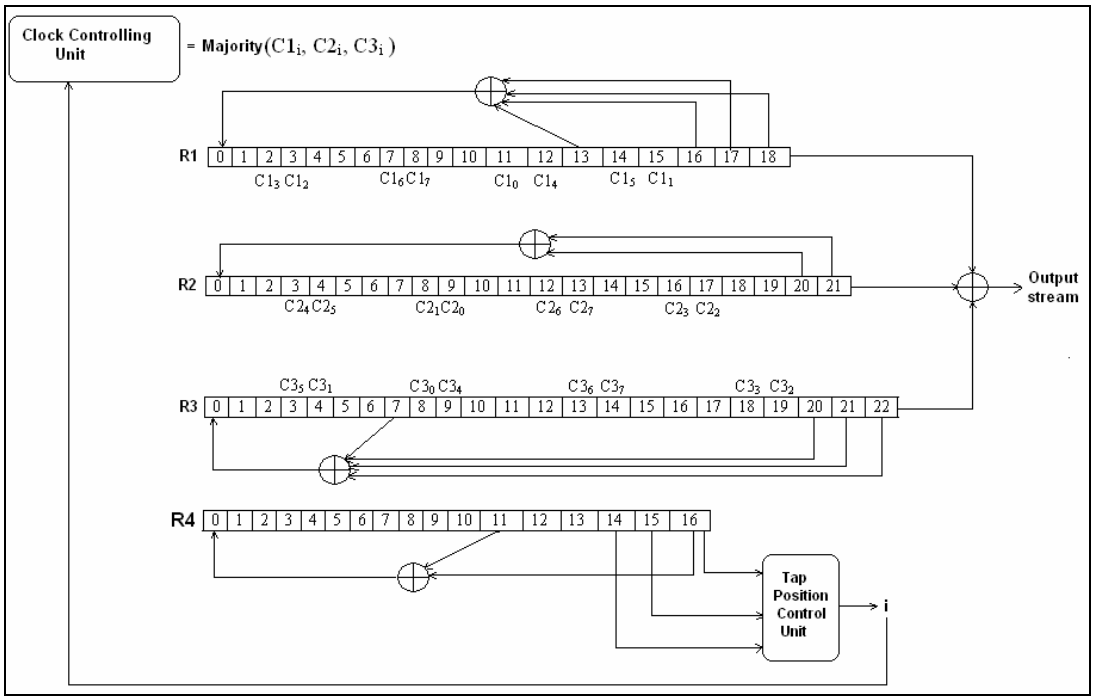

Figure 3.3. The proposed stream cipher model A5/H [24]

As an example, let the 14^{th} 15^{th} and 16^{th} bits of R4 be 0, 0 and 1 respectively. Then tap position control unit evaluates these bits as a “100” binary sequence and computes “4” as its decimal value. Then clocking unit takes inputs from $C1_4$, $C2_4$ and $C3_4$ as clocking tap bits (see Fig. 3). After majority of $C1_4$, $C2_4$ and $C3_4$ is calculated and two or three registers whose clocking tap value ($C1_4$, $C2_4$, $C3_4$) is the same as majority result are clocked. In fact clocking mechanism of A5/H is very similar to A5/1; Erguler and Anarim only change clocking tap positions of registers depending on R4. So it can be seen as a “clocking tap hopping”. The output is computed by XOR’ing of each LFSR’s last bit which is at position 18, 21 and 22 for R1, R2 and R3 respectively as in the case of A5/1 stream cipher.

A5/H algorithm also consists of three steps that are the initialization part, discarding first 100 produced bits part and producing 228 bits of key stream sequence part. In the initialization part, 64 bit secret session key K_c XOR'ed (ignoring majority rule) in parallel into the LSBs of the four registers which are all zeroed at the beginning of the process. Then 22 bit frame number F_n XOR'ed (ignoring majority rule) in parallel into the LSBs of the four registers. In second part A5/H is run for 100 clock cycles according to clocking mechanism, however produced 100 bits are discarded. Finally, in the last part A5/H is run according to clocking mechanism to produce 228 bits of key stream sequence [30].

4. A SURVEY OF PREVIOUS WORK AND ATTACKS

The previously proposed attacks on A5/1 stream ciphers can be classified in two groups, i.e., the known cipher text attacks and the known plain text attacks.

In known cipher text attacks the attacker is assumed to be given a sequence of frame counters $F_n \varepsilon \{0,1\}^{22}$ and a sequence of pairs of cipher text frames $C_n \varepsilon \{0,1\}^{114} \times \{0,1\}^{114}$ a known cipher text attack against A5/1 aims to recover information about either the plaintext $P_n \varepsilon \{0,1\}^{114} \times \{0,1\}^{114}$ or the session key K.

In known plaintext attacks, the attacker tries to determine from a set of frames of cipher text and the corresponding plaintexts a set of output bit streams for A5/1, and from these attempts to compute the state A5/1 before the A5/1 frame counter [29].

In this respect, the major known cipher text and plaintext attacks include the basic brute force, divide and conquer time/memory tradeoff, biased birthday, random subgraph and clock control guessing attacks.

4.1. Brute Force Attacks

The brute force attack iterates over possible keys and creates a set of pseudo-random sequences for each frame and computes the plaintext pairs for each cipher text pair.

If the sequence of computed plaintexts is all recognized as a valid plaintext, a candidate key has been found. The feasibility of a brute-force attack in practice is limited partly by the possibility of efficient plaintext recognition.

4.2. Divide-and-Conquer Attacks

The divide-and-conquer attack against A5/1 is a known plaintext attack and the objective of the attack is to recover the initial state of A5/1 corresponding to a 64-bit prefix of known A5/1 output.

The attack starts by generating 10 bits for each register. These 10 bits infer for each register 10 linear equations. The resulting 30 equations are clearly linearly independent as an equation references one more register bit than the previous equation. Solving these equations solves therefore 30 bits of the initial state. The bits in the registers define now a linear equation involving the first plaintext bit and the first bits of each register after the 100^{th} initial clocking. These bits define also a set of state changes. The amount of state changes defined depends on the exact value of the bits due to the clock control. Each state change implies a linear equation involving a known plaintext bit and the output bits in the three registers. Each such equation involves at least two new bits per state change and therefore the equations are linearly independent. The equations are linearly independent from the equations in the previous step if each equation contains a bit not present in any of the equations in the previous step. This is false only if R1 has been shifted in every state transition, which is highly improbable.

Assuming the bits are random, the probability that a register does not shift during a state change is ¼. Therefore the 10 bits in each register define an expected 40/3 state changes, generating an expected 40/3 linear equations linearly independent with all the previous equations.

At this stage, there are still approximately 63.32 – 30 – 1 – 13.30 ~ 19.02 bits whose values remain unsolved for the initial state. The initial state can be recovered by generating a tree of depth 19.02/3 where a node contains the next three input bits to the clock control function and has as children nodes that contain the next clock inputs. The expected number of children for each node is 2.5 [5]. This means that the amount of possible assignments to the bits not yet defined by the linear equations are with an expected value of approximately $2^{10.16}$. The expected number of trials to find a correct state S(101) which generates the 64

bits of plaintext in question is now on average $2^{40.16}$. The algorithm should generate only few (presumably one) candidate states for S(101) [5, 31].

4.3. Time-memory tradeoff attacks

The time-memory tradeoff attack is composed of a pre-computation task and an online attack.

In any time-memory tradeoff attack there are five key parameters. Let's define them as:

- N represents the size of the search space
- P represents the time required by the preprocessing phase of the attack
- M represents the amount of random access memory available to the attacker
- T represents the time required in the realtime phase of the attack
- D represents the amount of realtime data available to the attacker.

In the case of block ciphers, the size N of the search space is the number of possible keys. It is assumed that the number of possible plaintexts and cipher texts is also N, and that the given data is a single cipher text block produced from a fixed chosen plaintext block.

The best known time/memory tradeoff attack is due to Hellman [5]. It uses any combination of parameters, which satisfy the following relationships: $T*M^2 = N^2$, $P = N$, $D = 1$. The optimal choice of T and M depends on the relative cost of these computational resources. By choosing T = M, Hellman gets the particular tradeoff point $T = N^{2/3}$ and $M = N^{2/3}$.

Hellman's attack is applicable to any block cipher whose key to cipher text mapping (for a fixed plaintext) behaves as a random function f over a space of N points. If this function happens to be an invertible permutation, the tradeoff relation becomes $T*M = N$, which is even better. An interesting property of Hellman's attack is that even if the attacker

is given a large number D of chosen plaintext/cipher text pairs, it is not clear how to use them in order to improve the attack.

Stream ciphers have a very different behavior with respect to time/memory tradeoff attacks. The size N of the search space is determined by the number of internal states of the bit generator, which can be different from the number of keys. The real-time data typically consists of the first D pseudorandom bits produced by the generator, which are computed by XOR'ing a known plaintext header and the corresponding cipher text bits. The goal of the attacker is to find at least one of the actual states of the generator during the generation of this output, after which he can run the generator forwards an unlimited number of steps, produce all the later pseudorandom bits, and derive the rest of the plaintext. It should be noted that in this case there is no need to run the generator backwards or to find the original key, even though this is doable in many practical cases.

Golic [4] and Babbage [30] independently described the simplest time/memory tradeoff attacks on stream ciphers. They associate with each one of the N possible states of the generator with the string consisting of the first log(N) bits produced by the generator from that state. This mapping f(x) = y from states x to output prefixes y can be viewed as a random function over a common space of N points, which is easy to evaluate but hard to invert. The goal of the attacker is to invert it on some substring of the given output, in order to recover the corresponding internal state. The preprocessing phase of the attack picks M random x_i states, computes their corresponding y_i output prefixes, and stores all the $(x_i ; y_i)$ pairs in a random access memory, sorted into increasing order of y_i . The real-time phase of the attack is given a prefix of D + log(N) - 1 generated bits, and derives from it all the D possible windows $y_1 ; y_2 ; \dots ; y_D$ of log(N) consecutive bits (with overlaps). Then each y_j is looked up from the data in logarithmic time in the sorted table. If at least one y_j is found in the table, its corresponding x_j makes it possible to derive the rest of the plaintext by running the generator forwards from this known state 1.

The threshold of success for this attack can be derived from the birthday paradox, which states that two random subsets of a space with N points are likely to intersect when the product of their sizes exceeds N. If we ignore logarithmic factors, this condition becomes D*M = N where the preprocessing time is P = M and the attack time is T = D.

This represents one particular point on the time/memory tradeoff curve T*M = N. By ignoring some of the available data during the actual attack, we can reduce T from D towards 1, and thus generalize the tradeoff to T*M = N and P = M for any $1 \leq T \leq D$.

This T*M = N tradeoff is similar to Hellman's T*M = N tradeoff for random permutations and better than Hellman's $T*M^2 = N^2$ tradeoff for random functions (when T = M we get $T = M = N^{1/2}$ instead of $T = M = N^{2/3}$). However, this formal comparison is misleading since the two tradeoffs are completely different: they are applicable to different types of cryptosystems (stream vs. block ciphers), are valid in different parameter ranges ($1 \leq T \leq D$ vs. $1 \leq T \leq N$), and require different amounts of data (about D bits vs. a single chosen plaintext/cipher text pair) [31].

4.4. Biased birthday attacks

The biased birthday attack is first introduced by Biryukov, Shamir and Wagner in their pioneering work [7]. In this work, authors categorize the states through which the registers are running according to the prefixes of the output bits that they generate and denote states that produce output sequences starting with a certain prefix as red and the states generating output sequences that contain an occurrence of the same prefix in the output not at the beginning of the output but between bit positions 101 and 277 as green.

The red states are the states that is stored in the disk, looked for in the data, and tried to be collided by comparing their prefixes. The green states are all the states that could serve as initial states in frames that contain ff. Non-green initial states are of no interest to us, since the frames they generate from the actual data are discarded.

The size of red states is approximately 248, since there are 264 possible states, and the probability that the selected prefix occurs right at the beginning of the output sequence is 2^{-16}.

The crucial observation that makes the author’s biased birthday attack efficient is that in A5/1 there is a huge variance in the weights of the various red states. The authors run the

tree exploration algorithm on 100,000,000 random states and compute their weights. They find out that the weight of about 85 per cent of the states was zero, because their trees died out before reaching depth 100. Other weights range all the way from 1 to more than 26,000.

The intuition behind the biased time memory tradeoff attack is to store red states, but what the authors really try to achieve is to collide the green states in their belts (that are accessible from the red roots by an easy computation). Standard time memory tradeoff attacks store random red states, but each stored state increases the coverage by just 177 green states on average. With the optimized choice in the preprocessing stage, they increase each stored state's coverage by 12,500 green states on average, which improves the efficiency of the attack by almost two orders of magnitude [7].

4.5. Random subgraph attacks

The random subgraph attack is another time/memory tradeoff attack against A5/1. The attack relies on the special states defined above, but attempts to cover all special states in the pre-computation phase, and hence requiring fewer frames of known A5/1 output. The objective of the attack is to compute a set of possible internal states of A5/1 corresponding to the state of A5/1 at different points in the known output sequence.

The random sub-graph attack is based on the Hellman time-memory tradeoff. Let P be the set of plaintexts, K the set of keys and C the set of cipher texts. Let $E_k : P \rightarrow C$ be the encryption function for key $k \varepsilon K$. Let $K = P = C = U$.

The idea of the attack is to define a function $f(K) = E_K(P)$ and a set of t permutations $\pi_i : U \rightarrow U$. Define $f_i(r) = \pi_i(f(r))$. Compute m random values $r_n \varepsilon U$ the pairs $(r_n, fit(r_n))$ for all functions f_i.

Each pair $(r_n, fit(r_n))$ defines a sequence that covers 212 of elements in U. Choosing a sufficient amount of such sequences from start points to end points allows one to cover the set U with high probability.

The attack consists of iterating the functions f_i over the known plaintext p ε U and each iteration comparing the result with the end points in the computed table. If a collision is found, then the attack proceeds by iterating f_i from the start point corresponding to the determined end point until a collision is found with p. The key is with high likelihood the element in used as input to the last iteration of f_i [31].

4.6. Time/Memory/Data Tradeoff Attacks with Sampling

One practical problem with tradeoff attacks is that random access to a hard disk requires about 8 milliseconds, whereas a computational step on a fast PC requires less than 2 nanoseconds. This speed ratio of four million makes it crucial to minimize the number of disk operations the attacker performs, in addition to reducing the number of evaluations of f_i. An old idea due to Ron Rivest was to reduce the number of table lookups in Hellman's attack by defining a subset of special points whose names start with a fixed pattern such as k zero bits.

Special points are easy to generate and to recognize. During the preprocessing stage of Hellman's attack, the attacker starts each path from a randomly chosen point, and stops it only when it encounters another special point (or enter a loop, which is unlikely when $t^2 > N$). Consequently, it knows that the disk contains only special endpoints. If it chooses k = log(t), the expected length of each path remains t (with some variability), and the set of mt endpoints it stores in all the t tables contains a large fraction of the N=t possible special points.

The main advantage of this approach is that during the actual attack, you have to perform only one expensive disk operation per path (when we encounter the first special point on it). The number of evaluations of fi remains $T = t^2$, but the number of disk operations is reduced from t^2 to t, which makes a huge practical difference.

The question is whether you can use a similar sampling of special points in tradeoff attacks on stream ciphers? Consider first the case of the BG tradeoff with TM = N , P = M ,

and $1 < T < D$. You can say that an output prefix is special if it starts with a certain number of zero bits, and that a state of the stream cipher is special if it generates a special output prefix. You would like to store in the disk during preprocessing only special pairs of (state, output prefix). Unlike the case of Hellman's attack (where special states appeared on sufficiently long paths with reasonable probability, and acted as natural path terminators), in the BG attack you deal with degenerate paths of length 1 (from a state to its immediate output prefix), and thus you have to use trial and error in order to find special states.

Assume that the ratio between the number of special states and all the states is R, where $0 < R < 1$. Then to find the M special states you need would like to store during preprocessing, you have to try a much larger number $M = R$ of random states, which increases the preprocessing time from $P = M$ to $P = M = R$. The attack time reduces from $T = D$ to $T = DR$, since only the special points in the given data (which are very easy to spot) have to be looked up in the disk. To make it likely to have a collision between the M special states stored in the disk and the DR special states in the data, you have to apply the birthday paradox to the smaller set of NR special states to obtain $M \times D \times R = N \times R$. The invariant satisfied for all the possible values of R is thus $T \times P = M \times D = N$ for $1 < T < D$.

An interesting consequence of this tradeoff formula is that the sampling technique had turned the original BG time/memory tradeoff ($T \times M = N$) into two independent time/preprocessing ($T \times P = N$) and memory/data ($M \times D = N$) tradeoffs, which are controlled by the three parameters m, t, and R. For $N = 2^{100}$ the first condition is easy to satisfy, since both the preprocessing time P and the actual time T can be chosen as 250. However, the second condition is completely unrealistic, since neither the memory M nor the data D can exceed 240. You now describe the effect of this sampling technique on the new tradeoff $TM^2D^2 = N^2$ described in the previous subsection. The main difference between Hellman's original attack on block ciphers and the modified attack on stream ciphers is that the attacker uses a smaller number $t = D$ of tables, and force T to satisfy $T = D^2$. Unlike the case of the BG attack, the preprocessing complexity remains unchanged as $N = D$, since you do not need any trial and error to pick up the random start points, and simply wait for the special endpoints to occur randomly during the path evaluation. The total memory required to store the special points remains unchanged at $M = m\ t = D$. The total time T consists of t^2 evaluations of the f_i functions but only t disk operations. One can

thus conclude that the resultant time/memory/data tradeoff remains unchanged as $TM^2D^2 = N^2$ for $T - D^2$, but you gain by reducing the number of expensive disk operations by a factor of t. Rivest's sampling idea thus has no asymptotic effect on “Hellman-like” tradeoff curves for block and stream ciphers, but drastically changes the BG tradeoff curve for stream ciphers [34].

4.7. Clock control guessing attacks

The clock control guessing attack stems from guessing the behavior of the clock control for each output bit and deriving the linear equations and checking for consistency. In respective papers, the authors first point out that the A5/1 generator produces one output bit per master clock cycle, and that there are four different behaviors of the clock control. Then they note that equivalent linear equations are easily constructed given that the clocking resides on the majority function.

In connection with clock-controlled bit stream generators, the linear consistency test technique may be used in a slightly different way, yielding a very simple method of proving an upper bound on the running time. Clock control generators have the following properties:

- The output bit depends on the inner state of the generator in some linear way. For each clock cycle t and each assignment to the output bit y_t , a linear equation q can be given such that the inner state S(t) generates output bit yt iff S(t) is a solution to q.
- The behavior of the clock control depends on the inner state of the generator in some linear way. For each clock cycle t and each assignment to the clock control behavior ξ_t, a set Q of linear equations can be given such that the inner state S(t) generates the clock control value ξ_t iff S(t) is a solution to Q.
- The number of possible behaviors of the internal clock is small.

The attack runs as follows: Given a generator that has these properties, the adaptive bit guessing attack can be modified as follows. Instead of guessing individual bits, for each clock cycle t = 0; 1; …, the associated clocking ξ_t is guessed. All linear equations that

follow from output bit y_t and clock control ξ_t are added to the linear equation system and checked for consistency.

4.8. Instant Cipher text-only Cryptanalysis Attacks

In [32] Barkan, Biham and Keller introduce how to mount a cipher text-only attack on an earlier version of A5/3, namelt on A5/2. Their attack requires a few dozen milliseconds of encrypted data, and its time complexity is about 2^{16} dot products. In simulations the attack finds the key in less than a second on a personal computer. They show that the attack they propose on A5/2 can be leveraged to mount an active attack even on GSM
networks that use A5/1, A5/3, or GPRS networks, thus, realizing a real-time
active attack on GSM networks, without any prior required knowledge. The full
attack is composed of three main steps:

- The first step is a very efficient known plaintext attack on A5/2 that recovers the initial key. This first attack is algebraic in nature. It takes advantage of the low algebraic order of the A5/2 output function. The authors represent the output of A5/2 as a quadratic multivariate function in the initial state of the registers. Then, they construct an overdefined system of quadratic equations that expresses the key-stream generation process and we solve the equations.
- The second step is improving the known plaintext attack to a cipher text- only attack on A5/2. The authors observe that GSM employs Error-Correction codes before encryption. They show how to use this observation to adapt the attack to a cipher text-only attack on A5/2.
- The third step is leveraging of an attack on A5/2 to an active attack on A5/1, A5/3, or GPRS-enabled GSM networks.

Barkan, Biham and Keller point out that due to the GSM security modules interface design, the key that is used in A5/2 is the same as in A5/1, A5/3, and GPRS. They show how to mount an active attack on any GSM network and show how to mount a passive cipher text-only attack on networks that employ A5/1. It is basically a time/memory/data

tradeoff attack. They claim that the same attack on A5/1 can be similarly leveraged to active attacks on the protocols of GSM, but the complexity is higher than the A5/2 case.

4.9. Distinguishing Attacks

A distinguishing attack on a cipher relates to the formal model of security, where an adversary can distinguish between the output of a particular cipher and the output of a truly random process, with a non-negligible probability. If attackers cannot make this distinction, an algorithmically derived stream cipher will look to them like a Vernam Cipher (often called a One Time Pad), and will be information-theoretically secure. Of course it is true that there is always a distinguishing attack against any algorithmic cipher; since it must have a finite key, and so brute-force key enumeration will yield a distinguishing attack of complexity 2^{k-1} where k is the key length.

In the case of block ciphers, being able to identify some distinguishing characteristic of the output might lead to an attack that reveals information about the key of the cipher. For example, Differential Cryptanalysis [33] of DES proceeds by identifying a distinguishing characteristic of the first rounds of DES, and then uses that characteristic to verify guesses about the key bits.

In [34] the authors draw an informal distinction between those attacks that yield useful cryptanalytic information such as information about the key or unknown key stream, and attacks that do not yield such useful information. They call these powerful and weak distinguishing attacks respectively.

4.9.1. Distinguishing Attacks against Stream Ciphers

In the context of the stream cipher attack on RC4, and of published distinguishing attacks on other ciphers such as SNOW and SOBER [6,7], these distinguishing attacks do not yield any usable information about the state of the cipher generator, and cannot be used to attack the generator itself. RC4 is still considered perfectly adequate for encryption in SSL and TLS, with 128-bit keys, despite the fairly powerful distinguishing attack

mentioned above. For another example, given 295 words of known key-stream output from SNOW, and a large amount of cipher text encrypted with the sub-sequent output from the same generated key stream, there is still no known or hypothesized attack that would reveal any useful information at all about the corresponding unknown plaintext. Neither the content of the cipher text, state of the generator, nor the input key, is compromised by these attacks.

One way of looking at this is that 295 words of known plaintext was required to re-cover one bit of information about the cipher text.

There is no clear relationship between the key length and the amount of generated keystream for which such a weak distinguishing attack would justify calling a stream cipher "broken". This decision depends more on the amount of data to be encrypted than upon the key length. Conversely, the strong distinguishing attack corresponding to key enumeration requires no known plaintext at all, and only enough cipher text to exceed the unicity distance of the input language and key size.

4.10. Correlation Attacks

The original correlation attack was proposed by Siegenthaler [35]. There, the attack was described as a cipher text only attack. Since a cipher text only attack requires that there exists redundancy in the plaintext message, one usually applies correlation attacks in known plaintext scenarios, where there is no requirement of redundancy in the message.

Assume that we have observed a keystream sequence, z, of length N , $z = z_1, z_2, \ldots, z_N$. The keystream sequence is generated from a generator with n different LFSRs. If one can find a correlation between the output of one of the shift registers, called the target LFSR, and the keystream, i.e., $P(u_i = z_i) \neq 0{:}5$, where u_i is the output of the LFSR and z_i is the known keystream symbol, then one can try to find the initial state in a “divide-and-conquer” type of attack on the target LFSR. Furthermore, we assume that the feedback polynomial g(x) of the target LFSR is known. There is no requirement of any further

structure in the generator. The only thing that matters is the fact that we can find a correlation.

In the previous section several standard methodologies for destroying the linear properties of LFSR sequences and produce a keystream sequence with a large linear complexity were given. One of these methodologies is the nonlinear combination generator. It is worth noticing that for a nonlinear combination generator there always exists a correlation between the generator output z_i and either one or a set of M LFSR output symbols. It is well known that if f is an (M - 1)-resilient (but not M-resilient) function then there exists a correlation, which can be expressed in the form

$$P(z_i = u_i^{(i_1)} + u_i^{(i_2)} + ... + u_i^{(i_M)}) \neq 0.5. \quad (4.1)$$

It is also known that there is a tradeoff between the resiliency and the non-linearity of f, and hence M must be rather small [36]. Returning to the previously mentioned correlation attacks, the above overview demonstrates that finding a low complexity algorithm that successfully uses the existing correlation in order to determine a part of the secret key can be a very efficient way of attacking such stream ciphers.

The principle of the original attack proposed by Siegenthaler is the following. Assume that the LFSR we consider has length l and that the correlation between the keystream sequence and the LFSR sequence is 1 - p, where $p < 0.5$. Here we also assume that the feedback polynomials of the LFSRs are known. This is different from the setting in [37] were an additional exhaustive search was made over all primitive polynomials of degree l.

For a binary LFSR of length l there are 2^{l} possible initial states. For each possible initial state $u_0 = (u_1 ; u_2 ; \ldots; u_l)$ the LFSR output sequence $u = (u_1 ; u_2 ; \ldots ; u_N)$ is generated. Let be defined as = N d_H (u; z), where d_H (u; z) is the Hamming distance between u and z, i.e., the number of positions in which u and z differ. There are two hypothesis to be considered: H_1: The guessed initial state is correct or H_0 : The guessed initial state is wrong.

Under these two hypothesis, β is binomially distributed with mean value. Hence, we see that if we run through all possible initial states, and if N is large enough, will with high probability take its largest value when u_0 is the correct initial state.

If a correlation can be found for each of the LFSRs in the generator, the complexity of finding the correct key is reduced.

4.11. Improved Correlation Attacks

4.11.1. Maximov, Johansson and Babbage Attack

In [37] Maximov, Johansson and Babbage introduce an improved version of the correlation attack. They consider the Ekdahl-Johansson attack as the basis, and apply several new improvements. As the result, the new attack now needs only less then 1 minute of computations, and a few seconds of known conversation. It does not need any notable pre-computation time, and needs reasonable space of operation memory.

For the case of a cipher text-only attack on A5/1, the authors use the fact that some redundancy is part of the plaintext. There are at least two kinds of redundancy that are explicit and may be used in an attack where only cipher text is available. The first kind is the fact that coding is done before encryption, which results in linear relationships in the plaintext since the parity check symbols are also encrypted.

The second kind of redundancy is the fact that during silence, a special frame including a large number of zeros is sent. Silence occurs very often, but unfortunately these frames used for silence are transmitted less frequently, one to initialize a period of silence and then two each second. The proposed attack can be considered in a cipher text-only scenario, in which case we use this redundancy during silence to get some known outputs from the cipher. Although several of the previous attacks are sufficient to break A5/1 in a known plaintext attack, the authors claim that further progress is very important.

4.11.2. Ekdahl and Johansson Attack

This attack was proposed in 2002 by Ekdahl and Johansson. In [38], the authors propose a new attack on the A5/1 stream cipher, based on an identified correlation. In contrast to previous attacks, theirs is not a time-memory trade-off attack, but uses completely different properties of the cipher. It explores the weak key initialization, which allows separating the session key from the frame number in binary linear expressions.

The complexity of the attack is only linear in the length of the shift registers and depends instead on the number of irregular clockings before the keystream is produced. The implemented attack needs the 40 first bits from about 2^{16} (possible non-consecutive) frames, which corresponds to about five minutes of GSM conversation. Their implementation of the attack shows that they have a high success rate; more than 70 per cent. This can be improved by using larger list size and/or larger interval size. The complexity of the attack using the parameters presented is quite low and the attack can be carried out on a modern PC in less than five minutes using very little precomputation time and memory. The complexity of the attack presented in this paper is roughly linear in the shift register lengths. The authors also claim that the previous attacks also need either much precomputation and/or memory or they have a high time complexity.

4.12. Inversion Attack

The inversion attack is a known plaintext attack on some particular filter generators. It was proposed by Golic in 1996. A generalization to any filter generator, called generalized inversion attack, was presented by Golic, Clark and Dawson in 2000. Both inversion attack and generalized inversion attack aim at recovering the initial state of the linear feedback shift register (LFSR) from a segment of the running-key when the LFSR feedback polynomial, the tapping sequence and the filtering function are known. Original inversion attack. The original inversion attack only applies when the filtering function f is linear in its first input variable (forward attack) or in its last input variable (backward attack), i.e., when

$$f(x_1 ; x_2 ;\ldots; x_n) = x_1 + g(x_2 ;\ldots; x_n) \quad (4.2)$$

or

$$f(x_1 ; x_2 ;\ldots; x_n) = g(x_1 ;\ldots; x_{n-1}) + x_n \quad (4.3)$$

where g is a Boolean function of n-1 variables. In the first case, the key stream s is defined as

$$s_t = f(u_{t+}\gamma_1; u_{t+}\gamma_2; \ldots; u_{t+}\gamma_n) \quad (4.4)$$

$$= u_{t+}\gamma_n + g(u_{t+}\gamma_2; \ldots ; u_{t+}\gamma_n) \quad (4.5)$$

where $(u_t)_{t>0}$ is the sequence generated by the LFSR and $(\gamma_i)_{1<i<n}$ is a decreasing sequence of non-negative integers. The attack relies on the fact that bit $u_t+\gamma_1$ can be deduced from the $(\gamma_1 - \gamma_n)$ previous terms, $(u_{t+}\gamma_1; : : : ; u_{t+}\gamma_n)$ if the running-key bit s t is known. The relevant parameter of the attack is then the memory size of the filter generator, defined by $M = \gamma_1 - \gamma_n$. Indeed, the complete initialization of the LFSR can be recovered by an exhaustive search on only M bits.

The backward attack, which applies when the filter function is linear in its last variable, is similar. The complexity of both forward and backward attacks is $O(L2^M)$. It follows that the memory size of a filter generator should be large and preferably close to its maximum possible value L - 1.

Moreover, the complexity of the attack dramatically decreases when the greatest common divisor of all spacings between the taps, $d = \gcd(\gamma_i - \gamma_{i+1})$, is large. Indeed, the inversion attack can be applied to the d-decimation of the LFSR sequence, i.e., to the sequence obtained by sampling the LFSR sequence at intervals of d clock cycles.
Therefore, the effective memory size of the filter generator corresponds to

$$M' = \frac{\gamma_1 - \gamma_n}{\gcd(\gamma_i - \gamma_{i+1})} \quad (4.6)$$

The related design criterion is then that the greatest common divisor of all spacings between the taps should be equal to 1.

5. INTRODUCTION OF THE "PECULIAR EVENT" CONCEPT

The "peculiar event" concept is essentially a derivation from Biham and Dunkelman's attack presented in [8]. As stated in their pioneering paper, the main idea behind their attack is to wait until an event that leaks a large amount of data occurs and then exploit it. They assume that for 10 consecutive rounds the third register R_3 is not clocked. In such a case, they gain about 31 bits of information about the registers. In this paper, we will call derivation of similar seldom but important in cryptanalytic terms events as "peculiar events".

In their work, Biham and Dunkelman start their attack by guessing the clock controlling and the output bit of the third register. So they find out the ten bits following the clock bit in the other two registers as they have to be equal to the complement of the third register's clock control bit. As the output stream is assumed to be known, by estimating the output bid of the third register, they compute the other two registers output bits XOR'ed. As there are at least 11 rounds for which the same output bit is used from R_3, eleven linear equations can be gained about the registers.

By guessing nine bits from R_1 (R_1[18; 17; 16; 15; 14; 12; 11; 10; 9]) and another one from R_2(R_2[0]), they uncover all of R_1 and R_2. This means that just by the cost of guessing twelve bits (R_3[10], R_3[22], R_2[0], and 9 bits of R_1), and they recover all the bits of R_1 and R_2 . By waiting until R_3 moves, and they gain another bit (R_3 [21]). By even further guessing the bits R_3[0]; … ; R_3[9] they continue clocking the third register and receive all the unknown bits of R_3 (total of 11 unknown bits). The workload of this stage is the cost of guessing 10 bits, multiplied by the cost of work needed to retrieve the unknown bits for each guess. Since each time the register R_3 advances they recover a new bit of R_3 , they need to wait for 12 clockings of R_3 in order to retrieve the full register, and since the probability that the register R_3 advances is 3/4, it is expected that 16 clocks suffice to retrieve the whole register. Now they have a possibility for the full internal state and need to check for its correctness. The amortized cost for checking whether this is the right state is two clock cycles for each guess (all cases need to be clocked at least once, half of the

cases should be clocked another time, 1/4 a third time, etc.). Thus, this stage requires $2^{10} \cdot 2^4 \cdot 2 = 2^{15}$ workload.

The total expected running time of this attack is $2^{12} \cdot 2^{15} = 2^{27}$, given the location where R_3 is static. This location is of course unknown, thus, they need to examine about 2^{20} possible starting locations (from many possible frames). Hence, their basic attack requires about 2^{47} running time and about 2^{20} bits of the output stream.

Although Biham and Dunkelman's application of a peculiar event is a refined method, the attack has no practical applicability as the probability that R_3 will not be clocked for ten consecutive clocks is 2^{-10}. This means that in a known plaintext attacks with a conversation of one-to-two minutes, the probability that such a peculiar event to occur is only < 2 per cent and since the key space is not fully covered, the probability of success is extremely low.

In this work, we define another peculiar state, which significantly reduces the time complexity of the attack by more effectively utilizing the full memory available to the attacker.

6. IMPROVEMENT TO HELLMAN'S TIME/MEMORY ATTACK

6.1. Previous Improvements to Hellman's TMTO Method

As stated previously in detail, Hellman introduced in [5] for a cryptosystem having N keys the time-memory-tradeoff method that recovers the secret key in $N^{2/3}$ operations using $N^{2/3}$ words of memory. The typical application of this method is the recovery of a key when the plaintext and the cipher text are known.

Multiple improvements have been proposed to this method until now. The most noteworthy of these includes Rivest's suggestion about using distinguished points (Distinguished points are points for which a simple criterion holds true (e.g. the first ten bits of a key are zero)) as endpoints for the chains. When given a first cipher text, the attacker stores in the pre-computation stage only distinguished points in the memory and generates a chain of keys until he finds a distinguished point and only then looks it up in the memory. This greatly reduces the number of memory lookups.

Other improvement ideas use similar optimizations, i.e. optimization of table parameters and minimizing total cost of the method based on the costs of memory and of processing engines are discussed in varius works. Actually, Kim and Matsumoto show that the parameters of the tables can be adjusted such as to increase the probability of success, without increasing the need for memory or the cryptanalysis time [40]. This is actually a trade-off between pre-computation time and success rate. However, the success rate is of course bound upwards.

The major deficiency of Hellman's time-memory tradeoff is that "overlap situations" frequently occur. The overlap situations are overlaps in the Hellman chains that substantially decrease the coverage of pre-computed states in the memory over the state-space. The different cases leading to overlaps can be summarized as follows:

In the tradeoff method described, one tries to store information about as many different keys as possible by taking as long chains as possible. Consequently, the very

short chains increase the memory complexity of the tradeoff and could be rejected. However, different critical overlap situations can appear and add constraints to the tradeoff, namely:

- A chain can cycle. This is the case where i and j with $i \neq j$ and $K_{i+1} = K_{j+1}$.
- Two chains computed with the same mask function can merge. This is the case where two different start points have the same image. This means that from the moment of the merge until the end of at least one chain, both chains contain the same keys.
- A chain can collide with another chain computed with a different mask function. This is the situation where two chains computed with different mask functions have some common points between the start and end points. It means that some keys are stored several times, which is not efficient.

The attacker can deal with cycles by choosing an adequate maximum chain length t and the mergers can be rejected after the pre-computation by choosing the longest of two merging chains. Still further performance improvements can be made through selection of efficient mask functions.

In most of the TMTO attacks, authors suggest permutations as mask functions. In [41], Xavier and Gael propose an efficient mask functions in terms of collisions and implementation cost. They argue that the problem when using a permutation is the possibility that two mask functions have some common keys after a collision. For example, if the mask function corresponds to a permutation of two bits, the common length after a collision is obviously ½ + ¼ + …

Moreover, if a large number of mask functions is needed (which is practically probable), their implementation becomes fastidious (especially in hardware designs that are particularly efficient in time-memory tradeoffs). They suggest the following definition of efficient mask functions. Let $m_i(x)$ and $m_j(x)$ be two mask functions:

$$M_i(x): \{0,1\}^n \rightarrow \{0,1\}^k \quad (5.1)$$

$$m_j(x): \{0,1\}^n \rightarrow \{0,1\}^k \quad (5.2)$$

Let $n(x) = m_i(x) - m_j(x)$. One can define efficient mask functions such that for every $i \neq j$, you have:

$$Ker(n(x)) = 0 \tag{5.3}$$

where the *Ker* function is defined as: [20]

$$Ker(n(x)) = \{x \varepsilon \{0,1\}^n \mid n(x) = 0\} \tag{5.4}$$

Then, a possible definition for an efficient mask function is

$$m_i(x): \{0,1\}^n \rightarrow \{0,1\}^k: m_i(x) = XOR\ (x, i) \tag{5.5}$$

We will also use this mask function in our work.

6.2. Oechslin's proposal

A last improvement comes from Oechslin's work on effective table designs. The original idea of Hellman's attack was using a pre-computed table stored in memory. By defining as M is required memory, T is the operations in time and N is possible solutions to search over, he showed that if the equation $mt^2 = N$ is satisfied, attacks where $M = mt$ and $T = t^2$ are possible.

In his attack, Hellman chooses a large number of random *m* start points from *N*, iterate *f* on each one of them *t* times and store the first and last elements. The reason of storing only the first and last elements of the iterative chains is to save memory as by using the first key of the chain the whole chain can be regenerated. The realization of the Hellman's attack is done by searching a match between end points of the pre-computed table and the given $f(K)$ for some unknown *K*. It can be obtained by repeatedly applying $f(\ )$ in the easy forward direction until a match occurs with a stored end point and going to the corresponding start point and continuing to iterate *f* from this starting point. The last point before the match $f(K)$ is likely to be the desired key *K*.

In fact, finding a matching endpoint does not guarantee the key to be in the table. Because the key may be part of a chain that has the same endpoint, it may not be in the table. Since a single table with m starting points and t iterations satisfying $mt^2 = N$, covers only a fraction of $mt / N = 1/t$ of the solution space, Hellman offered to use t unrelated tables, each of which has m starting points iterated t times, to cover the whole space. The difference of each table stems from using different reduction R functions, so each table uses variants of $f_i( )$, as permuting the output bits of f. In other words, for each table the bits of $f( )$ is reordered by corresponding R function so variants of $f( )$ function are generated. Whole pre-computational table system is shown in Figure 2. Since there are t tables and each table has m rows with t operations Hellman's method requires $M = mt$ as memory and $T = t^2$ as computational time [10].

In [9], Oechslin argues that the main limitation of the original scheme is the fact that when two chains collide in a single table they merge. Thus, he proposes a new type of chains that can collide within the same table without merging. These chains use a successive reduction function for each point in the chain. They start with reduction function 1 and end with reduction function t−1. Thus if two chains collide, they merge only if the collision appears at the same position in both chains. If the collision does not appear at the same position, both chains will continue with a different reduction function and will thus not merge. For chains of length t, if a collision occurs, the chance of it being a merge is thus only 1/t. The probability of success within a single table of size m*t is there claimed to be:

$$P_{table} = 1 - \prod_{i=1}^{t} (1 - \frac{m_i}{N}) \tag{5.7}$$

$$\text{where} \quad m_1 = m \quad \text{and} \quad m_{n+1} = N\left(1 - e^{-\frac{m_n}{N}}\right) \tag{5.8}$$

For derivation of the success probability, see [9].

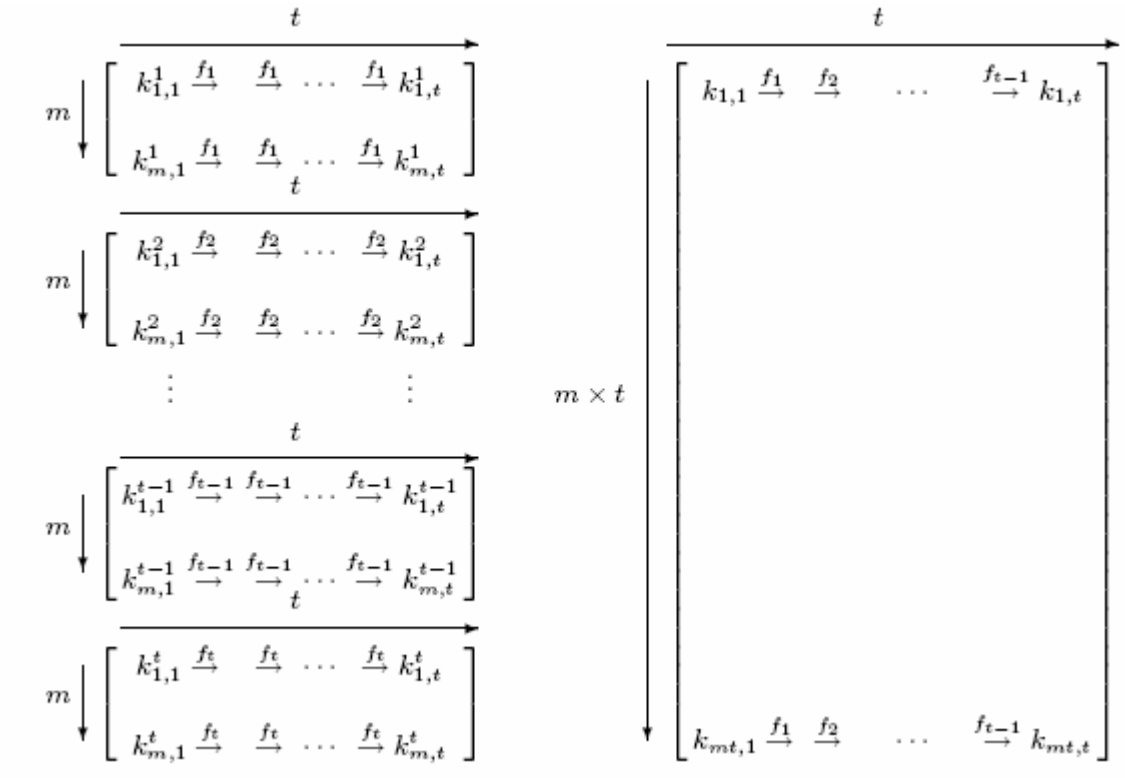

Figure 6.1. Classic vs. new setup of the key tables

If the success probability of rainbow tables are compared to that of classical tables, the success probability of t classical tables of size m×t comes out to be approximately equal to that of a single rainbow table of size mt × t. In both cases the tables cover mt^2 keys with t different reduction functions.

For each point a collision within a set of mt keys (a single classical table or a column in the rainbow table) results in a merge, whereas collisions with the remaining keys are not merges.

To lookup a key in the new tables, Oechslin suggest to proceed in the following manner: First apply R_{n-1} to the cipher text and look up the result in the endpoints of the table. If you find the endpoint you know how to rebuild the chain using the corresponding starting point. If you don't find the endpoint, you try if you find it by applying R_{n-2}, f_{n-1} to see if the key was in the second last column of the table. Then try to apply R_{n-3}, f_{n-2}, f_{n-1}, and so forth. The total number of calculations you have to make is thus $t(t-1)^2$. This is half as much as with the classical method. Indeed, one needs t^2 calculations to search the corresponding t tables of size m × t.

Oechslin claims that these new chains have four main advantages:

- The number of table look-ups is reduced by a factor of t compared to Hellman's original method.
- Merges of these chains result in identical endpoints and are thus detectable, as with distinguished points. They can thus be used to generate merge-free tables.
- They have no loops, since each reduction function appears only once. This is better than loop detection and rejection as described before, because we don't spend time on following and then rejecting loops and the coverage of our chains is not reduced because of loops than can not be covered.
- They have a constant length whereas chains ending in distinguished points have a variable length that reduces the number of false alarms.

Due to their simplicity, robustness and superior performance, we also will be using "rainbow chains" in our proposed attack.

7. EMBEDDING PECULIAR EVENTS IN TMTO ATTACK

The main idea of our attack is to embed the peculiar states concept in Hellman's time/memory attack. More specifically, assume that for six consecutive rounds the third register is not clocked. In this case, the attacker can calculate the 64-bit internal state of the three shift register by storing 2^{34} distinct 74-bits input/output pairs.

The attack runs as follows. Assume that the attacker knows the clock controlling bit and the output bit of the third register. Under the assumption that the third register is not clocked for six consecutive rounds, this enables the attacker to calculate for these six rounds what the clock controlling bits in the other two registers should be, namely the complement of the controlling bit of the third register. So the attacker immediately gets access to 12 bits in the first and second registers. Knowing that the third register delivers the same output bit for six consecutive rounds, the attacker can now guess the last six bits of the first register are and then calculate the corresponding bits in the second register. (by reversing the XOR operation)

The complexity so far is the cost of guessing the last six bits of the first register (i.e. $R_1[18,17,16,15,14,13]$), the clock controlling bit of the third register (i.e. $R_3[10]$) and the last bit of the third register (i.e. $R_3[22]$), i.e. in total eight bits. Actually, by guessing the remaining 7 bits of the first and 10 bits of the second register, the attacker can recover the whole content of the first and second registers. This increases the complexity of the attack to 8+7+10=25 bits. Additionally, the third register needs to be recovered.

To recover the remaining of the third register, the attacker needs to calculate and/or guess the missing 21 bits (the 10^{th} and 22^{nd} bits have already been determined in the first step to recover the clock controlling and output bits of the first and second registers). To do so, assuming that the attacker has full knowledge about of the output bits (which is a common assumption attacks of similar type) the attacker needs actually to guess only the first 10 bits (i.e. $R_3[0,1,2,3,4,5,6,7,8,9]$) and the 12^{th} bit (i.e. $R_3[11]$) of the third register as the other bits can be calculated by complementing the XOR of the already calculated bits in the first and second registers. The reason why these 12 bits of the third registers must be

guessed but cannot be calculated is that the attacker doesn't know whether the third register is being clocked during the next eleven clocking rounds or not. It is of course possible that the third register does not clock for some of the next ten clocking as well, which would leave the clocking bits for consecutive clocking rounds undetermined, driving the success rates down.

Alternatively, a more robust way would be to guess the R_3[21,20] bits instead of R_3[11] as this would enable the attacker to calculate the clocking bits for app. one more round. Given that the probability of a register advancing at any clocking round is 3/4, gaining knowledge about 13 consecutive bits enables the attacker to calculate 9.75 more bits. This is just sufficient to recover the missing nine bits, i.e. R3[19,18,17,16,15,14,13, 12,11].

The memory complexity of the attack so far (which would have relied solely on peculiar states) is 8+7+10+10+2=37 bits. As the attacker needs to store the first 64 output bits corresponding to each bundle of 37 guessed bits, the total memory complexity for each input-output pair would have been 37+64=101 bits. Covering the whole key space thus would have required 2^{37}*101 bits = $2^{10.76}$ GB = $10^{3.24}$ GB of memory. However, this is not possible through current commercially available computers. As current commercially available storage devices are limited by 2^7 GB, an alternative method needs to be developed to decrease the memory requirement by more than eight times.

At this stage, we think that further clarification on our attack's steps up to this point may be useful. Namely, it may sound contradictory that we assume the output bits given to recover the content of the first and second registers in the first part of our attack while this time calculating the same output bits once we have recovered the content of the third register at the second step.

Actually, there is no contradiction. In the first part, we assign arbitrary bits to the first 6+10=16 bits to the first output bits. We use these assigned random bits to calculate the content of the registers. Then, in the second part of our attack, we take the first 16 bits of the output as given and calculate just the next 21 bits of the output by running the A5/1 algorithm with the computed register contents plugged in as the initial configuration.

Our suggested alternative method is to embed this attack into an improved version of Hellman's time/memory tradeoff. More specifically, by forming a chain of 2^3 encryptions using a chosen plaintext and 2^3 different 37-bit guessed bits bundles, we propose to calculate Hellman's pre-computation. The most important aspect of this step would be to choose two appropriate mask functions that would enable the attacker first to derive the 64-bit key from the 228-bit long output and then the 37-bit guessed bits bundle from this 64-bit key. As the probability of chains colliding or merging into each other is higher in this case compared to previous Hellman time/memory tradeoff attacks, using permutations of two bits as mask functions is not recommended (the common length after a collision is namely 1/2+1/4+1/8+…, which would require two 37 and 64-bit permutation calculations, which is not feasible.) Thus, following Xavier and Gael's logic [20] employing the XOR operator as explained in the previous sections is more feasible.

The combined function of the suggested two mask functions would work as the new *f*( ) function linking the different column of the matrix in the improved Hellman time/memory tradeoff attack to each other.

At this point, the attacker stores $2^{37} / 2^3 = 2^{34}$ guessed bit bundles in the memory each of which belong to a different peculiar state. Than 2^3 coupled two mask functions are run where the input is the 228-bit output while the output is the 37-bit guess bundle. A 2^{34} x 74-bit table (through the use of the two coupled mask functions it's sufficient to store the 37-bit guess bundle instead of its 64-bit output) is calculated this way that has a memory complexity of only $2^{40.21}$-bits (i.e. <160 GB), which perfectly fits into a single commercially available storage unit.

At real-time, the attack is then run as follows. Since a 228 bit key stream sequence provides 165 successive 64-bit blocks, initially these 64 bit blocks are taken. Next, the inverse function of the second mask function is applied to each 64 block. Next, a search between this result and last column of the stored table is performed. If a match occurs then the corresponding stored 37-bit guess bundle is found from the table. Afterwards, by repeatedly applying *f( )'s,* the "right" internal guess bundle is determined. If a match does not occur, then inverse of the last two mask function couples is applied to the 64 bit blocks.

The result is then searched as in the previous case. If a match doesn't occur the inverse of the last three mask functions is applied to the 64-bit blocks, etc. This process is repeated at most 2^3 times with each 64 bit block until a match occurs. Once the guessed bits and thus the internal state of the registers are recovered, the retrieval of the key can be performed as suggested by Biryukov in [7]. Finally, the other 228 bit key stream sequences can be generated with the obtained key sequence to assure the result is true.

8. INVESTIGATING THE PERFORMANCE OF THE PROPOSED ATTACK

We will start investigating the performance of the new attack by calculating its pre-computation complexity. As stated in the previous section(s), the attacker needs to guess 25 bits in the first and second registers to calculate the missing bits in the third register. Again, following the logic in the previous section, the attacker needs to advance the third register for app. 10 clockings to fully recover the content of the third register besides the 12 bits to be guessed except the two previously guessed bits used to recover the six consecutive bits of the first and second registers following the respective clocking bits of the two registers. Given that the cost of checking the calculated bits for correctness is two clock cycles (all cases need to be clocked at least once, half of the cases should be clocked another time, ¼ a third time, etc. [8]), the total computational complexity of recovering the third register comes out to be equal to 2^{12} (guess twelve bits for R_3) * 10 (wait for 10 clockings) * 2 (calculate the state) = $2^{16.33}$ clockings.

Then, the total pre-computation complexity of the attack is equal to $2^{25} * 2^{16.33} = 2^{41.33}$ clockings.

The memory requirement M of the new attack is as shown in the previous section the storage of a 2^{34} x 74-bit table that amounts to $2^{40.21}$-bits (i.e. <160 GB).

The computational time requirement of the new attack is given by $\frac{t(t-1)}{2}$ where t is equal to 2^{13}, as to lookup a key the attacker has to apply the inverse of the first mask function and search for a match. If no match is found, apply the second mask function and look for a match, etc. Thus, the real- time complexity T of the attack comes out to be $2^{25.00}$ unit computation durations.

Given the relation that the data requirement D can be expressed as $D^2=N^2/M^2T$, where $M^2=2^{80.42}$, $T=2^{25}$ and $N^2=2^{128}$, $D=2^{11.29}$ clockings. As we have assumed the third

register not to clock for six consecutive rounds, we have to multiply this by 2^6 to achieve a similar same success rate. Then, the data requirement comes out to be 3.23 seconds.

The following table compares our results with those of previous attacks.

Table 8.1. Comparison of our attack's performance with that of previous attacks

Attack	**Precomputation Complexity**	**Computational Time**	**Required Known Conversation**	**Required Memory**
Golic Time-Memory Tradeoff	$2^{35.65}$	$2^{27.67}$	160.78 minutes	862 GB
Biased-Birthday Attack (1)	2^{42}	n/a	2 minutes	292 GB
Biased-Birthday Attack (2)	2^{48}	n/a	2 minutes	146 GB
Random Subgraph Attack	2^{48}	n/a	2 seconds	292 GB
Basic Attack	2^{38}	$2^{39.91}$	2.36 minutes	64 GB
Erguler-Anarim Attack	2^{45}	2^{39}	14.6 seconds	256 GB
Our Proposed Attack	$\mathbf{2^{41.33}}$	$\mathbf{2^{25}}$	**3.23 seconds**	**160 GB**

9. POSSIBLE IMPROVEMENTS TO THE PROPOSED ATTACK

We believe that several techniques can be employed to lower the total pre-computation, memory and real-time complexity of the attacks. In our opinion, the most relevant topics for further investigation are:

Keep contents of the shift registers as cyclical arrays. The A5/1 algorithm can be extremely efficiently run in software by exploiting flaw in its design, namely that each one of the three shift registers is so small that one can precompute all its possible states, and keep them in memory as three cyclic arrays, where successive locations in each array represent successive states of the corresponding shift register. In fact, as suggested in [7], the attacker doesn't have to keep the full states in the arrays, since the only information he has to know about a state is its clocking tap and its output tap. A state can thus be viewed as a triplet of indices into three large single bit arrays. Furthermore, since there is no mixing of the values of the three registers, the registers only interaction is in determining which of the three indices should be incremented by one and this can be determined by a pre-computed table with three input bits (the clocking taps) and three output bits (the increments of the three registers)

Use special states. As correctly indicated by Biryukov in [7], an important consideration in implementing time-memory tradeoff attacks is that access to disk is about a million times slower than a computational step, and thus it is crucial to minimize the number of times one looks for data on the hard disk. An old idea due to Ron Rivest is to keep on the disk only special states which are guaranteed to produce output bits starting with a particular pattern of length k, and to access the disk only when we encounter such a prefix in the data. This reduces the number b of disk probes by a factor of about 2^k. The downside is that preprocessing stage gets longer in time.

Special states can efficiently be computed. As pointed on in [7], a major weakness of A5/1 which we exploit in both attacks is that it is easy to generate all the states which produce output sequences that start with a particular pattern without trying and discarding other states. This is due to a poor choice of the clocking taps, which makes the register bits

that affect the clock control and the register bits that affect the output unrelated for about 16 clock cycles, so we can choose them independently.

Consider states with higher occurrence probability (i.e. red/green states). Its has been shown in multiple different sources that the probability distribution of different states in A5/1 is not uniform .Thus, storing only the states that have for the have strong positive correlations, the attack can be run much more efficiently.

Before further investigating the possible improvements on the proposed attack, let us introduce the following two concepts:

Definition 1. A state S is defined as red, if the sequence of output bits produced from state S starts with ff (i.e., it is a special state). The subspace of all the red states is denoted by R.

Definition 2. A state is defined as green, if the sequence of output bits produced from state s contains an occurrence of ff which starts somewhere between bit positions 101 and 277. The subspace of all the green states is denoted by G.

The red states are the states that the attacker keeps in the disk, look for in the data, and try to collide by comparing their prefixes. The green states are all the states that could serve as initial states in frames that contain ff. Non-green initial states are of no interest to us, since they are discarded. According to [7], the “redness” of a state is not directly related to its separate coordinates in the triplet space and they can be viewed as randomly and sparsely located in this representation. The size of G is approximately 177 times the size of R and since a short path of length 277 in the output sequence is very unlikely to contain two occurrences of ff, the relationship between green and red states is many-to-one.

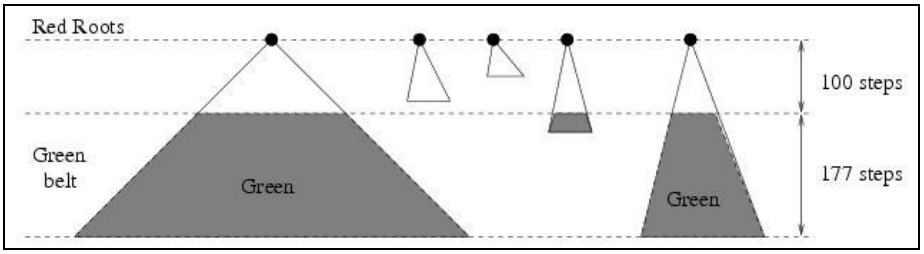

Figure 9.1. Red roots and their green belts forming trees of different sizes

Thus, the set of all the relevant states we consider can be viewed as a collection of disjoint trees of various sizes, where each tree has a red state as its root and a ``belt'' of green states at levels 101 to 277 below it (see Figure.3 above from [7]). The weight W (s) of a tree whose root is the red state s is defined as the number of green states in its belt, and s is called k-heavy if W (s) ≥ k.

The crucial observation made in [7] is that in A5/1 there is a huge variance in the weights of the various red states. According to the results in [7], the weight of about 85 per cent of the states is zero, because their trees die out before reaching depth 100. In Biryukov's actual attack, they keep 2^{35} red states on the disk. And find that although they store only a tiny fraction of the whole red states space R, they can choose particularly heavy trees with an average weight of 12,500. The expected number of colliding red states in the disk and the actual data increases then significantly.

The same concept can also be utilized in our proposed attack by selectively choosing only those states as red states that have wide green belts and aiming at colliding the green states in their belts. By cleverly choosing the red states with wide green belts (e.g. like the ones proposed in [7] covering over 12,500 green states) the efficiency of the proposed attack can be increased by almost two orders of magnitude.

To summarize, in this paper, we describe a new real-time attack on A5/1 stream ciphers that achieves superior performance in terms of reduced computation time and calculation complexity compared to most of the previous attacks. We introduced the concept of "peculiar events" that can be applied to any array of mutually clock-controlled stop/go shift registers but works exceptionally well in A5/1 due to certain subtle flaws in its design, particularly the position of its clocking tabs. The proposed attack was an

advancement of the “peculiar events” concept through embedding it into an improved version of M. Hellman’s cryptanalytic time-memory tradeoff attack.

We showed that the proposed technique is suitable for real-time attacks on commercially available computers with a pre-computational complexity of $2^{41.33}$ clockings, a memory requirement of $2^{40.21}$ bits (<160 GB), real-time complexity of 2^{25} unit computation durations and ~1 minute of known conversation.

Finally, we compared our attack’s performance with other widely accepted techniques and concluded that ours achieves acceptable success rates at comparably lower memory, data and calculation complexity requirements and pointed on other possible techniques to improve our attack’s performance.

REFERENCES

1. Golic, J., "Cryptanalysis of Three Mutually Clock-Controlled Stop/Go Shift Registers", *IEEE Transactions on Information Theory*, Vol. 46, No.3, May 2000.

2. Rueppel, R.A., "Stream Ciphers", *Contemporary Cryptology: The Science of Information Integrity, IEEE Press*, pp. 65-134, 1991.

3. Briceno, M., Goldberg, I. and Wagner, D., "A pedagogical implementation of A5/1", *http://www.scard.org*, May 1999.

4. Golic, J., "Crytanalysis of Alleged A5 Stream Cipher", *Proceedings of EUROCRYPT'97*, LNCS 1233, pp.239-255, Springer-Verlag, 1997.

5. Hellman, M.E., "A Cryptanalytic Time-Memory Tradeoff", *IEEE Transactions on Information Theory*, Vol. IT-26, N 4, pp.401-406, July 1980.

6. Borst, J., Preneel, B. and Vandewalle, J., "On the Time-Memory Tradeoff between Exhaustive Key Search and Table Precomputation", *Proceedings 19th Symposium on Information Theory in the Benelux*, pp. 111-118, May 28-29, 1998.

7. Biryukov A., A. Shamir and D. Wagner, "Real Time Cryptanalysis of A5/1 on a PC", *Proceedings of FSE 2000, Lecture Notes in Computer Science,* Vol. 1978, pp. 1–18, 2001.

8. Biham, E. and O. Dunkelman, "Cryptanalysis of the A5/1 GSM Stream Cipher", *Proceedings of INDOCRYPT 2000*, *Lecture Notes in Computer Science,* Vol. 1977, pp. 43–51, 2000.

9. Oechslin, P., "Making a Faster Cryptanalytic Time-Memory Trade-Off", *CRYPTO 2003, Lecture Notes in Computer Science,* vol. 2729, pp 617-630, 2003.

10. Erguler, I., Anarim, E. , “A Cryptanalytic Time-Memory Tradeoff for Stream Ciphers”, 2004.

11. Jonsson, “Some Results on Fast Correlation Attacks”, *Ph.D. Thesis*, 2002.

12. Kahn, *The Codebreakers: The Story of Secret Writing*, Micmillan Publishing, 1967.

13. Singh, S., *The Code Book,* Fourth Estate, London*,* 1999.

14. Shannon, C.E., “Communication theory of secrecy systems”, *Bell System Technical Journal,* Vol. 27.65671, 1949.

15. Diffie, W. and Hellman, M.E., “New directions in cryptography.”, *IEEE Transactions on Information Theory*, IT-22:644, 654, 1976.

16. Rivest, R.L., Shamir, A. and Adleman, L., “A method for obtaining digital signatures and public key cryptosystems.” *Communications of the ACM*, vol. 21.120.126, 1978.

17. Coppersmith, D., Krawczyk, H. and Mansour, Y., “The shrinking generator”, *Advances in Cryptology CRYPTO'93*, Vol. LNCS 547, pages 22 39, Springer-Verlag, 1993.

18. Schneier, B., *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, John Wiley and Sons, New York, 2nd edition, 1996.

19. Stinson, D.R., *Cryptography: Theory and Practise*, CRC Press, Boca Raton, 1995.

20. Cover and Thomas, *Elements of Information Theory*, John Wiley and Sons, New York, 1991.

21. Ekdahl, P., “On LFSR Based Stream Ciphers”, *Ph.D. Thesis*, Lund University, 2003.

22. Erguler, I., "The Design Principles of Linear Feedback Shift Register Based Clock Controlled Stream Ciphers", *Master Thesis*, 2005.

23. Schneier, B., *Applied Cryptography Second Edition: Protocols, Algorithms, and Source Code in C*, John Wiley & Sons, New York, First Edition, 1996.

24. Menezes, A., van Oorschot, P. C., and Vanstone, S. A., *Handbook of Applied Cryptography*, CRC Press Inc., 1997.

25. Maximov, A., Johansson, T., Babbage, S., "An Improved Correlation Attack on A5/1", Springer Verlag, Berlin Heidelberg, 2005.

26. Ekdahl, P. and T. Johansson," Another Attack on A5/1", *IEEE Transactions on Information Theory,* Vol. 49, pp. 1-7, 2003.

27. Briceno, M., Goldberg, I. and Wagner, D., "A pedagogical implementation of A5/1", *http://jya.com/a51-pi.htm*, 1999.

28. Erguler, I. and Anarim, E., "A Hybrid Model for GSM Stream Generator", 2004.

29. Tarkkala, L., "Tik,110.551: Attacks against A5", *HUT TML*, 2000.

30. Babbage, S., "A Space/Time Tradeoff in Exhaustive Search Attacks on Stream Ciphers", *European Convention on Security and Detection, IEE Conference Publication*, No. 408, May 1995.

31. Biryukov, A. and A. Shamir, "Cryptanalytic Time/Memory/Data Trade-offs for Stream Ciphers", *Proceedings of ASIACRYPT 2000, Lecture Notes in Computer Science*, Vol. 1976, pp.1-13, 2000.

32. Barkan, E., E. Biham and N. Keller, "Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication", *Proceedings of CRYPTO 2003*, *Lecture Notes in Computer Science*, Vol. 2729, pp. 600-616, 2003.

33. Biham, E. and Shamir, A., “Differential Cryptanalysis of DES-like Cryptosystems”, *Journal of Cryptology*, Vol.4, No.1, 1991.

34. Rose, Hawkes, “On the Applicability of Distinguishing Attacks Against Stream Ciphers”, 2002.

35. Siegenthaler, “Decrypting a class of stream ciphers using ciphertext only”, *IEEE Transactions on Computers*, C-34, pp. 81 85, 1985.

36. Siegenthaler, “Correlation-immunity of nonlinear combining functions for cryptographic applications”, *IEEE Transactions on Information Theory*, IT-30:776 780, 1984.

37. Maximov, Johansson, Babbage, “An Improved Correlation Attack on A5/1”, 2005.

38. Ekdahl, P. and Johansson, T. “Another attack on A5/1”, *IEEE Transactions on Information Theory*, Vol.49(1), pp284–289, 2003.

39. Golic, J., Clark, A. and Dawson, E., “Inversion Attack and Branching”, *ACISP,* pp. 88-102, 1999.

40. Kim, Matsumoto, “Achieving higher success probability in time-memory tradeoff cryptanalysis without increasing memory size”, *IEICE Transactions on Communications/Electronics/Information and Systems*, 1999.

41. Standaert, F., Rouvroy, G., Quisquater, J. and Legat, J. “A Time-Memory Tradeoff Using Distinguished Points: New Analysis & FPGA Results”, *CHES,* pp. 593-609, 2002.

42. Kim, K., S. Lee, S. Park and D. Lee, “Securing DES S-boxes against Three Robust Cryptanalysis”, *Proceedings of the Workshop on Selected Areas in Cryptography*, pp. 145-157, 1995.