Cryptanalysis of the *A5/1* Stream Cipher

A thesis submitted to
Indian Institute of Science Education and Research Pune
in partial fulfillment of the requirements for the
BS-MS Dual Degree Programme

Thesis Supervisor: Ayan Mahalanobis

by
Jay Jitesh Shah
April, 2012

Indian Institute of Science Education and Research Pune
Sai Trinity Building, Pashan, Pune India 411021

This is to certify that this thesis entitled "Cryptanalysis of the *A5/1* Stream Cipher" submitted towards the partial fulfillment of the BS-MS dual degree programme at the Indian Institute of Science Education and Research Pune, represents work carried out by Jay Jitesh Shah under the supervision of Ayan Mahalanobis.

Jay Jitesh Shah

Thesis committee:
Ayan Mahalanobis
Amit Kalele

A. Raghuram
Coordinator of Mathematics

Dedicated to my parents, Madhvi Shah and Jitesh Shah.

Acknowledgments

I would like to take this opportunity to thank everyone who helped me directly or indirectly for the success of this dissertation. First and foremost, I would like to thank my family for their unconditional love and blessings. Any success I may achieve is directly traceable to their support of my interests and devotion to my development.

This dissertation could not have been written without the guidance of my mentor, Dr. Ayan Mahalanobis, who spent hours advising me and encouraged me throughout this project inspite of innumerable difficulties faced. He never accepted anything less than my best. I am greatly indebted to him for his patience and enthusiasm.

It is my privilege to thank Prof. Jörg Keller for giving me an opportunity to work under his guidance at the FernUniversität in Hagen, Germany in the summer of 2011. With his motivation, I was inspired to convert my summer project to a full-fledged project.

My internship at the Computational Research Laboratories Ltd., TATA Sons, Pune, under Dr. Amit Kalele proved of utmost importance as it spurred my interest in the *A5/1* stream cipher. I am greatly indebted to him and his research team for the discussions which helped me adapt different approaches to solve problems.

Last but not the least, I would like to thank my colleagues Nikhil Kasat, Shadab Alam and Shashwat Antony for their precious time and assitance during the implemention of my code. A special vote of thanks for my friends Kartik, Amitosh, Prashant and Niharika who have always been by my side through thick and thin.

Abstract

Cryptanalysis of the *A5/1* Stream Cipher

by Jay Jitesh Shah

In Europe and North America, the most widely used stream cipher to ensure privacy and confidentiality of conversations in GSM mobile phones is the *A5/1*. In this thesis, we study the *A5/1* and some known attacks on it. We explore the weaknesses of the cipher and suggest certain modifications to the *A5/1* encryption scheme with an aim to create a more secure cryptosystem resistant to most of the attacks already known. We have also designed a new attack on the *A5/1* stream cipher with a minimum space complexity of around 2^{40} and an average complexity of $2^{48.5}$, which is much less than the brute-force attack with a complexity of 2^{64}. We provide a detailed description of our new attack along with its implementation and results. Various statistical tests for randomness were performed on the suggested variants of the *A5/1* which prove that these modified stream ciphers are pseudo random number generators.

Keywords: *A5/1*, GSM, guess-and-determine attack, stream ciphers

Contents

Chapter 1

Introduction

In this review chapter, we give a brief introduction to stream ciphers, discuss the history of the *A5* family and give an insight to the current research of the same. We conclude this chapter with our contributions to the cryptanalysis of the *A5/1* cipher.

1.1 Introduction to Stream Ciphers

The art and science of making and breaking 'secret' codes is called *cryptology*. It can be divided into two parts: cryptography - the art and science of making secret codes; and cryptanalysis - the science of breaking secret codes i.e., recovering the plaintext of a message without access to the key. These secret codes are known as ciphers or cryptosystems [29].

The information a sender wishes to transmit to a receiver is the plaintext, while the unreadable text that results from encrypting the plaintext is the ciphertext. Encryption is accomplished by combining the plaintext with the key to yield the ciphertext.

$$\textbf{Plaintext} + \textbf{Key} = \textbf{Ciphertext}$$

Decryption is the inverse process, where the ciphertext is converted back to plaintext.

There are two types of ciphers: symmetric ciphers - where the same key is used for encryption and decryption; and asymmetric ciphers - where different keys are used for encryption and decryption. Symmetric ciphers can be divided into two categories: stream ciphers - where the plaintext is encrypted one bit at a time to give the ciphertext; and block ciphers - where the plaintext is encrypted in blocks (groups of bits) to give the ciphertext [28].

A stream cipher is a symmetric key cipher where plaintext bits are combined with a pseudorandom cipher bit-stream (key), usually by an exclusive-or (XOR) operation, to give a ciphertext. By definition, exclusive-or (XOR) is a logical operation on two operands that yields true iff exactly one (but not both) of the two conditions is true, and false if both conditions are the same. In a stream cipher the plaintext is encrypted one at a time to give the corresponding ciphertext[29].

In cryptology, a known-plaintext attack is an attack where the attacker has samples of both the plaintext and the corresponding ciphertext. If the attacker knows the method of encryption and has access to part or all of the plaintext and the ciphertext, then the attacker can deduce the secret key used to encrypt the plaintext message. This in turn can compromise the security of future messages sent with that key[24]. The following is a short representation of the same:

$$\textbf{Plaintext} \oplus \textbf{Key} = \textbf{Ciphertext}$$
$$\textbf{Ciphertext} \oplus \textbf{Plaintext} = \textbf{Key}$$

A shift register is a sequence of bits. The length of a shift register is figured in bits i.e., if it is n bits long, it is called an n-bit shift register. Each time a bit is needed, all the bits in the shift register are shifted one place to the left. The new right-most bit, known as the feedback bit, is computed as a function of other bits in the register. If a shift register has a linear feedback function, i.e., if the function involves only XOR operations of certain bits of the register, then it is known as a linear feedback shift register. The bits that will be XORed to give the right-most bit are called the tapping bits. The output of the shift register is one bit known as the most significant bit. The output sequence is known as keystream bits. The security of a stream cipher completely depends on the keystream. This keystream should be random for it to be secure. One should not be able to predict the $(n+1)^{th}$ bit given n output bits of the keystream. Hence, a stream cipher must be unpredictable and a pseudorandom number generator to be secure. The *period* of the shift register is the length of the output sequence before it starts repeating [26].

The most widely used stream cipher in softwares is the *RC4*. Other stream ciphers used are *Chameleon*, *FISH*, *Helix*, *ISAAC*, *MUGI*, *Panama*, *Pike*, *SEAL*, those in the *A5* family and so on. *A5/1* is used for encryption in GSM mobiles, *E0* for Bluetooth technology, and *A5/2*, *A5/3* and *A5/8* ciphers are used for other mobile phone technologies. *A5/1* is used to encrypt both voice and signaling data [28].

The following four parameters should be considered when comparing attacks from the viewpoint of efficiency and practical threat to security:

- Time complexity: Time complexity is the number of operations needed to complete an attack. The worst-case time complexity of an exhaustive search is equivalent to the size of the keyspace. The efficiency of any other attack must be compared to this. The cipher is said to be broken if there exists an attack that succeeds in finding the key in less than the exhaustive search 2^{64} in the case of *A5/1*.

  The time needed to complete an attack can be divided into pre-computational complexity and attack-time complexity. Pre-computations are those computations that are done before the attack is launched. Note the difference between worst-case complexity and the average-case complexity. Worst-case complexity is the time after which the attack is bound to finish, whereas average-case complexity refers to the time after which the attack is expected to finish.

- Data complexity: It is the amount of plaintext-ciphertext pairs (data) needed to complete the attack with a given success probability.

- Space complexity: It is the amount of memory needed to perform the attack successfully. Space complexity is directly related to pre-computational time complexity, i.e., large memory requirement implies long precomputation time.

- Success probability: Deterministic attacks are guaranteed to succeed within time T; given an amount D of plaintext and memory M. Whereas probabilistic attacks have success probability $p < 1$ for fixed parameters T, M and D.

1.2 History of the *A5* Ciphers

The *A5/1* encryption algorithm was developed in the late 1980s, but the development of GSM initiated earlier in that decade. In 1982, Groupe Speciale Mobile was established and the development of a new digital cellular standard began. This was the first initiative to create a pan-European mobile communication network. This was also the birth of the GSM acronym, which was later changed to Global System for Mobile communications. In 1989, the European Telecommunications Standards

Institute (ETSI) - a newly created entity, was in charge of the special development of GSM [17].

There are multiple versions of the encryption algorithm which belong to the *A5* family: *A5/0* is a dummy cipher with no encryption; *A5/1* (the subject of this Master's thesis) is the original *A5* algorithm used in Europe and North America that ensures over-the-air communication privacy and confidentiality of conversations in GSM mobile phones; *A5/2* is an intentionally weaker encryption algorithm created for export; while *A5/3* is a strong encryption algorithm created as part of the 3rd Generation Partnership Project *(3GPP)*, which is currently responsible for maintaining and developing GSM technical specifications around the world [17].

The *A5/1* was developed in 1987, when GSM was not yet considered for use outside Europe, and *A5/2* was developed in 1989. Both were initially kept secret. However, the general design was leaked in 1994, and the algorithms were entirely reverse engineered in 1999. In 2002, an additional new version *A5/3*, was added to the *A5* family. Unlike, *A5/1* and *A5/2*, it's internal design was published. *A5/3* is based on the block-cipher KASUMI, which is used in third generation (3G) networks.

Anderson [1], Golic [14] and Babbage [2] were the pioneers in initially cryptanalyzing the *A5/1* encryption algorithm when only a rough outline of the *A5/1* was leaked. After *A5/1* was reverse engineered, it was analyzed by Biryukov, Shamir and Wagner [6]; Biham and Dunkelman [5]; Ekdahl and Johansson [10]; Maximov, Johansson and Babbage [23]; Barkan and Biham [4]; Keller and Seitz [19]; and a few other researchers. We shall study some of these attacks in detail.

1.3 Current Research

Several attacks on the *A5/1* stream cipher have been designed in the last twenty years, but only a few of them have been implemented. Attacks on the GSM protocol can work even if the network supports only *A5/1* or *A5/3* encryption, as long as the mobile phone supports *A5/2* encryption. The main flaw that allows the implementation of these attacks is that the same key is used regardless of whether the phone encrypts using *A5/1*, *A5/2*, or *A5/3* algorithm. Therefore, the attacker can mount a man-in-the-middle attack, in which the attacker impersonates the mobile to the network, and the network to the mobile (by using a fake base station). The attacker might use *A5/1* for communication with the network and *A5/2* for communication

with the mobile. But due to the flaw, both algorithms encrypt using the same key. The attacker can gain the key through the passive attack on *A5/2*. The attacker who is in the middle can eavesdrop, change the conversation, perform call theft, etc. The attack applies to all the traffic including short message service (SMS) [4].

An International Mobile Subscriber Identity (IMSI) is a unique identification associated with all GSM network mobile phone users. An IMSI catcher is a device for identifying the IMSI of a nearby GSM mobile phone and intercepting it. It masquerades as a base station for all mobile stations in the vicinity, forcing the mobile to switch to *A5/0* mode, which has no encryption, making the call easy to intercept and convert to audio.

1.4 Our Contributions

We have studied the *A5/1* stream cipher and analyzed its weaknesses. We studied the known attacks on the *A5/1*. We have designed a new attack on the existing *A5/1* stream cipher with minimum complexity of approximately 2^{40} and an average complexity of $2^{48.5}$, which is much lesser than a brute-force attack of 2^{64} complexity (refer Chapter 4). This attack has a 100% success rate. With the knowledge of only 11 bits of the known keystream, the attack algorithm is able to determine a set of 64-bit complete state candidates for that input. A complete state candidate is a state candidate with no vacant bits. With every additional clocking round, the number of complete state candidates increases. Thus, the probability of finding the correct state candidate amongst all the complete state candidates increases with every additional round. We provide a detailed description of our new attack along with its implementation and results.

We suggest certain modifications to the existing *A5/1* with an aim to create a secure cryptosystem resistant to most of the attacks already known (refer Chapter 5). We have also performed various statistical tests for randomness on the suggested variants of the *A5/1* which prove that these modified stream ciphers are pseudo random number generators like the *A5/1*.

Chapter 2

The *A5/1* Encryption Algorithm

In this chapter, we describe the *A5/1* encryption algorithm. The description of the *A5/1* was initially kept secret, but it's design was disclosed in 1999 by reverse engineering [7]. The GSM organization has later confirmed the correctness of the algorithm [6]. We end this chapter by generalizing certain characteristics of an output stream and the properties of maximal length sequences.

2.1 Description of the *A5/1* Stream Cipher

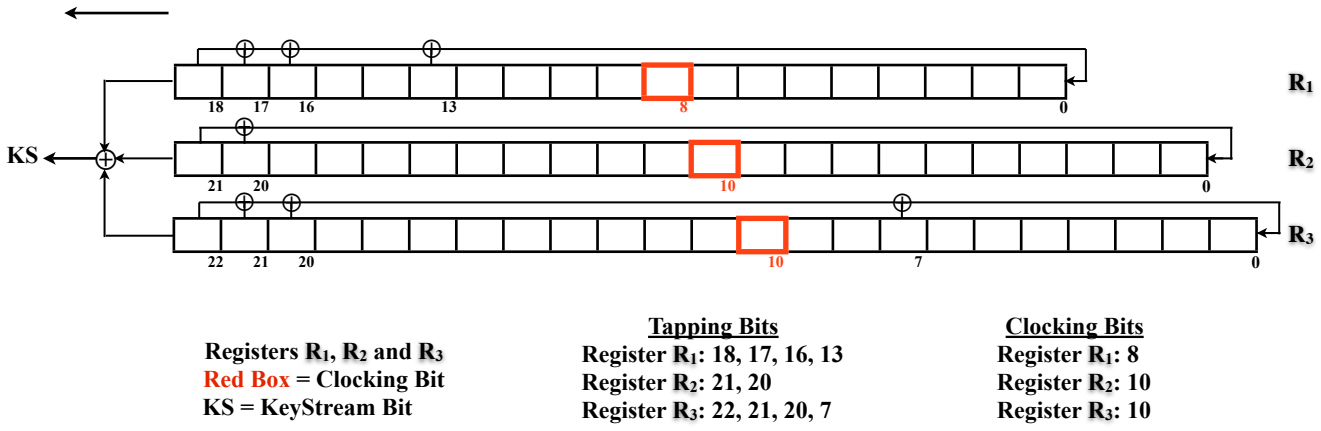

Figure 2.1: *A5/1* Stream Cipher

The *A5/1* stream cipher is built from three short linear feedback shift registers (LFSRs) of lengths 19, 22, and 23 bits, denoted by R_1, R_2 and R_3 respectively. The

rightmost bit in each register is labeled as bit zero. The tapping bits of R_1 are at bit positions 13, 16, 17, 18; the tapping bits of R_2 are at bit positions 20, 21; and the tapping bits of R_3 are at bit positions 7, 20, 21, 22 (Table 2.1).

Table 2.1: The *A5/1* Register Parameters

Register	Register Length	Clocking Bits	Primitive Polynimials	Tapping Bits
R_1	19 bits	8	$1 + x + x^2 + x^5 + x^{19}$	18, 17, 16, 13
R_2	22 bits	10	$1 + x + x^{22}$	21, 20
R_3	23 bits	10	$1 + x + x^2 + x^{15} + x^{23}$	22, 21, 20, 7

The tapping bits are pre-determined according to the corresponding primitive polynomials for the registers. A primitive polynomial is a polynomial that generates all elements of an extension field from a base field. A polynomial is said to be irreducible if it cannot be factored into nontrivial polynomials over the same field. For example, in the field of rational polynomials $Q[x]$ (i.e., polynomials $f(x)$ with rational coefficients), $f(x)$ is said to be irreducible if there does not exist two non-constant polynomials $g(x)$ and $h(x)$ in x with rational coefficients such that

$$f(x) = g(x)h(x)$$

Primitive polynomials are also irreducible polynomials. A polynomial of degree n over the finite field $GF(2)$ (i.e., with coefficients either 0 or 1) is primitive if it has polynomial order $2^n - 1$. For each register, when the register is clocked, its tapping bits are XORed together and the result is stored in the rightmost bit of the left-shifted register. The three registers are maximal length LFSRs with periods $2^{19} - 1$, $2^{22} - 1$, and $2^{23} - 1$, respectively [20].

The *A5/1* keystream generator works as follows [7]. First, an initialization phase is run. At the beginning of this phase, all bits of the registers are set to 0. Then the key setup and the *Initialization Vector (IV)* setup are performed. During the initialization phase, all three registers are clocked and the key bits followed by the IV bits are XORed with the *most significant bits* (MSBs) of all three registers. Thus, the initialization phase takes an overall of $64 + 22 = 86$ clock-cycles after which state S_i is achieved.

Based on this initial state S_i, a warm-up phase is performed where the generator is clocked for 100 clock-cycles and the output is discarded. This results directly in state S_w producing the first output bit 101 clock-cycles after the initialization

phase. During the warm-up phase and the stream generation phase that follows, the registers R_1, R_2, and R_3 are clocked irregularly according to the majority function rule [8] depending on the *clocking bits* (CBs) of the three registers. The majority function is a function from n inputs to one output. The value of the operation is true when $\frac{n}{2}$ or more arguments are true, and false otherwise.

The registers are clocked in a stop/go fashion using the following majority rule: Each register has a single clocking bit (bit 8 for R_1, bit 10 for R_2, and bit 10 for R_3) which decides the clocking pattern for its respective register. In each clock cycle, the majority function of the clocking taps is calculated and only those registers whose CBs agree with the majority function are clocked. At each step either two or three registers are clocked, and that each register has a probability of moving 3 out of 4 times (Figure 2.2). It is this clocking pattern which makes the stream cipher generate output bits which are random.

CB_1	**CB_2**	**CB_3**	**Majority**	**CLOCKING?**		
				R_1	**R_2**	**R_3**
0	**0**	**0**	**0**	✓	✓	✓
0	**0**	**1**	**0**	✓	✓	-
0	**1**	**0**	**0**	✓	-	✓
1	**0**	**0**	**0**	-	✓	✓
1	**1**	**0**	**1**	✓	✓	-
1	**0**	**1**	**1**	✓	-	✓
0	**1**	**1**	**1**	-	✓	✓
1	**1**	**1**	**1**	✓	✓	✓

Figure 2.2: Majority Function of the Clocking Bits

During encryption, a total of four cases are possible for clocking pattern of the registers. They are:

- Case 1: $CB_1 = CB_2 \neq CB_3$ (Clock R_1 and R_2 only)
- Case 2: $CB_1 \neq CB_2 = CB_3$ (Clock R_2 and R_3 only)
- Case 3: $CB_1 = CB_3 \neq CB_2$ (Clock R_1 and R_3 only)
- Case 4: $CB_1 = CB_2 = CB_3$ (Clock R_1, R_2 and R_3 i.e., all three registers),

where CB_i denotes the *clocking bit* for register i; $i = (1, 2, 3)$.

After clocking, an output bit is generated from the values of R_1, R_2, and R_3 by XORing their *most significant bits* (MSBs), as shown in Equation 2.1. This XORed bit is called the *keystream bit* (KS). After warm-up phase, the *A5/1* produces 228 output bits. For every clock cycle, 114 bits are used to encrypt uplink traffic, while the remaining 114 bits are used to decrypt downlink traffic [12].

$$R_1[18] \oplus R_2[21] \oplus R_3[22] = KS[i], \tag{2.1}$$

where KS[i] denotes the i^{th} keystream bit, $i = 0$ on initialization and increases by 1 after every clocking round.

2.2 Characteristics of an Output Stream

By definition, the period of a *Linear Feedback Shift Register* (LFSR) is the length of the output stream before repetition of the output stream sequence occur. The output streams for the *A5/1* are the *keystream bits*. Some of the features of the output stream discussed by Golomb [15] are as follows:

- Number of 0s and 1s: In a random output stream, the difference between the number of 1s and the number of 0s will tend to grow progressively smaller in proportion to the length of the stream as the stream gets longer. In an infinite random stream, the number of 1s and the number of 0s will be equal.
- Runs of 0s and 1s: A run is a pattern of equal values in the bit stream. A bit stream like 1011101001 has six runs of the following lengths in order: 1, 1, 3,

1, 1, 2, 1. One period of an n-bit LFSR with a maximal length tap sequence will have 2^{n-1} runs (e.g., a 5 bit stream yields 16 runs in one period). $\frac{1}{2}$ the runs will be one bit long, $\frac{1}{4}$ the runs will be two bits long, $\frac{1}{8}$ the runs will be three bits long, etc., up to a single run of zeroes that is $n-1$ bits long and a single run of ones that is n bits long. Statistically, a random stream of sufficient length shows similar behavior.

- Shifted Stream: Take the stream of bits in one period of an LFSR with a maximal length tap sequence and circularly shift it any number of bits less than the total length. Do a bitwise XOR with the original stream. The resulting pattern must exhibit the behaviors discussed in the above items.

- Deterministic Property: The LFSR output streams are deterministic i.e., if one knows the present state, one can predict the next state.

- Reversibility: The output stream is reversible. An LFSR with mirrored tapping bits will cycle through the output sequence in reverse order.

The output bits i.e., keystream bits of the *A5/1* have all the above-mentioned features.

2.3 Maximal Length Sequence

A *maximum length sequence* is a type of pseudorandom binary sequence generated using maximal *linear feedback shift registers* (LFSRs). The length of the sequence before repetition of the sequence occurs is called maximal length sequence. The maximal length sequence is equal to $2^n - 1$ where n is the degree of the shift register. LFSRs can have multiple maximal length sequences. There is no quick way to determine if a tap sequence is maximal length. However, there are some ways discussed by Golomb [15] to determine if one is not maximal length:
1) Maximal length tap sequences always have an even number of taps.
2) The tap values in a maximal length tap sequence are all relatively prime. A tap sequence like $12, 9, 6, 3$ will not be maximal length because the tap values are all divisible by 3.

The discovery of one maximal length tap sequence automatically leads to another. If a maximal length tap sequence is described by $[n, A, B, C]$, another maximal length

tap sequence will be described by $[n, n-C, n-B, n-A]$. For example, if $[32, 3, 2, 1]$ is a maximal length tap sequence, $[32, 31, 30, 29]$ is also a maximal length tap sequence.

Chapter 3

Known Attacks on the *A5/1*

This chapter deals with certain known guess-and-determine attacks on the *A5/1* stream cipher. These include Anderson's Attack [1], Golic's Attack [14], Biham-Dunkelman's Attack [5], Keller-Seitz's Attack [19] and Gendrullis-Novotny-Rupp's Attack (also known as the Modified Keller-Seitz Attack) [13]. We conclude this chapter by giving an insight of Hellman's Time Memory Trade-Off Attack in the known plaintext attack approach.

3.1 Guess-and-Determine Attacks

Guess-and-determine attacks [24] are general attacks on stream ciphers where the attacker guesses some bits of the cipher and the remaining bits are determined accordingly.

3.1.1 Anderson's Attack

Assumptions: Guess all the bits of registers R_1 and R_2. Guess the lower half of register R_3. 64 bits of keystream (KS) known.

Aim: Determine the remaining bits of register R_3 (Figure 3.1).

Protocol: Anderson [1] suggested to clock the registers according to the majority function and determine the most significant bits (MSBs) of register R_3 by the follow-

ing equation

$$R_3[22] = R_1[18] \oplus R_2[21] \oplus KS[i]$$

where KS[i] denotes the i^{th} keystream bit, $i = 0$ on initialization and increases by 1 after every clocking round.
In the worst-case, each of the 2^{52} determined state candidates need to be verified against the known keystream.

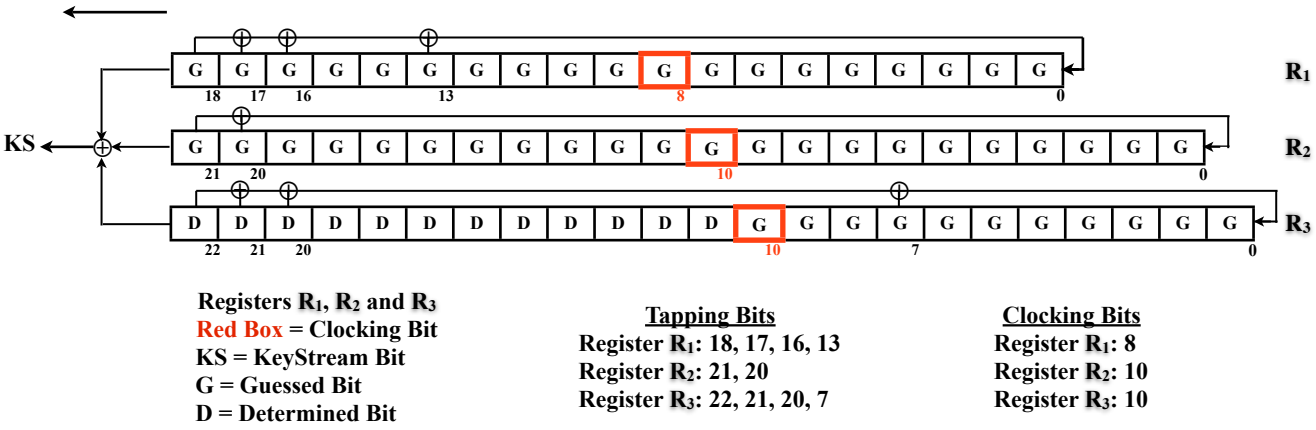

Figure 3.1: Anderson's Attack on the *A5/1* Stream Cipher

Discussion: Golic's approach [14], Biham-Dunkelman's approach [5], Keller-Seitz's approach [19] and Gendrullis-Novotny-Rupp's approach [13] are have lesser complexity than Anderson's approach. Hence, this attack was not implemented.

3.1.2 Golic's Attack

Assumptions: Guess the lower half of all three registers. 64 bits of keystream known.

Aim: Determine the remaining bits of all three registers (Figure 3.2).

Protocol: Clock the cipher until all the guessed bits are 'over'. Golic proposed an attack that has a complexity of 2^{40} linear equations sets. However, each operation in this attack is much more complicated since it is based on the solutions of system of linear equations. In practice, this algorithm is not better than the Anderson's approach [1] or Keller-Seitz's [19] approach. In deriving the solution of the system of

equations, we additionally require solving 44 linear equations.

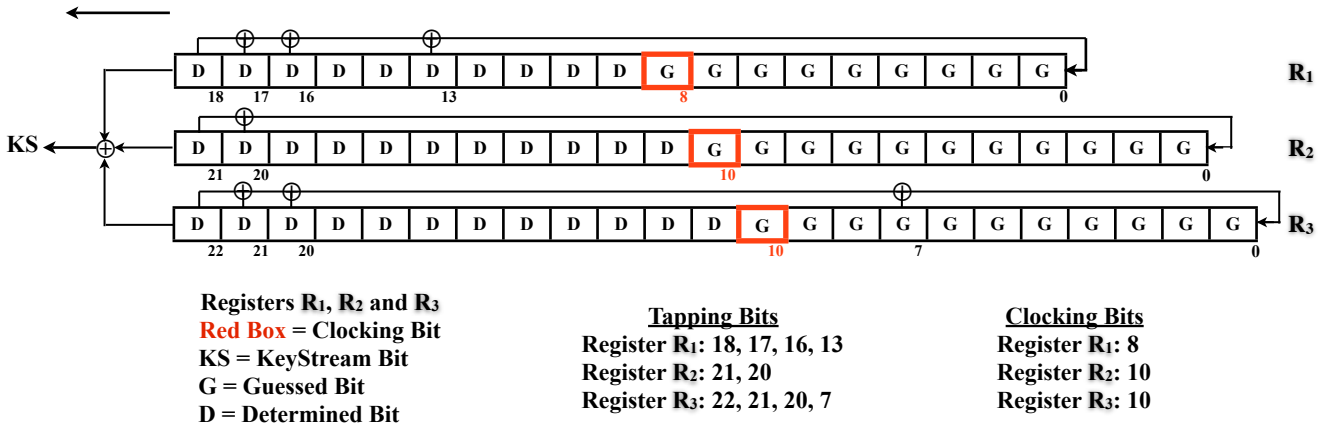

Figure 3.2: Golic's Attack on the *A5/1* Stream Cipher

Discussion: In Golic's attack, we have 44 linear equations. Firstly, guessing the lower half of each of the registers gives the first 31 equations (9 from register R_1, 11 from R_2 and 11 from R_3); Secondly, R_3 will need atleast 12 clocking cycles for it to be completely filled, hence 12 equations and one equation from the most significant bit of all 3 registers which on XOR gives the keystream bit. So in all 1+12+31= 44 equations. The probability for each LFSR to be clocked is three out of four and the majority-clocking rule guarantees that at least two of the three registers are clocked at each cycle. With this information, one can solve the system of linear equations using *Gaussian Elimination* method.

Now we have 44 equations and thus we have information of 44 bits of the internal state candidate. But still there are 20 bits vacant. That gives us a 64x64 Linear System of Equations to be solved, followed by the verification of the corresponding state candidate. This makes Golic's approach impractical to implement.

Pornin and Stern [27] proposed a *Software-Hardware tradeoff attack* that is based on Golic's approach. But in contrast to Golic's approach, they guess the clocking sequence at the very beginning. The increased assumptions and complexity of the attacks make the actual implementation very difficult and impractical.

3.1.3 Biham-Dunkelman's Attack

The Biham-Dunkelman attack [5] attempts to recover the internal state of the cipher. The attack is expected to be a thousand times faster than the Anderson's approach [5] or Keller-Seitz's approach [19], so the expected time complexity is less than a day on a standard PC. The attack requires 2^{47} *A5/1* clockings. The attack also requires about $2^{20.8}$ bits of plaintext data, which is equivalent to 2.36 minutes of conversation. Hence, a lot of pre-computation space is needed.

Assumptions: The clocking bit (CB) of register R_3 and *most significant bit* of R_3 (i.e., $R_3[22]$) are guessed. Register R_3 is assumed not to be clocked for ten consecutive rounds; i.e., CB of R_1 = CB of $R_2 \neq$ CB of R_3 for ten rounds. Guess nine bits of R_1 (i.e., $R_1[(9, 18)] \sim R_1[13]$) and one bit of R_2 (i.e., $R_2[0]$).

Aim: Determine all the remaining bits of register R_1 and register R_2 (Figure 3.3).

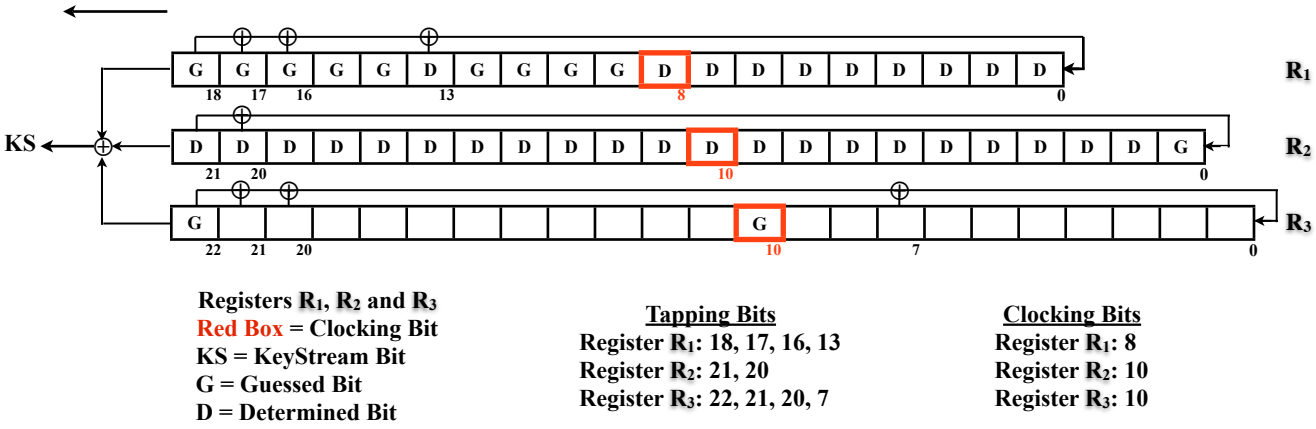

Figure 3.3: Biham-Dunkelman's Attack on the *A5/1* Stream Cipher

Protocol: For 10 rounds, we have 20 bits (10 from R_1 and 10 from R_2).

$$R_1[18] \oplus R_2[21] \oplus R_3[22] = KS[i]$$

where KS[i] denotes the i^{th} keystream bit, $i = 0$ on initialization and increases by 1 after every clocking round.

Here, $R_3[22]$ works for atleast 11 consecutive rounds; hence we have another 11 linear equations.

$$R_1[t] \oplus R_2[t+3];\ where\ t = [8, 18],\ t \in \mathbb{Z}$$

Guessing $R_1[18]$, $R_1[17]$ and $R_1[16]$ gives $R_1[13]$ because of the parity bit, and it also gives $R_2[21]$, $R_2[20]$, $R_2[19]$ and $R_2[16]$.

Complexity: $(2^{10} * 2^4 * 2^1) * 2^{12} = 2^{27}$ Since we have 20 possible starting locations for register R_3, the total time complexity is $2^{27} * 2^{20} = 2^{47}$

A Trade-Off between Computational and Plaintext Complexity

In this section, we note a couple of important questions discussed by Kasper [18]:

Q. By waiting for an event when the third register R_3 is not clocked for 10 consecutive cycles, what is the computational gain achieved?
A. In order to locate an event where R_3 stays unclocked for 10 consecutive cycles, we need 2^{20} different starting locations (for R_3) on average. For each such location, guessing 12 bits immediately reveals 31 more bits. Hence, the 41 bits of registers R_1 and R_2, along with 2 bits from register R_3, can be determined with a complexity of $2^{12} * 2^{20} = 2^{32}$. In comparison with the Keller-Seitz attack [19] approach, for the same 43 bits of internal state candidate, the complexity is almost 2^{42}, as compared to only 2^{32} for the Biham-Dunkelman attack. Thus, the latter attack is 2^{10} times faster than the former attack (for 43 bits).

Q. Find a general solution for other values of n.
A. Fix n to be an integer with the condition $0 \leq n \leq 10$. The output bits are $R_1[18]$, $R_2[21]$ and $R_3[22]$ and the clocking bits are R_1 [8], $R_2[10]$ and $R_3[10]$. The desired event occurs iff $R_1[8] = R_1[7] = \ldots = R_1[8-(n-1)] = R_2[10] = R_2[9] = \ldots = R_2[10-(n-1)] \neq R_3[10]$

If any one of the $2n+1$ bits is fixed, we can immediately determine the remaining $2n$ bits. $\therefore$ one out of the 2^{2n} internal states satisfy this property. Hence, in order to locate our answer, on average, 2^{2n} locations need to be checked.

Guessing the clocking bit for n consecutive cycles yield $2n$ bits from the two registers R_1 and R_2. Now guess $R_3[22]$ and all the remaining $19-n$ bits of R_1. We

calculate $n+1$ bits of R_2. Now we need to guess remaining $22-n-(n+1) = 21-2n$ bits of R_2. Hence, out of 43 bits, only $1 + 1 + (19 - n) + (21 - 2n) = 42 - 3n$ bits need to be guessed for 2^{2n} different locations, giving us a total complexity of $2^{42-3n} * 2^{2n} = 2^{42-n}$.

Confirming the Biham-Dunkelman attack, let $n = 10$ consecutive clocking cycles where register R_3 is not clocked, we need to guess only $42 - 3 * (10) = 12$ bits, hence giving a total complexity of 2^{32}.But when $n = 0$, we need to guess 42 bits, thereby giving a total complexity of 2^{42} which is the brute force or the exhaustive search method.

Limitations: The attacker must know exactly the location of the information-leaking event where register R_3 is unclocked for 10 consecutive rounds. Such an event will happen one out of 2^{20} possible cipher states. This is a big assumption. Thus the attacker will need to probe about 2^{20} different starting locations by trial-and-error before the event actually occurs. Also, the probability that such an event, where register R_3 is not clocked for consecutive 10 rounds occurs is close to zero. This attack requires a lot of data and pre-computation space. Hence this attack is not practical for implementation.

3.1.4 Keller-Seitz's Attack

This approach is based on a simple guess-and-determine attack proposed by Anderson [1], where the shorter registers R_1 and R_2 are completely guessed, the lower half of register R_3 is guessed and the rest of the register R_3 is determined. But since Anderson neglected the asynchronous clocking of the registers at first, only the 12 most significant bits of R_3 could be determined from the known *keystream* whereas the remaining bits have to be guessed as well. Keller-Seitz [19] took into account the asynchronous clocking of the *A5/1* stream cipher and designed this attack.

Assumptions: All bits of registers R_1 and R_2 are guessed. 64 bits of known *keystream*.

Aim: Determine all bits of register R_3 (Figure 3.4).

Protocol: Keller-Seitz's attack was divided into two phases: a *determination phase* in which a possible state candidate consisting of the three registers of *A5/1* after its *warm-up phase* [7] is generated, and a subsequent *post-processing-phase* in which the

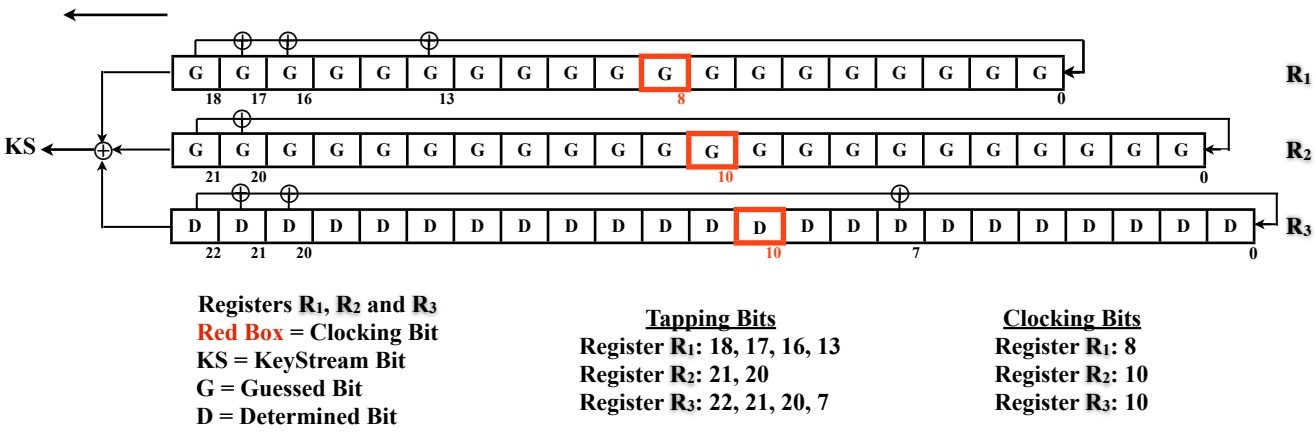

Figure 3.4: Keller-Seitz's Attack on the *A5/1* Stream Cipher

state candidate is checked for consistency. In the determination phase, the authors try to reduce the complexity of the simple *guess-and-determine* attack by early recognizing contradictions that could occur by guessing the clocking bit of R_3 such that R_3 will not be clocked. Hence, all states arising out of the contradictory guess neither need to be computed further on nor checked afterwards.

Limitations: The authors further reduce the complexity by not only discarding the incorrect possibilities for $R_3[22]$ in case of contradiction, but also limit the number of choices to the one of not-clocking R_3, if this is possible without any contradiction. If a case arises where $R_1[8] = R_2[10]$ and $R_3[10]$ has to be guessed, then the authors suggest to always consider the case $R_1[8] = R_2[10] = R_3[10]$ and clock register R_3 with register R_1 and register R_2. This leaves out the possible case of $R_1[8] = R_2[10] \neq R_3[10]$. Thus, the success probability of this attack is approximately 18%, and the number of state candidates inspected by Keller and Seitz to the number of valid states is $\frac{86}{471} \approx 0.18$.

3.1.5 Modified Keller-Seitz's Attack

Gendrullis, Novotny and Rupp [13] proposed a guess-and-determine approach. Unlike Keller-Seitz [19], the authors only discard the wrong possibilities for the clocking bit of register R_3 that would lead to a contradiction. But if no contradiction exists, they check all possibilities of the clocking bit of R_3, which means the case of clocking and not-clocking R_3. Thus, every possible state candidate is taken into account, hence giving us a success probability of 100%.

Assumptions: All bits of registers R_1 and R_2 guessed. 64 bits of known *keystream* bits.

Aim: Determine all bits of register R_3 (Figure 3.5).

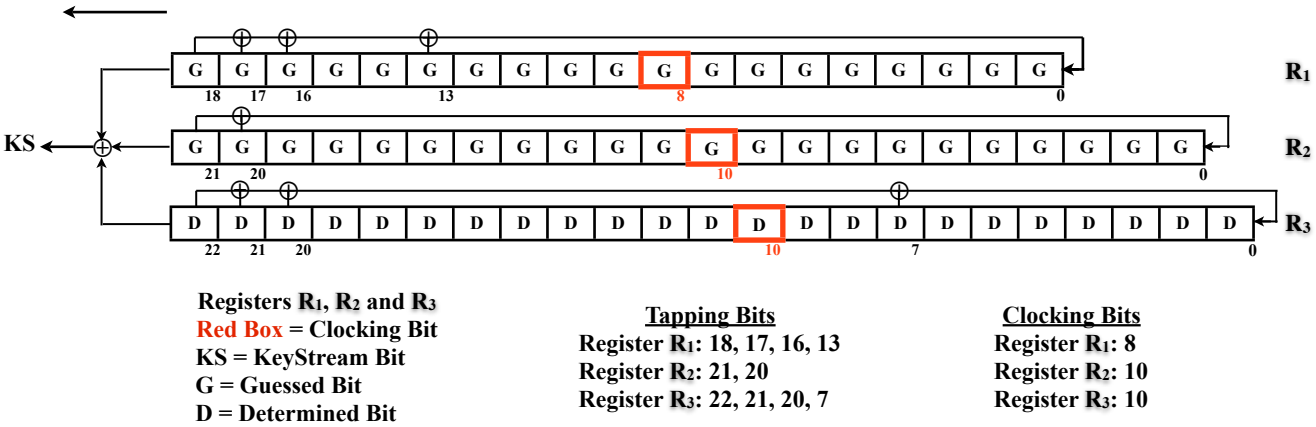

Figure 3.5: Modified Keller-Seitz's Attack on the *A5/1* Stream Cipher

Protocol: The initial approach was to compute register R_3 by regular clocking sequence used during the encryption process. The most significant bit of register R_3 was computed by the equation $R_3[22] = R_1[18] \oplus R_2[21] \oplus KS[i]$, where KS[i] denotes the i^{th} keystream bit, $i = 0$ on initialization and increases by 1 after every clocking round. The clocking bit of R_3 is guessed, followed by clocking of the respective registers. If register R_3 is clocked, the feedback bit (i.e., $R_3[0]$) is calculated by the XOR of the tapping bits (i.e., $R_3[22]$, $R_3[21]$, $R_3[20]$ and $R_3[7]$). But since the tapping bits of R_3 are unknown, one cannot create the feedback bit. To create the feedback bit, one would have to guess the tapping bits of R_3 too, which would increase the complexity. Hence the approach had to be modified.

Solution: Let the register R_3 shift in front and no feedback needed after shifting of register. This was done for 11 clocking cycles of R_3. Hence, the total R_3 register was of the length 23+11=34 bits. The number 11 arises because at each point of determining R_3 bits, we could calculate $R_3[22]$ and $R_3[10]$ (i.e., MSB and clocking bit of R_3 respectively), with an interval of 11 bits between them. This 34-bit register was required to generate the initial R_3.

Note: Let t_3 denote the number of times register R_3 is clocked. During implementation, note:
At time $t_3 = 0$, MSB of R_3 is $R_3[11]$, clocking bit of R_3 is $R_3[23]$, total length of R_3 is 34 bits.
At time $t_3 = 11$, MSB of R_3 is $R_3[0]$, clocking bit of R_3 is $R_3[11]$, useful length of R_3 is 23 bits, total length is 34 bits.

Problem: During the implementation of the code, we encountered a problem of backtracking.

Discussion: We found a solution to implement backtracking in the code.
Implementation of backtracking was attempted in two different ways:
a) Using the concept of doubly-linked list in C programming language.
b) Using recursions in C++ programming language.

In the part (a), we used MALLOC function to allocate space for each bit at each step. According to graph theory of binary search tree algorithm, everytime we created three children from one parent, we stored all the information of the children in the parent. This used a lot of memory, eventually causing the program to give 'segmentation fault' (as this was tried on a normal PC with limited Memory space allocation). An alternate approach had to be taken to avoid this error.

Solution: Use recursions in C++ programming.

Implementation: We used all the above points to implement the modified Keller-Seitz attack, which does the cryptanalysis of the *A5/1* stream cipher for a fixed

set of R_1 and R_2 registers (all bits of R_1 and R_2 are guessed), and solves for R_3. The *most significant bit* (MSB) of R_3 was computed first according to the equation $R_3[22] = R_1[18] \oplus R_2[21] \oplus KS[i]$, followed by guessing the clocking bit of R_3. The method is described below in brief.

Methodology: Using the Modified Keller-Seitz approach, we check the clocking bit (CB) of R_1 and R_2 and the known keystream (KS) bits, to decide the CB of R_3. There are two possibilities for $R_1[8]$ and $R_2[10]$:

- Case 1: $R_1[8] = R_2[10]$
  If $R_1[8] = R_2[10]$, then R_1 and R_2 are surely going to get clocked by the majority function (clocking rule). $R_1[17]$ and $R_2[20]$ will become the new MSBs of R_1 and R_2 respectively after clocking. Hence the CB of R_3 has to be decided on $R_1[17]$, $R_2[20]$ and the KS bit after clocking. If $R_1[17] \oplus R_2[20] \oplus KS[i+1]$ bit does not equal the MSB of R_3, then R_3 has to be clocked in that round. This implies that the CB of R_3 will be equal to the CBs of R_1 and R_2. If $R_1[17] \oplus R_2[20] \oplus KS[i+1]$ equals the MSB of R_3, then R_3 may or may not be clocked. Consider one of the two possibilities first. If we encounter the contradiction later, we would come back to this state, change the CB of R_3 and proceed.

- Case 2: $R_1[8] \neq R_2[10]$
  If $R_1[8] \neq R_2[10]$, then there exist two possibilities; i.e., if $R_1[8] = R_3[10]$, then R_1 and R_3 are clocked and R_2 is not clocked, else if $R_2[10] = R_3[10]$, then R_2 and R_3 are clocked and R_1 is not clocked. Proceed until we encounter a contradiction. If contradiction occurs, discard the state candidate.

This method is carried out recursively till R_3 is clocked 11 times (i.e., $t_3 = 11$). Thus, we have determined the complete register R_3. Now perform the *post-processing-phase* to bit-wise check if the keystream bits generated by clocking the new state candidate match the known keystream bits. If it matches with the known keystream bits, then the state candidate obtained is the correct state candidate. If it does not match the known keystream then, we go back to the place where we encounter the contradiction, change the clocking bit of R_3 to the case which was not considered yet and continue determining the remaining bits of register R_3.

3.2 Known Plaintext Attack

The known plaintext attack [24] is an attack model where the attacker has access to both the plaintext and its encrypted ciphertext. This can be used to reveal the secret key used for encrypting the known plaintext to the known ciphertext.

3.2.1 Time Memory Trade-Off Attack (TMTOA)

In the known plaintext attack, if the objective is to recover the preceding internal states for any observed 64 successive *keystream bits*, one can do so by exhaustive search or brute force attack. It is then that we use the Time Memory Trade-Off Attack. This attack was introduced by Hellman [16].

Cryptanalytic attacks based on exhaustive search need a lot of computing power or a lot of time to completely implement the attack. When this attack has to be carried out multiple times, it may be possible to execute exhaustive search in advance and store all results in memory as pre-computed data. Once this pre-computation is carried out, the attack is instantaneous. But this is not practical because of large amounts of memory required. Hellman introduced a method to trade memory against attack time, by using pre-calculated data stored in memory. Thus, Hellman was able to bring out an optimal solution with this attack.

Let N = number of possible solutions, T = time needed, M = memory required, then

$$T \;=\; M \;=\; N^{\frac{2}{3}} \tag{3.1}$$

Protocol: We try to generate all possible ciphertexts in advance by encrypting the plaintext with all possible N keys. The ciphertexts are organized in chains, where only the first and last elements of the chain are stored in memory. This is the trade-off for this attack (i.e., saving memory space at the cost of cryptanalysis time). The chains are created using a reduction function R, which creates a key from the ciphertext. The ciphertext is longer in length than the key and so it is termed as reduction function. By successively applying encryption function E_k and reduction function R, we can create chains of alternating keys and ciphertexts as shown below.

$$K_i \xrightarrow{E_{k_i}(P_0)} C_i \xrightarrow{R(C_i)} K_{i+1}$$

Let $R(E_k(P_0)) = f(k)$, $\therefore$ this succession generates keys from a key, and so on. Hence giving us a chain of keys.

$$K_i \rightarrow K_{i+1} \rightarrow K_{i+2} \rightarrow \ldots$$

In this way, m chains of length t are created and their first and last columns are stored in a table. Given a ciphertext C, we can try to find out if the key used to generate C is among the ones used to generate the table. To do so, we generate a chain of keys starting with $R(C)$ upto length t. If C were indeed obtained with a key used while creating the table, then we would eventually generate the key that matches the last key of the corresponding chain. The last key has been stored in memory together with the first key of the chain. With the first key, the complete chain can be regenerated and in particular, the key that comes just before $R(C)$. This is the key that was initially used to generate the specific C.

Besides Golic [14] and Babbage [2], Biryukov-Shamir-Wagner [6] proposed an attack with a complexity of 2^{48} requiring about 300GB of memory, where the online phase of the attack can be executed within minutes with a 60% success probability. However, 2 seconds of known keystream (i.e., about 25000 bits) are required to perform the attack, making this attack impractical.

Barkan-Biham-Keller [4] also proposed another attack along these lines. However, in the precomputation phase of such an attack huge amounts of data need to be computed and stored. For example, with three minutes of ciphertext available, one needs to precompute about 50 TB of data to achieve a success probability of about 60%. These are practical obstacles making actual implementations of such attacks very difficult.

Chapter 4

A New Attack on the *A5/1*

This chapter deals with the description of our new attack algorithm. Our approach is based on the *guess-and-determine attack* proposed by Anderson [1], but with several modifications. The attack is divided into two phases: the *determination phase* and the *post-processing-phase*. With 64 bits of the keystream known and all bits of the shortest register R_1 guessed, we can determine all bits of the two longer registers R_2 and R_3. But unlike the approaches of Anderson [1], Golic [14], Biham-Dunkelman [5] and Keller-Seitz [19], we consider all possible cases i.e., no case is discarded.

4.1 The Attack

In this attack, all bits of the first register R_1 are known (guessed) and all bits of registers R_2 and R_3 are determined. We determine these bits based on 64 known *keystream* bits (KS). At the end, we come up with about $2^{48.5}$ possible state candidates, which is much smaller than the exhaustive search where we have 2^{64} state candidates (refer Section 4.4). Hence this attack is better than the exhaustive search approach. The minimum space complexity (lower bound) for the attack is approximately $2^{45.2}$, which is attained after 11 rounds of clocking. The attack consists of two phases, the *determination phase* and the *post-processing-phase*. The *determination phase* is again divided into two parts, the *processing-phase1* and the *processing-phase2*.

Assumptions: Register R_1 is completely guessed (known). 64 bits of keystream (KS) known.

Aim: Determine all bits of register R_2 and register R_3 (Figure 4.1).

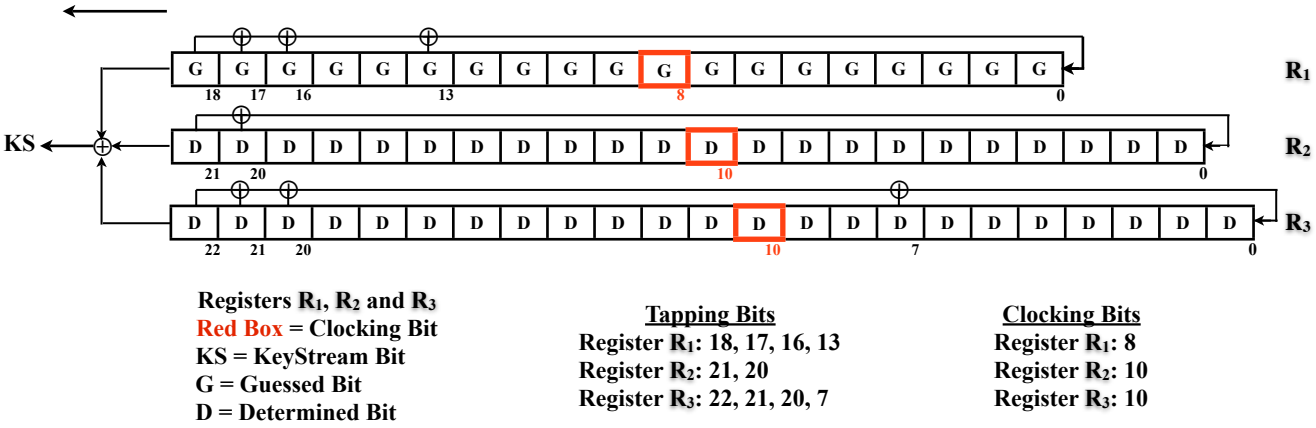

Figure 4.1: New Attack on the *A5/1* Stream Cipher

4.1.1 Determination Phase:

The *determination phase* generates all possible state candidates after the *warm-up phase* [7] is completed. Let t_2 and t_3 denote the number of times the registers R_2 and R_3 are clocked, respectively. Everytime a register is clocked, increase the counter of that register by one. Initialize the algorithm by giving the input of known keystream bits (KS) and guessing all bits of the smallest register R_1 (Figure 4.2).

1. **Processing-Phase1:** Compute the most significant bits (MSBs) of register R_2 and register R_3 using the MSB of register R_1 and KS bit by Equation 4.1:

$$KS[i] = R_1[18] \oplus R_2[21] \oplus R_3[22], \tag{4.1}$$

where $KS[i]$ denotes the i^{th} keystream bit, $i = 0$ on initialization and increases by 1 after every clocking round.

If the values of three of these bits are known, the fourth can be computed easily by the above equation. But if $R_2[21]$ and $R_3[22]$ are unknown, then there exist

two possible combinations of $R_2[21]$ and $R_3[22]$ bits. During initialization (refer Algorithm1), i is set to 0, and with every additional clocking round, the value of i increases by 1. Note that i also denotes the total number of clocking rounds that have taken place. There occur two possibilities:

- *If* $R_1[18] = KS[i]$, *then* $R_2[21] = R_3[22] = 0$ or $R_2[21] = R_3[22] = 1$.
- *If* $R_1[18] \neq KS[i]$, *then* $R_2[21] = 0$, $R_3[22] = 1$ or $R_2[21] = 1$, $R_3[22] = 0$.

If we did not use Equation 4.1, there will be four possible combinations for $R_2[21]$ and $R_3[22]$; i.e., 00, 01, 10 and 11. But the equation reduced the number of possibilities to two. This reduces the number of possible cases by half and the number of possible state candidates to half.

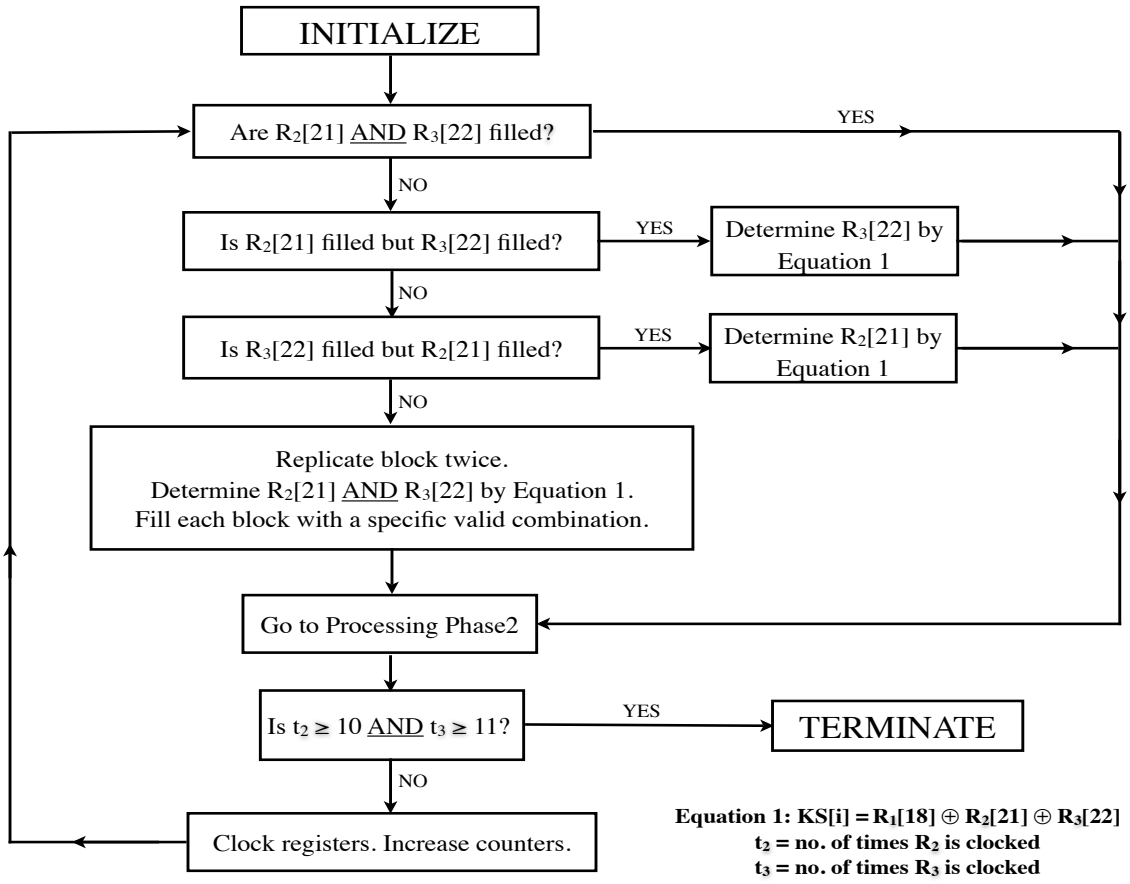

Figure 4.2: Determination Phase of the Attack (Processing-Phase1)

2. **Processing-Phase2:** Consider the *clocking bits* of registers R_2 and R_3. There are three possibilities:

   - *If* $R_2[10]$ is filled and $R_3[10]$ is vacant, *then* replicate the state candidate twice, fill one copy with $R_3[10] = 0$, and the other copy with $R_3[10] = 1$
   - *If* $R_2[10]$ is vacant and $R_3[10]$ is filled, *then* replicate the state candidate twice, fill one copy with $R_2[10] = 0$, and the other copy with $R_2[10] = 1$
   - *If* $R_2[10]$ and $R_3[10]$ are both vacant, *then* replicate the state candidate four times, fill the first copy with $R_2[10] = 0$, $R_3[10] = 0$; the second copy with $R_2[10] = 0$, $R_3[10] = 1$; the third copy with $R_2[10] = 1$, $R_3[10] = 0$; and the fourth copy with $R_2[10] = 1$, $R_3[10] = 1$.

   Thus, all possible combinations are taken into consideration (Figure 4.3).

   Now consider the bits $R_2[20]$ and $R_3[21]$. If registers R_2 and R_3 are clocked, then these bits will become the new MSBs for their respective registers after clocking. If both these bits are vacant, there are four possible combinations for these bits; i.e., 00, 01, 10 and 11. But the Equation 4.1 reduce them to two possibilities. This reduces the number of possible cases by half.

   If only one of these bits is vacant, there are two possibilities for the vacant bit; i.e., 0 or 1. But the above equation reduces them to only one possibility. For example, if $R_2[10] \neq R_1[8] = R_3[10]$, then $R_3[21] = R_1[17] \oplus KS[i+1] \oplus R_2[21]$. In this case, only $R_3[21]$ is unknown. This bit can be calculated by the above equation. Here, two possibilities for $R_3[21]$ reduce to only one possibility. This reduces the number of cases by half.

   Follow this protocol till $t_2 < 10$ and $t_3 < 11$. Once this condition is not satisfied, i.e., the first time $t_2 \geq 10$ and $t_3 \geq 11$, stop. At this moment, registers R_2 and R_3 are completely determined for the known KS and register R_1. The number of bits between the clocking bit (CB) and the MSB for register R_2 is 10 and for register R_3 is 11. Hence, register R_2 has to be clocked atleast 10 times and register R_3 has to be clocked atleast 11 times to determine all the bits of that register.

   A *complete state candidate* is a state candidate with all bits filled. The minimum number of KS bits required to obtain a set of complete state candidates is eleven.

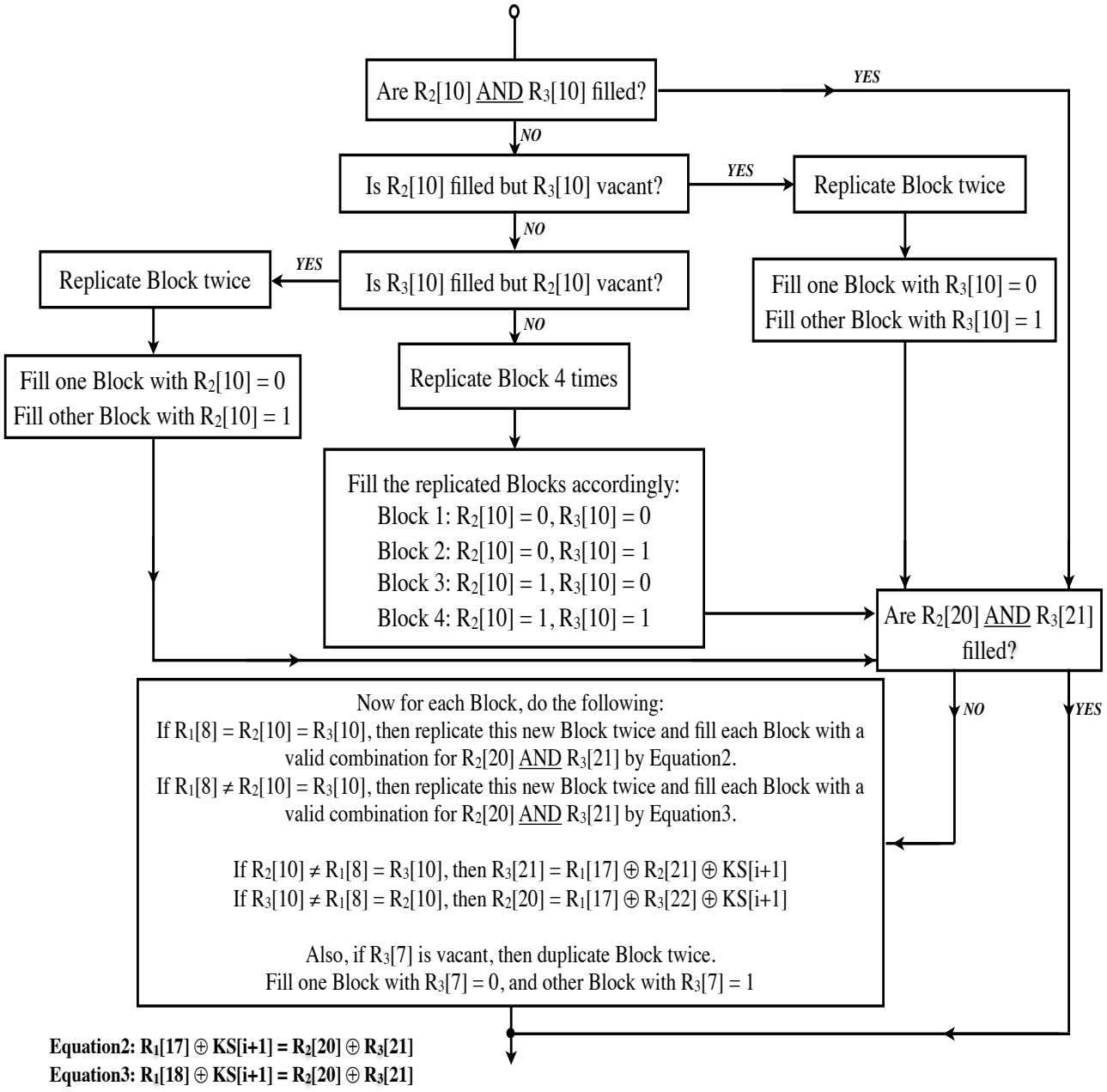

Figure 4.3: Determination Phase of the Attack (Processing-Phase2)

This will happen when both registers R_2 and R_3 are clocked together for 10 consecutive clocking cycles and register R_3 is clocked again in the next round.

4.1.2 Post-Processing-Phase:

The post-processing-phase checks a complete state candidate obtained after the determination phase is the correct state candidate i.e., the key (refer Algorithm 3). As discussed in the preceding subsection, the minimum number of rounds needed to perform the post-processing-phase to obtain a set of complete state candidates is 11. The number of possible state candidates increases with every additional round. Hence, the probabilitity of finding the correct state candidate increases with every additional round.

In this phase we generate output bits by performing normal *A5/1* encryption with each of the complete state candidates obtained from the *determination phase*. Match these output bits bit-wise with the known KS bits. If the KS bits and output bits match, continue clocking and generating output bits till a contradiction of bit-wise matching occurs. If all the output bits match with the given 64 KS bits and no contradiction occurs, then that state candidate is the correct state candidate. Hence, we have found the correct state candidate amongst all the possible state candidates obtained.

4.2 The Attack Algorithms

The algorithms for the implementation of the attack are listed in this section.

Algorithm 1 Determination Phase: Processing-Phase1

Input: All bits of register R_1 are known, 64 keystream (KS) bits are known.
Aim: Determine all bits of registers R_2 and R_3. Hence, determine all possible valid state candidates.

1: **Initialize** $t_2 = 0; t_3 = 0; i = 0;$
2: **while** $R_2[21]$ vacant $\underline{\text{OR}}$ $R_3[22]$ vacant **do**
3: replicate internal state candidate ▷ refer Figure 4.2
4: compute $R_2[21]$ $\underline{\text{AND}}$ $R_3[22]$ by Equation 4.1: $KS[i] = R_1[18] \oplus R_2[21] \oplus R_3[22]$
5: **end while** ▷ $R_2[21]$ $\underline{\text{AND}}$ $R_3[22]$ determined
6: goto Processing-Phase2 ▷ Algorithm 2
7: **if** $t_2 \neq 10$ $\underline{\text{OR}}$ $t_3 \neq 11$ **then**
8: apply clocking rule
9: $i \leftarrow i + 1$
10: **if** R_2 is clocked **then**
11: $t_2 \leftarrow t_2 + 1$
12: **else if** R_3 is clocked **then**
13: $t_3 \leftarrow t_3 + 1$
14: goto step 2
15: **end if**
16: **end if**
17: **terminate** ▷ All possible VALID state candidates determined

Algorithm 2 Determination Phase: Processing-Phase2

Input: All bits of register R_1 are known, 64 keystream (KS) bits are known.
Aim: Determine complete registers R_2 and R_3. Hence, determine all possible valid state candidates.

1: **while** $R_2[10]$ vacant OR $R_3[10]$ vacant **do**
2: replicate internal state candidate accordingly ▷ refer Figure 4.3
3: fill each replication with all cases for $R_2[10]$ AND $R_3[10]$
4: **end while** ▷ $R_2[10]$ AND $R_3[10]$ determined
5: **while** $R_2[20]$ OR $R_3[21]$ vacant **do**
6: replicate internal state candidate
7: **if** $R_1[8] = R_2[10] = R_3[10]$ **then**
8: compute $R_2[20]$ AND $R_3[21]$ by Equation1: $KS[i+1] \oplus R_1[17] = R_2[20] \oplus R_3[21]$
9: **else if** $R_1[8] \neq R_2[10] = R_3[10]$ **then**
10: compute $R_2[20]$ AND $R_3[21]$ by Equation2: $KS[i+1] \oplus R_1[18] = R_2[20] \oplus R_3[21]$
11: **else if** $R_2[10] \neq R_1[8] = R_3[10]$ **then**
12: compute $R_3[21]$ by the Equation: $KS[i+1] \oplus R_1[17] \oplus R_2[21] = R_3[21]$
13: **else if** $R_1[8] = R_2[10] \neq R_3[10]$ **then**
14: compute $R_2[20]$ by the Equation: $KS[i+1] \oplus R_1[17] \oplus R_3[22] = R_2[20]$
15: **end if**
16: **end while** ▷ $R_2[20]$ AND $R_3[21]$ determined
17: **while** $R_3[7]$ vacant **do**
18: replicate internal state candidate two times
19: fill one replication with value 0 and the other replication with value 1
20: **end while** ▷ $R_3[7]$ determined

Algorithm 3 Post-processing Phase

Input: All possible valid state candidates derived from Algorithm1 and Algorithm2 after the *determination phase*. 64 keystream bits are known.
Aim: Determine the correct state candidate

1: **while** $i \neq 63$ **do**
2: **if** $R_1[18] \oplus R_2[21] \oplus R_3[22] = KS[i]$ **then**
3: **continue**
4: $i \leftarrow i + 1$
5: apply clocking rule
6: **else**
7: discard current state candidate ▷ contradiction during bit-wise matching with original KS
8: check next state candidate
9: **end if**
10: **end while**
11: **terminate** ▷ correct state candidate found

4.3 Analysis of the Attack

We now discuss each phase of the attack step-by-step. After initialization, we perform the first step of implementation, i.e., the *determination phase*. The state candidate has all bits of register R_1 known and all other bits vacant. According to protocol (refer Algorithm 1 and Algorithm 2), the determination phase determines the most significant bits (MSBs) of registers R_2 and R_3 (i.e., $R_2[21]$ and $R_3[22]$) by processing-phase1; the clocking bits of R_2 and R_3 (i.e., $R_2[10]$ and $R_3[10]$), bit $R_3[7]$ and if possible, bits $R_2[20]$ and $R_3[21]$ by processing-phase2.

Now we consider the first stage of the determination phase i.e., processing-phase1. The MSBs of registers R_2 and R_3 have to be determined. The number of possible combinations reduces from four to two by Equation 4.1. Thus saving two combinations, i.e., a saving of 50%. During the implementation of further rounds, there is a possibility where only one of the MSBs of R_2 or R_3 is vacant. We determine these vacant bit(s) by Equation 4.1.

We now proceed to processing-phase2 of the determination phase. Here we first consider the four vacant bits: $R_2[10]$ (CB of R_2), $R_3[10]$ (CB of R_3), $R_2[20]$ and $R_3[21]$. But all these four bits (except the first step after initialization) may not

be vacant together at all times. In the following table (Figure 4.4), we consider all possible cases of these four bits being empty, and the number of maximum possible valid combinations that exist as a result of Equation 4.1. We now consider the bit $R_3[7]$. There are two possibilities for this bit, i.e., 0 and 1. But we cannot eliminate any case by any method. Hence, we need to consider both cases.

EMPTY?				POSSIBLE CASES	MAX. POSSIBLE VALID CASES	% SAVE
CB2	$R_2[20]$	CB3	$R_3[21]$			
✓	✓	✓	✓	16	6	62.5
✓	✓	✓	-	8	NA	NA
✓	✓	-	✓	8	3	62.5
✓	-	✓	✓	8	4	50
-	✓	✓	✓	8	3	62.5
✓	✓	-	-	4	2	50
✓	-	✓	-	4	NA	NA
✓	-	-	✓	4	2	50
-	✓	-	✓	4	2	50
-	✓	✓	-	4	NA	NA
-	✓	-	✓	4	2	50
-	-	✓	✓	4	2	50
✓	-	-	-	2	2	0
-	✓	-	-	2	1	50
-	-	✓	-	2	NA	NA
-	-	-	✓	2	1	50
-	-	-	-	0	0	0

Figure 4.4: All possibilities during Processing-Phase2

Whenever the CB of register R_3 is vacant, the bit $R_3[21]$ has to be vacant too. Hence there are some cases in the following table which are *not applicable* (NA). The last column depicts the percentage of the total possible cases that are discarded due to the attack algorithms.

In the determination phase, a total of 7 bits have to be determined. These 7 bits would have $2^7 = 128$ possible combinations. But our algorithms give only 24 valid

possible combinations. Thus saving 104 combinations i.e., a saving of 81.25%. The number of possible state candidates where register R_3 is not clocked is four.

Consider an example to implement the processing-phase2 of the determination phase of the attack. Let register R_1 be completely filled (guess). Also known are some bits of the keystream (KS). We have to determine the six vacant bits: $R_2[21]$, $R_2[20]$, $R_2[10]$, $R_3[22]$, $R_3[21]$ and $R_3[10]$ according to the attack algorithms. Let the bits $R_2[21] = a$, $R_2[20] = b$, $R_2[10] = c$, $R_3[22] = d$, $R_3[21] = e$ and $R_3[10] = f$. Known KS bits (two bits) = 11. Since we are in the processing-phase2 of the determination phase, the MSBs of registers R_2 and R_3 are already determined (in the processing-phase1). Let the determined bits be $R_2[21] = a = 0$ and $R_3[22] = d = 0$ (Figure 4.5)

We now have four vacant bits. These vacant bits would give rise to $2^4 = 16$ possible combinations to fill them. But our attack algorithm (Algorithm 2) gives

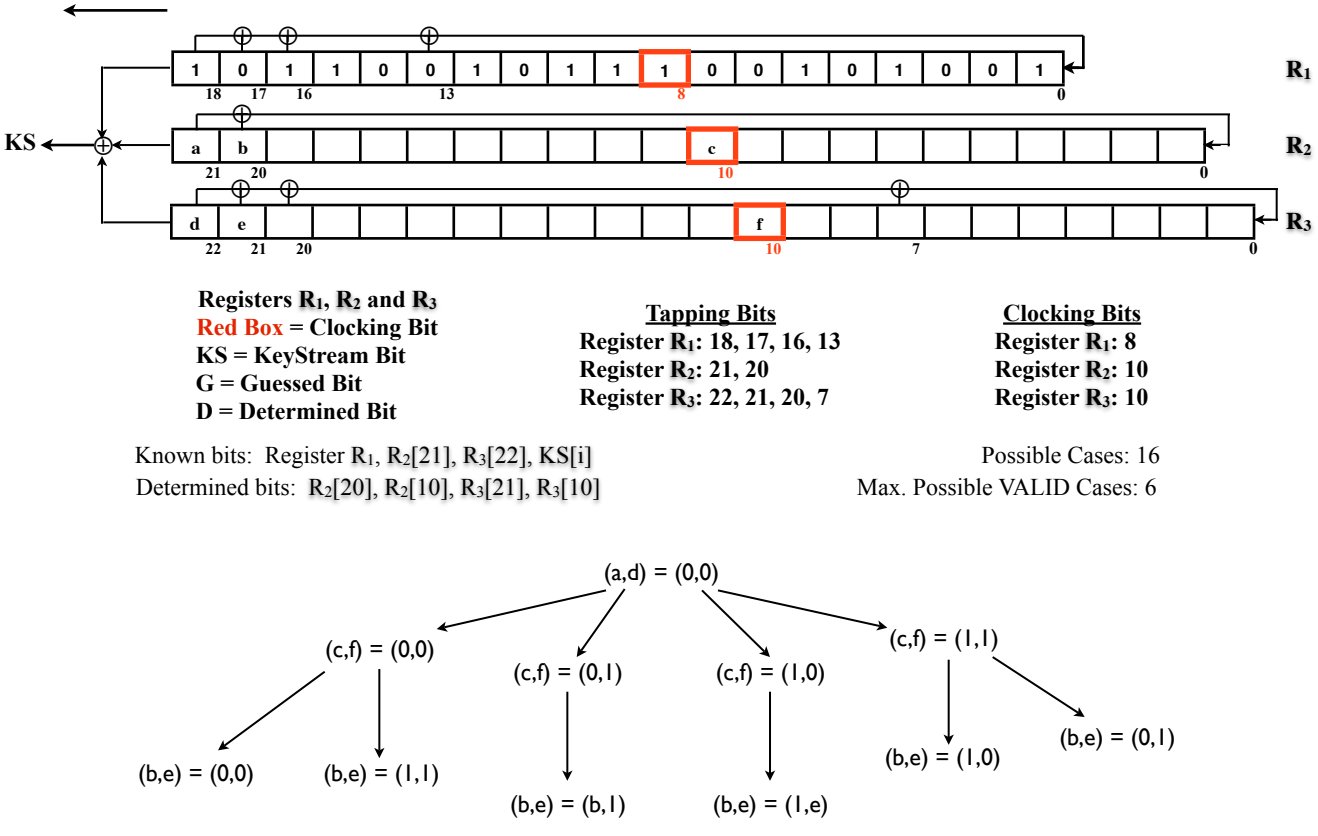

Figure 4.5: Implementation of the Processing-Phase2 (Determination Phase)

only 6 valid cases. The remaining 10 possibilities are discarded. Thus saving 62.5% of the total possible combinations.

In the above example, we have considered only the implementation of processing-phase2 of the determination phase, assuming processing-phase1 is completed. But, we now consider the implementation of the processing-phase1 too. After initialization, we perform the processing-phase1 of the determination phase. At this stage, the state candidate has only register R_1 completely filled. The MSBs of R_2 and R_3 (i.e., $R_2[21]$ and $R_3[22]$) are still vacant. We now implement both parts of the determination phase (Figure 4.6).

Consider the following six vacant bits:$R_2[21]$, $R_2[20]$, $R_2[10]$, $R_3[22]$, $R_3[21]$ and $R_3[10]$. These vacant bits would give rise to $2^6 = 64$ possible combinations to fill

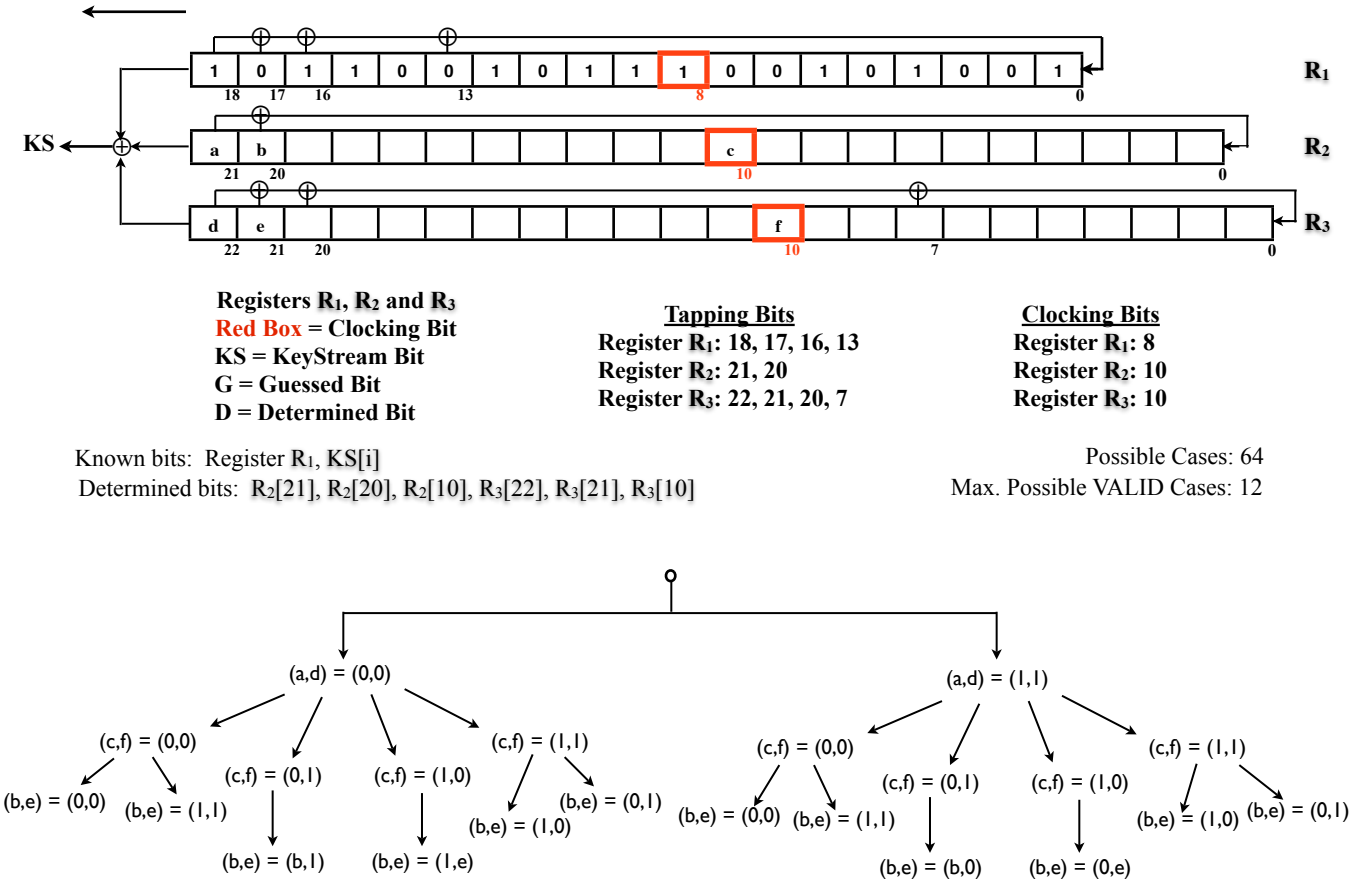

Figure 4.6: Implementation of the Complete Determination Phase

them. But the attack protocol gives only 12 valid cases. The remaining 52 possibile cases are discarded. Thus saving 81.25% of the total possible combinations.

If $R_3[7]$ is not considered, the first round of implementation will always generate 12 state candidates. On an average, the second round generates 60 state candidates and the third round generates 300 state candidates. The number of state candidates (till round 10) can be approximated by the formula $12 * 5^{n-1}$, where n denotes the n^{th} round, $n \in \mathbb{Z}^+$, $n < 11$. It is only after the 11^{th} round that we will get the first set of complete state candidates (with all registers full). When bit $R_3[7]$ is taken into consideration, the first round of implementation will always generate 24 state candidates. From round three to round ten, the number of possible state candidates after every round is approximately five times the total number in the previous round.

4.4 Discussion

In this section, we discuss in detail a probabilistic approach to determine the time complexity, data complexity, memory complexity and the success probability of the new attack. The results of this probabilistic approach are also corroborated by experimental data. According to these results, the average number of rounds necessary to get the correct state candidate (key) is 15.5 and the average number of complete state candidates obtained after 15.5 rounds is $2^{48.5}$.

4.4.1 Time complexity

The number of bits between the clocking bit (CB) and the most significant bit (MSB) for register R_2 is 10 and for register R_3 is 11. Hence, the number of times the registers R_2 and R_3 have to be clocked to determine all the bits of that register is atleast 10 and 11 respectively. The minimum number of KS bits required to obtain a set of complete state candidates (with no vacant bits) is 11. This will occur when both registers R_2 and R_3 are clocked together for 10 consecutive clocking cycles and register R_3 is clocked again in the following round.

With every clocking round, the number of complete state candidates increases. Hence, the probability of finding the correct state candidate increases with every round of clocking.

According to the *majority function* of the *clocking rule* for the *A5/1* (Figure 2.2), a register will get clocked 3 out of 4 times. At every clocking cycle, atleast two

registers will get clocked. Let n_1 be the event that registers R_2 and R_3 are clocked together, and n_2 be the event that register R_1 is clocked either with register R_2 or with register R_3. The probabilities that events n_1 and n_2 occur are given by $P(n_1)$ and $P(n_2)$ respectively. Let n_2' be the event that only registers R_1 and R_2 are clocked. Let n_2'' be the event that only registers R_1 and R_3 are clocked. The probabilities that events n_2' and n_2'' occur are given by $P(n_2')$ and $P(n_2'')$ respectively. From Figure 2.2, one can conclude that $P(n_1) = P(n_2) = \frac{1}{2}$. Thus,

$$P(n_1) + P(n_2) = \frac{1}{2} + \frac{1}{2} = 1$$

and

$$P(n_2) = P(n_2') + P(n_2'') = \frac{1}{4} + \frac{1}{4}.$$

i.e.,

$$P(n_1) + P(n_2') + P(n_2'') = 1$$

Registers R_2 and R_3 have to be clocked atleast 10 and 11 times respectively to determine all bits of that register i.e., to obtain a set of complete state candidates. They may be clocked by n_1 or n_2.

Let X be the random variable denoting the number of clocking cycles needed to obtain complete state candidate. Let x_1 be the number of clocking cycles needed for event n_1, x_2 for n_2' and x_3 for n_2''. Here, $x_1 = 10$, $x_2 = 10$ and $x_3 = 11$. Then the expectation for this variable X is defined as

$$E[X] = \frac{x_1 * P(n_1) + x_2 * P(n_2') + x_3 * P(n_2'')}{P(n_1) + P(n_2') + P(n_2'')}$$

$$\implies \frac{10 * \frac{1}{2} + 2 * (10 * \frac{1}{4} + 11 * \frac{1}{4})}{\frac{1}{2} + \frac{1}{4} + \frac{1}{4}} = 15.5$$

An Experiment

We now perform normal encryption of *A5/1* using random inputs for all three registers. The aim of this experiment is to determine the average number of clocking rounds needed for register R_2 and R_3 to be clocked atleast 10 and 11 times respectively. We performed this experiment thrice, each time with 250 inputs. The average number of clocking rounds needed turned out to be 15.51 with a standard deviation of 1.785. Hence, the experimental results corroborate with the theoretical proof.

The number of clocking rounds necessary to obtain a set of complete state candidates is approximately 15.5. This set of complete state candidates would contain the correct state candidate (key) with a high probability.

4.4.2 Data Complexity

The minimum number of KS bits required to generate a set of complete state candidates (all bits filled) is 11. With every additional clocking round, the number of complete state candidates increase. We can perform the post-processing-phase of the attack after every round simultaneously while performing the determination phase of the next round. Hence, the probability of finding the correct state candidate also increases with every round. But we require atleast 64 KS bits for the post-processing-phase of the attack to bit-wise match and check for the correct state candidate.

4.4.3 Space Complexity and Success Probability

The number of possible state candidates obtained after the 11^{th} round is approximately $2^{45.2}$. The total number of complete state candidates is approximately $2^{39.2}$ i.e., around 1.6% of total number of possible state candidates obtained after the 11^{th} round. Thus, the minimum complexity of the atack (lower bound) is around 2^{40}. As stated earlier, the number of possible state candidates increases with every round. Here we plot a table (Table 4.1) of the results of the experimental data. The four columns of the table are: number of clocking rounds; total number of state candidates obtained after that round; total number of complete state candidates obtained; and percentage of number of complete state candidates over the total number of state candidates for that round. All values of the experimental data in the table are approximated to one decimal place.

Table 4.1: Space Complexity and Success Probability

No. of Rounds	Total State Candidates	Complete State Candidates	$\frac{Complete}{Total} * 100$
11	$2^{45.2}$	$2^{39.2}$	1.6%
12	$2^{46.0}$	$2^{42.5}$	9.0%
13	$2^{46.7}$	$2^{44.5}$	22%
14	$2^{46.9}$	$2^{45.3}$	30%
15	$2^{47.1}$	$2^{46.1}$	50%

Remark: In each round, the number of possible choices reduce to atleast half in each case (refer Figure 4.4). Hence, a minimum saving of 50% takes place in every round. As stated in Section 4.4.1, the average number of rounds to get the correct state candidate (key) is around 15.5. In each round, we save atleast half the possible cases. Hence for 15.5 rounds we save $(\frac{1}{2})^{15.5}$ cases. Thus, the average number of complete state candidates obtained after 15.5 rounds will be $(2^{64})(\frac{1}{2})^{15.5} = 2^{48.5}$. The correct state candidate (key) would be amongst the set of complete state candidates obtained after the 15th round, with a probability of 50%.

4.5 Conclusion

Our attack is based on the *guess-and-determine* approach proposed by Anderson [1], but with several modifications. In this attack, all bits of the first register R_1 are known (guessed) and all bits of registers R_2 and R_3 are determined. We determine these bits based on 64 known *keystream* bits (KS). This attack has an average space complexity of $2^{48.5}$. This is much smaller than the exhaustive search where the space complexity is 2^{64}. Hence this attack is better than the exhaustive search approach. The minimum space complexity (lower bound) for the attack is approximately 2^{40}, which is attained after 11 rounds of clocking. The average number of rounds necessary to obtain the correct key from the set of complete state candidates is 15.5. The probability of success in finding the key after 15 rounds, by post-processing phase, is more than 50%. With every round of clocking after 11 rounds, the number of complete state candidates increases. Thus, the probability of finding the correct state candidate increases with every clocking round. The attack is successful with 100% probability.

Chapter 5

Further Analysis of the *A5/1*

In this chapter, we explore in detail certain weaknesses that we discovered during our study of the *A5/1* stream cipher. We then suggest certain modifications to the existing stream cipher aiming towards a more secure cryptosystem resistant to most of the attacks already known. We end this chapter with results stating that these modified stream ciphers are indeed pseudo random number generators too, like the *A5/1* stream cipher.

5.1 Weaknesses of the *A5/1*

Here we state certain weaknesses in the *A5/1* stream cipher and our views on them:

1. The *A5/1* stream cipher has three components. An ideal stream cipher would contain only one component. Hence the concreteness and solidity of the *A5/1* security is reduced.

   Remark: Since we would not like to change the basic skeletal structure of the *A5/1*, we would not modify or change it.

2. The registers are too short in length, i.e., 19, 22 and 23 bits only.

   Remark: Although the registers are short in length, they are three in number and they are aptly divided to give a total of 64 bits.

3. The stream cipher is of only 64 bits; hence an attack can be manipulated on it in real time.

Remark: The criteria for acceptance of stream ciphers is 64 bits, 80 bits or 128 bits only (discussed by Eric Zenner [30]). This was stated in the eSTREAM project organized by the EU ECRYPT network [9].

4. Register R_2 of length 22 has only 2 tapping bits, and both are adjacent to each other. This property can be exploited and made useful for an attack on the stream cipher.

   Remark: Our views on this match with those already discussed by Elad Barkan [3] where he states:
   a) Register R_2 has 2 adjacent tapping bits.
   b) Clocking bit of register R_2 is exactly in the center.
   c) A correlation equation can be formed from register R_2 useful for further attack i.e., $S_{(i)} + S_{(i-1)} = S_{(i+22)}$

5. A major issue in the *A5/1* security is the short period of the cipher, which is based on the three LFSRs. Without stop/go operation, the period of sum of the three LFSRs is given by: $(2^{19}-1)(2^{22}-1)(2^{23}-1)$. However, experiments show that the period of *A5/1* is roughly around $\frac{4}{3}(2^{23}-1)$ [20]. Hence, we would like to create a modified *A5/1* stream cipher with a period greater than or equal to the existing one.

6. The clocking bits should be intermixed with the clocking bits of all three registers to provide increased randomness. Hence making an attack on the cipher more difficult.

7. The majority function is not a good function in terms of correlation with all affine functions.

   Remark: We put forward the idea of introducing irregular clocking or motor clocking to the LFSRs, (for example, when the majority function is equal to 1, then clock the respective registers twice; else clock the respective registers once). This will make sure the majority function is not the only criteria essential for encryption. Motor clocking would further increase the security of the stream ciphers. This shall be discussed in the next section.

8. A slightly greater modification can be suggested. One could keep the feedback bit of each register as a combination and dependence of the other two registers

too. This would modify the basic skeletal structure of the *A5* encryption model, but would increase the security of the cryptosystem immensely; rendering the existing attacks not so much of use.

Note: It is on points 5, 6 and 7 that we have extensively worked on, with an aim to make a more secure cryptosystem.

5.2 Modifications Suggested to the *A5/1*

Erguler-Anarim [11] have suggested a few modifications to the *A5* with a *match-rule*. We suggest a few different ideas and modifications for the security enhancement of the *A5/1* stream ciphers, making the encryption scheme resistant to most of the attacks already known. The basic skeletal structure of the *A5/1* is kept the same in these modifications.

1. **Encrypt1:** The clocking pattern of the *A5/1* encryption algorithm is based on the majority function of the clocking bits (CBs). According to this, if the CBs match the majority function, the respective registers will get clocked once. If not, then that register is not clocked for that round. Atleast two registers will get clocked in each round. The attacks proposed by Anderson [1], Golic [14], Keller-Seitz [19] and Gendrullis-Novotny-Rupp [13] exploit this clocking pattern. These attacks can be avoided by the following modification. Let the tapping bits be the same as in the original *A5/1* stream cipher. Let the clocking pattern of the modified *A5/1* stream cipher (Encrypt1) be as follows:

   - Case 1: CB1 = CB2 $\neq$ CB3 (Clock R_1 and R_2 twice)
   - Case 2: CB1 $\neq$ CB2 = CB3 (Clock R_2 and R_3 twice)
   - Case 3: CB1 = CB3 $\neq$ CB2 (Clock R_1 and R_3 twice)
   - Case 4: CB1 = CB2 = CB3 (Clock R_1, R_2 and R_3 once),

   where CBi denotes the clocking bit for register R_i; $i = 1, 2, 3$.

   This motor clocking pattern would ensure irregular clocking. In the case where two out of three registers are getting clocked, the bit next to the most significant bits will never be XORed to give the keystream bit. Hence, according to the protocol of the attacks mentioned above, that bit can never be determined.

That bit will play a crucial role in determining the feedback bit (Bit 0) for the registers to be clocked. Hence the attacks would require many more rounds which would add to the complexity and increase the computation space and time. Also, there would be a case when all three registers are never clocked for consecutive rounds. In that case, computing the feedback bit becomes very difficult. Thus, the modified *A5/1* becomes more resitant to the known attacks.

Note: Several variants of this type of motor clocking can be designed to make the encryption scheme more resistant to the known attacks.

2. **Encrypt2:** The newly proposed stream cipher has the same three LFSRs but with different *maximal length tap sequences* [22] (Refer Section 2.3).
(i). A register of length 19 has a maximal length period of 524287. Some of the tapping bits leading to this maximal period are:
18, 17, 16, 13 (this one is currently used in *A5/1*)
18, 17, 16, 4
18, 15, 11, 8
18, 15, 8, 4 (we suggest this)

Since the period of the register is the same irrespective of the above-mentioned tapping bits combination, we suggest the register R_1 of length 19 to have tapping bits 18, 15, 8 and 4. This will be the modified register R_1. By doing so, several flaws encountered in the original register can be dealt with. We know that the clocking bit of register R_1 is bit 8; hence, the security of the register increases if we include the clocking bit as one of the tapping bits. By including bit $R_1[4]$ as a tapping bit, we have spaced out the tapping bits on both sides of the clocking bit, making the Biham-Dunkelman attack [5], which exploits this flaw, difficult.

(ii). A register of length 22 has a maximal length period of 4194303. The existing register R_2 in the *A5/1* scheme has the tapping bits 21, 20. We suggest a modification in this with tapping bits 22, 19, 10 and 2. It has already been discussed that there lay some flaws in the register R_2 with 22 bits, as the tapping bits are too close to each other and only two in number. We suggest changing the tapping bits of register R_2 from bits 21, 20 to bits 21, 19, 10, 2. This new register will have the same maximal length period of 4194303. The tapping bits will now be well spaced out, i.e., on either side of the clocking bit. The clocking bit for this register is bit 10, and if we include the clocking bit as a tapping bit

too, it would make the attack on *A5/1* more difficult.

(iii). A register of length 23 has a maximal length period of 8388607. The existing register R_2 of *A5/1* has the tapping bits as 22, 21, 20, 7. We suggest a modification in this to tapping bits 22, 20, 10 and 6. This modified register also has the same period as the original register R_3 in the *A5/1* scheme, and also includes the clocking bit of register R_3.

Suggested Modifications (Figure 5.1):
Encrypt0 = *A5/1* with no changes in tapping bits and clocking pattern
Encrypt1 = *A5/1* with motor clocking
Encrypt2 = *A5/1* with tapping bit changes

5.3 Tests for Random Number Generators

A random bit generator is a device or an algorithm that generates an output sequence of statistically independent and unbiased binary digits. The *A5/1* stream cipher is a pseudorandom bit generator where the input to this generator is called *seed* and the output is a *pseudorandom bit sequence*.

Here, we test the existing *A5/1* to check if it truly is a pseudorandom number generator (PRNG). We shall also test if the newly proposed modified *A5/1* are PRNGs.

Encrypt0, Encrypt1 and Encrypt2 pass the DIEHARD Battery of Tests of Randomness [21] and NIST Test Suites for random number generators [25]. Hence all three stream ciphers are pseudorandom number generators.

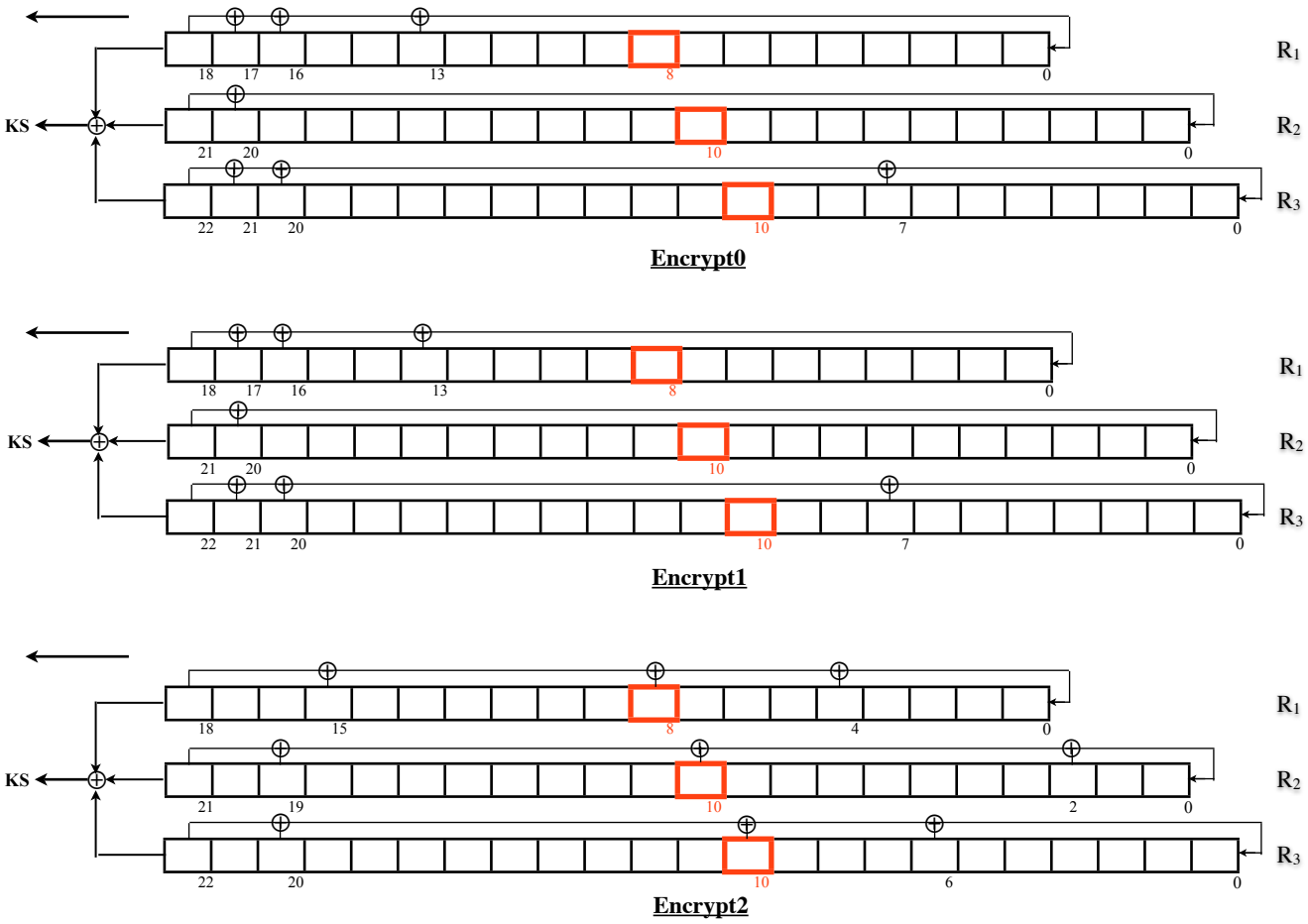

Figure 5.1: The Suggested Variants of the *A5/1*

Bibliography

[1] Anderson, R., A5 (was: Hacking digital phones), http://yarchive.net/phone/gsmcipher.html, Newsgroup Communication, 1994.

[2] Babbage, S., *A Space/Time Tradeoff in Exhaustive Search Attacks on Stream Ciphers*, European Convention on Security and Detection, 1995.

[3] Barkan, E., *Cryptanalysis of Ciphers and Protocols*, PhD Research Thesis, Israel Institute of Technology, Haifa, 2006.

[4] Barkan, E., Biham, E., and Keller, N., *Instant Ciphertext-only Cryptanalysis of GSM Encrypted Communication*, Technical Report CS-2006-07, Technion, 2006.

[5] Biham, E. and Dunkelman, O., *Cryptanalysis of the A5/1 GSM Stream Cipher*, In Proc. of Indocrypt'00, vol 1977 of LNCS. Springer-Verlag, 2000.

[6] Biryukov, A., Shamir, A. and Wagner, D., *Real Time Cryptanalysis of A5/1 on a PC*, In Proc. of FSE'00, vol 1978 of LNCS, pp 1-18. Springer-Verlag, 2001.

[7] Briceno, M., Goldberg, I., and Wagner, D., *A Pedagogical Implementation of the GSM A5/1 and A5/2 "voice privacy" Encryption Algorithms*, http://cryptome.org/gsm-a512.html, 1999.

[8] Donald, E., *Introduction to Combinatorial Algorithms and Boolean functions. The Art of Computer Programming. 4.0*, Upper Saddle River, NJ: Addison-Wesley. pp 64-74. ISBN 0321534964, 2008.

[9] ECRYPT, Call for Stream Cipher Primitives, available online at http://www.ecrypt.eu.org/stream/call

[10] Ekdahl, P. and Johansson, T., *Another Attack on A5/1*, IEEE Transactions on Information Theory, 49(1), pp 284-289, 2003.

[11] Erguler, I., and Anarim, E., *A Modified Stream Generator for the GSM Encryption Algorithms A5/1 and A5/2*, EUSIPCO, 2005.

[12] Gendrullis, T., *Hardware-Based Cryptanalysis of the GSM A5/1 Encryption Algorithm*, Diploma thesis, Ruhr-University Bochum 2008.

[13] Gendrullis, T., Novotny, M., and Rupp, A., *A Real-World Attack Breaking A5/1 within Hours*, Proc. of CHES'08, vol 5154 of LNCS, pp 266-282. Springer-Verlag, 2008.

[14] Golic, J., *Cryptanalysis of Alleged A5 Stream Cipher*, In Proc. of Eurocrypt'97, vol 1233 of LNCS, pp 239-255. Springer-Verlag, 1997.

[15] Golomb, S., *Shift Register Sequences*, San Francisco, Holden-Day, ISBN 08941220484, 1967.

[16] Hellman, D., *A Cryptanalytic Time-Memory Tradeoff*, IEEE Trans. on Info. Theory, vol 26, pp 401-406, 1980.

[17] Hillebrand, Friedhelm, *GSM and UMTS: The creation of Global Mobile Communication*, Wiley 2002, ISBN 0470843325.

[18] Kasper, E., *Complexity Analysis of Hardware-Assisted Attacks on A5/1*, Master's Thesis, University of Tartu, 2006.

[19] Keller, J., and Seitz, B., *A Hardware-Based Attack on the A5/1 Stream Cipher*, http://pv.fernuni-hagen.de/docs/apc2001-final.pdf, 2001.

[20] Kostopoulos, G., Sklavos, N., Galanis, M., and Koufopavlou, O., *VLSI Implementation of GSM Security: A5/1 and W7 Ciphers*, In Proc. of IEEE Workshop on Wireless Circuits and Systems (IEEE WoWCAS'04), Canada, 2004.

[21] Marsaglia G., DIEHARD: a battery of tests of randomness, Available online at http://stat.fsu.edu/ ego/diehard.html, 1996.

[22] Maximal Length tap Sequences, available online at http://homepage.mac.com/afj/taplist.html

[23] Maximov, A., Johnasson, T. and Babbage, S., *An Improved Correlation Attack on A5/1*, In Proc. of SAC'04, vol 3357 of LNCS, pp 239-255. Springer-Verlag, 2005.

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

[25] National Institute of Standards and Technology (NIST), Available online at http://csrc.nist.gov/groups/ST/toolkit/rng/batteries_stats_test.html

[26] Paar, C., and Pelzl, J., *Understanding Cryptography: A Textbook for Students and Practitioners*, Springer 2010, ISBN 9783642041006

[27] Pornin, T., and Stern, J., *Software-hardware Trade-offs: Application to A5/1 Cryptanalysis*, Proc. of CHES'00, vol 1965 of LNCS, pp 318-327, Springer-Verlag, 2000.

[28] Schneier, B., *Applied Cryptography: Protocols, Algorithms and Source Code in C*, Wiley 2009, ISBN 9788126513680

[29] Stamp, M., and Low, R., *Applied Cryptanalysis: Breaking Ciphers in the Real World*, Wiley 2007, ISBN 9780470114865

[30] Zenner, E., *Stream Cipher Criteria*, eSTREAM ECRYPT, Report 2006/032, 2006, http://www.ecrypt.eu.org/stream