Insert here your thesis' task.

Czech Technical University in Prague
Faculty of Information Technology
Department of Computer Systems

Bachelor's thesis

Correlation Attack on A5/1

Martin Holec

Supervisor: Mgr. Martin Jureček

12th May 2015

Acknowledgements

I would like to thank to my supervisor for his leadership, understanding and his willingness to help and also his criticism.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on 12th May 2015 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Holec, Martin. *Correlation Attack on A5/1.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2015.

Abstrakt

Popis šifry A5/1, základních kryptografických útoků a rozbor útoku korelační metodou. Tato práce bude prospěšná každému, kdo se bude zajímat o proudové šifry, základní myšlenky kryptografických útoků a speciálně myšlenku útoku korelační metodou aplikovanou na šifru A5/1.

Klíčová slova A5/1, korelační metoda, útok na šifru, proudová šifra, LFSR

Abstract

Description of A5/1 cipher, basics of cryptographic attacks and analysis of correlation attack. This thesis is helpful for everyone who is intrested in stream ciphers, basic ideas of cryptographic attacks and correlation attack on A5/1 in particular.

Keywords A5/1, correlation, attack on cipher, stream cipher, LFSR

Contents

List of Figures

List of Tables

Introduction

Mobile phones are surrounding us all around. Every call is encrypted although not so many people are aware of insufficiency of this protection. The A5/1 cipher was broken many times still the public feel false safety. That is the reason why I chose to deal with this theme.

My goal is to create program which shows user, how fast could the decrypt be. The solution consists of analysis of A5/1 cipher, design of decryption process and implementation and final discussion about the results.

Chapter 1

Description

1.1 History

Origin of the cipher is dating back to the year 1987 [9]. It was designed and intended only for Europe. Design was kept secret but there was leak of it in the year 1994 and in the 1999 the exact design of A5/1 cipher was published due to reverse engineering [12]. A5/1 belongs to seven ciphers intended to protect GSM (Groupe Spécial Mobile). In 1989 was presented the second version of A5 called A5/2 which was intended for american continent and which is much weaker version of A5/1 [15].

1.2 GSM

It is abbreviation of french words Groupe Spécial Mobile. Family of 3GPP networks achieved 6.44 billion devices which represents 90.6 % of the global market and the GSM is the most widespread mobile network [8]. It is used for comunication by mobile phones.

1.2.1 Structure

GSM belongs to cell systems. Network is divided to a hexagonal cells which are formed into clusters. Every cell in every cluster has one base station, which is responsible for selected channels and mobile devices in reach of that station. The sectorization method improves features. Every cluster is divided to 21 smaller cells and every small cell contains own base station . Every cell overlaps with all cells around. It ensures stable signal over the whole area [10]. Cells are divided into the three groups.

Big cells
: Diameter of 3 to 35 km. Antennas are typically the highest points of buildings. The watchtowers and hills above the cities can be used.

Small cells
Diameter smaller than 3 km. Antennas are not necessarily placed on the highest points of the surroundings.

Microcells
Even smaller diameter, usually bellow 300 m. Signal is spreaded by rebound of waves in the streets. Buildings and roofs are used for antenna placement.

Picocells
These cells are used in places with high concentration of devices such as skycrappers, office buildings, etc. Diameter of cell is up to tens of meters.

We distinguish three types of subsystems used for communication [6]:

Base Station Subsystem
This subsystem is designed for communication with mobile devices. The device communicates with the base station (BTS) using the Um interface and this station consequently communicates with control units (BSC) using Abis interface.

Network and Switching Subsystem
Provides communication between participants of GSM communication. The communication continues from BSC to mobile switching center (MSC) as well as to the area of registers with information about SIM cards and to the authentication center which was designed to ensure device authentication and secure transmission.

Operation and Support Subsystem
Services of OSS are traffic management, maintenance and financial aspects of communication. It is used by MSC.

1.3 Linear Feedback Shift Register

LFSR is type of shift register used in many branches of electronics industry. This is due to a simple implementation and useful properties. It works on a principle of finite state machine with a maximum of $2^n - 1$ states. LFSR found the application in design of stream ciphers as a part of keystream generators. But at first we will discuss some basic expressions.

1.3.1 Irreducible polynomial

Definition 1.3.1. [2, page 78] Let $f(x) \in F[x]$ be a polynomial of degree at least 1. Then $f(x)$ is said to be *irreducible* over F if it cannot be written as the product of two polynomials in $F[x]$, each of positive degree.

Irreducible polynomials are the base of unique factorization domain (UFD) and they are applicated in encryption algorithms. They are analogy to primes because they could not be further factorized. So the point is the factorization of large polynomials to irreducible polynomials. It is the same principle such as factorization of large numbers in RSA cipher.

1.3.2 Primitive polynomial

Definition 1.3.2. An irreducible polynomial $f(x) \in \mathbb{Z}_p[x]$ of degree m is called a *primitive polynomial* if x is a generator of $\mathbb{F}^*_{p^m}$, the multiplicative group of all the non-zero elements in $\mathbb{F}^*_{p^m} = \mathbb{Z}[x]/(f(x))$.[2, page 84]

If $q(x)$ is a primitive polynomial, it has to be able to generate all nonzero polynomials of the *mod* $q(x)$ group. This implicates feature which is imporant for LFSR. Every group containing generator is cyclic and primitive polynomial is that generator so it means it generates each nonzero polynomial and provides the longest cycle in the group [11]. LFSR uses this feature because usage of primitive polynomial as a function provides the longest cycle of pseudo-random numbers.

1.3.3 Description of LFSR

Definition 1.3.3. A *linear feedback shift register* (LFSR) of length L consists of L *stages* (or *delay elements*) numbered $0, 1, ..., L-1$, each capable of storing one bit and having one input and one output; and a clock which controls the movement of data. During each unit of time the following operations are performed:

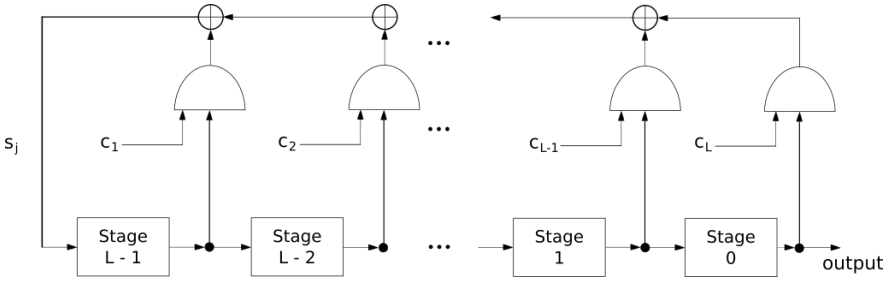
1. the content of stage 0 is output and forms part of the *output sequence*;
2. the content of stage i is moved to stage $i-1$ for each $i, 1 \leq i \leq L-1$; and
3. the new content of stage $L-1$ is the *feedback bit* s_j which is calculated by adding together modulo 2 the previous contents of a fixed subset of stages $0, 1, ..., L-1$.

[2, p. 195]:

Figure 1.1 describes the LFSR of length L. Every c_j is a bit value (0 or 1) and s_j is feedback bit. It is modulo 2 sum of the stored values of stages $i, 0 \leq i \leq L-1$, for which $c_{L-i} = 1$.

Definition 1.3.4. [2, page 197]: The LFSR of Figure 1.1 is denoted $< L, C(D) >$, where $C(D) = 1 + c_1 D + c_2 D^2 + \ldots + cLD^L \in \mathbb{Z}_2[D]$ is the *connection polynomial*. The LFSR is said to be *non-singular* if the degree of $C(D)$ is L (that is, $c_L = 1$). If the initial content of stage i is $s_i \in \{0, 1\}$ for

Figure 1.1: Design of LFSR of length L



each $i, 0 \leq i \leq L-1$, then $[s_{L-1}, \ldots, s_1, s_0]$ is called the *initial state* of the LFSR.

Singular LFSRs have not all output sequences periodic [2, p. 197], but the are *ultimately periodic*. If we ignore specific finite number of terms at the begining of sequence, we got a periodic subsequence of our sequence. Moreover if LFSR $< L, C(D) >$ is not singular and the connection polynomial $C(D)$ has degree L, the output sequence is always periodic.

The periodicity of LFSR output sequences depends on connection polynomial $C(D) \in \mathbb{Z}_2[D]$ of degree L. Such a polynomial can be irreducible or even primitive. Primitive polynomial $C(D)$ and corresponding $2^L - 1$ non-zero initial states produce an output sequence with maximum period $(2^L - 1)$. Polynomial $C(D)$ irreducible over $\mathbb{Z}_2$ and corresponding $2^L - 1$ non-zero initial states produce an output sequence with period equal to the least positive integer N such that $C(D)$ divides $1 + D^N$ in $\mathbb{Z}_2[D]$. Let define the last necessary term.

Definition 1.3.5. [2, p. 197]: If $C(D) \in \mathbb{Z}_2[D]$ is a primitive polynomial of degree L, then $< L, C(D) >$ is called a *maximum-length* LFSR. The output of a maximum-length LFSR with non-zero initial state is called an *m-sequence*.

Our designated m-sequence has statisticaly important property. The distribution of patterns with fixed length is almost exactly uniform. Let k be the length of pattern and $1 \leq k \leq L$ and $\hat{s}$ be our chosen pattern, subsequence of s of length $2^L + k - 2$. Each sequence has length k and non-zero sequences appears exactly 2^{L-k} times and zero sequence $2^{L-k} - 1$ times as a subsequence $\hat{s}$.

One metrics to measure sequences is called linear complexity. It is given by relation of sequences to LFSRs. So let denote s as an infinite sequence $s_1, s_2, \ldots.$ and s^n as an finite sequence $s_1, s_2, \ldots, s_{n-1}$. LFSR *generates* the infinite sequence s if there is an initial state which leads for identical output sequence as s. Similarly, a LFSR *generates* a finite sequence s^n if there is an initial state which leads for output sequence t with first n terms identical as s^n, so $t^n = s^n$.

Definition 1.3.6. [2, p. 198] The *linear complexity* of an infinite binary sequence s, denoted $L(s)$, is defined as follows:

1. If s is the zero sequence $s = 0, 0, 0, \ldots$, then $L(s) = 0$.
2. If no LFSR generates s, then $L(s) = \infty$.
3. Otherwise, $L(s)$ is the length of the shortest LFSR that generates s.

Let s, t be some binary sequences. Then the linear complexity of bitwise XOR operation between s and t is given by $L(s \oplus t) \leq L(s) + L(t)$. Let n be positive integer, then the linear complexity of the subsequence s^n is given by $0 \leq L(s^n) \leq n$. Linear complexity of sequence s^n, i.e. $L(s^n) = 0$ if and only if s^n is the zero sequence and $L(s^n) = n$ if and only if the sequence $s^n = 0, 0, \ldots, 0, 1$.

1.4 Design

1.4.1 Stream ciphers

Definition 1.4.1. Let A be the alphabet of q symbols, let $M = C$ be the set of finite strings over the A and let K be the set of keys. Stream cipher consists of transformation (generator) G, mapping E and mapping D. Generator G generates sequence of key $h(1), h(2), \ldots$ for every key $k \in K$ while $h(i)$ represents arbitrary substitutions $E_{h(1)}, E_{h(2)}, \ldots$ over the A alphabet. Mappings E and D assign transformations of encryption E_k and decryption D_k to every key $k \in L$. Encryption of plaintext $m = m(1), m(2), \ldots$ runs according to:

$$c(1) = E_{h(1)}(m(1)), c(2) = E_{h(1)}(m(1)) \tag{1.1}$$

and decryption according to:

$$m(1) = D_{h(1)}(c(1)), m(2) = D_{h(1)}(c(1)) \tag{1.2}$$

for $D_{h(i)} = E^{-1}_{h(i)}$ [16]

Stream ciphers was inspired by the unbreakable Vernam cipher (one-time pad). Vernam cipher is based on keystream consisting of completely random sequence of digits at minimal length of plaintext. Every character of plaintext is combined with one character of keystream and the keystream is never used again. This process implicates the impossibility to decrypt the ciphertext because we can not determine the true plaintext, the ciphertext does not contain any information about the key so every combination of characters is possible.

The ciphers used the idea of completely random keystream. We can create pseudo-random sequence using key of a specific length. The key is proceeded

by generator of keystream and creates the pseudo-random sequence of keystream which is then used to produce ciphertext. The LFSR is used often as a generator due to very simple architecture and low costs. The most common operation is exclusive or (XOR) so the operation is reversable and therefore symetrical.

1.4.1.1 Self-synchronizing Stream Ciphers

Definition 1.4.2. In a self-synchronizing, or asynchronous, stream cipher, the keystream depends on the secret key of the scheme, but also of a fixed number, say t, of ciphertext digits (that have already been produced, or read; this distinguishes it from a synchronous stream cipher). [16]

The dependance of keystream on previous produced digits allows us to produce ciphertext which is not resistant to insertion, loss or change of single digits because the error is reflected to t consecutive digits at most. This does not affect the cipher capability of self-resynchronization. But the detection of active attack, such as insertion, deletion or replay of ciphertext digits in more difficult to reveal as well. [14]

1.4.1.2 Synchronous Stream Ciphers

Definition 1.4.3. A synchronous stream cipher is a stream cipher, in which the keystream is generated independently of the plaintext and of the ciphertext. The keystream is usually produced by a pseudorandom generator, parameterized by a key, which is the secret key of the whole scheme. [16]

The keystream is using the initialization vector (IV) which is public known but new for every frame. The keystream is generated independently on plaintext and ciphertext but on the other hand the synchronization of both encryption and decryption keystream generators is necessary. Every error leads to incorrect decryption and keystream generators have to be resynchonized again at the position of error. Synchronous stream ciphers got better properties in error propagation. One false digit of keystream leads to one false digit of decrypted ciphertext because there is no feedback typical for self-synchronizing stream ciphers.

1.4.2 Architecture

A5/1 belongs to stream ciphers. This one works with blocks of information which are encrypted separately but the principle of encryption is based on stream cipher. In the case of A5/1 these blocks are 228 bits long. Every block corresponds to 4.6 miliseconds of communication in digital form. There are two parameters. One of them is 64 bits long secret key and the second is 22 bits long publicly known frame number.[1]

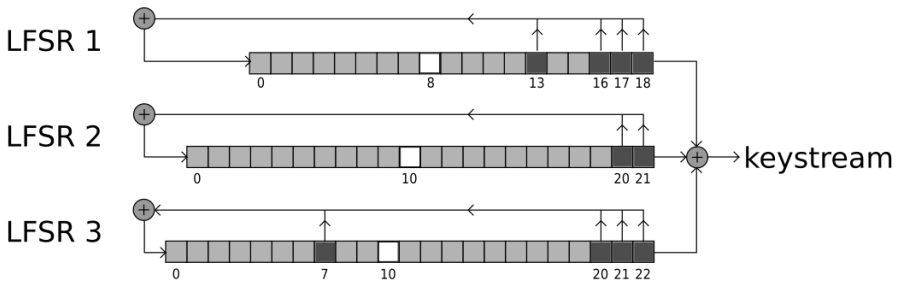
The cipher contains three LFSRs, denoted $R1$, $R2$ and $R3$. Registers are 19, 22 and 23 bits long, bits of every register are denoted from left side and begins by zero. Each of registers has feedback bits. These are 13, 16, 17, 18 for register $R1$, 20, 21 for register $R2$ and 7, 20, 21, 22 for register $R3$. Feedback bits are chosen according to the primitive polynomial which represents given register. The only nonlinear part of LFSRs are tap bits. Every register contains one tap bit. Register $R1$ has tap bit on position 8 and registers $R2$ and $R3$ have tap bits on position 10. LFSRs are shifted only in the case of the same value of tap bit as the dominant value. Dominant value is determined by majority rule. This rule is shown in table 1.1 . The last bits of every LFSR are XORed during every clock and outcome bit of this operation is the keystream bit. In the case of the same value of tap bit and dominant value, feedback bits for every single register are XORed and the resulting bit is sent to 0 position. This moves other bits one position. The keystream is used at the end to finish the whole procedure and XOR itself with the plaintext.

Table 1.1:

$R1$	$R2$	$R3$	Dominant value
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

At the begining, registers are empty, they contain zeros at all positions. The first part of initialization consists of 64 cycles. Feedback bits and bits of secret keys are XORed during these. During next 22 cycles, feedback bits and bits of frame number are XORed. These two parts of initialization ignores majority function. The last part consist of 100 cycles when nonlinear mojority function is used. The output is ignored and then the encryption of plaintext takes part. This encryption consists of 228 cycles. During these cycles the bit of keystream is created and it is XORed with bit of plaintext. The result is the ciphertext. [12]

Figure 1.2: Design of A5/1



1.4.3 Initialization in pseudocode

```
INPUT:  64 bit key
                22 bit frame number
                228 bit plaintext

OUTPUT: 228 bit ciphertext

        1.      Prepare registers.
                In for cycle for i from 1 to 19 load 0 to
                registers on position i, R1[i], R2[i], R3[i].
                In for cycle for i from 20 to 21 load 0 to
                registers on position i, R2[i], R3[i].
                Load 0 to R3[22].

        2.      Load key during 64 clocks ignoring irregular
                clocking.
                For i from 1 to 64 clock R1, R2, R3 and XOR
                    bit of key on position i with output bit
                    of each register, then send it as a
                    feedback.

        3.      Load frame number during 22 clocks ignoring
                irregular clocking.
                For i from 1 to 22 clock R1, R2, R3 and XOR
                    bit of frame number on position i with
                    output bit of each register, then send it
                    as a feedback.

        4.      Clock registers using irregular clocking.
                For i from 1 to 100 clock R1, R2, R3
                    according to majority function.
                Use output only as a feedback.
```

1.5 Features

We can deduce average number of shifts of every register during the initialization phase from the design of the nonlinear element of cipher. Every register is shifted with probability $\frac{3}{4}$. It means that after 101 cycles (using majority function) there are 76 register shifts in average.

1.5.1 Weaknesses

There are few main weaknesses in the design of A5/1 cipher, that cause inclusion of this cipher to weak ciphers. The greatest problem is missing ability to generate unpredictable keystream that is long enough. Over time, it was discovered more important weaknesses in the design. These are the most imporant:

- Generation of final keystream which is ment to be XORed with plaintext. Unfortunately, XOR is a linear function so we can recover the state before XORing and this is the reason why XOR operation does not ensure suitable security. It is easily breakable using linear cryptanalysis.
- Next problem is a short periodicity of the containt of LFSR register. If we exclude majority function we get

  $(2^{19} - 1)(2^{22} - 1)(2^{23} - 1)$

  of posibilities that can occure in registers. However, after scientific measurements, the number of posibilities we get in reality is only about $(4/3)(2^{23} - 1)$.
- The third weakness is the collision between the initial state of LFSR register (seed) and the resulting keystream. Not every seed ensures unique keystream. This results only 70 % of seeds, that guarantees an absolutely unique keystream.
- Last but not least, usage of majority function turned out to be one of the worst funcions all over the affinite functions.

Chapter 2

Analysis and design

2.1 Attack types

We distinguish two basic types of attacks, brute-force attacks and dictionary type of attacks.

2.1.1 Brute-force attack

Brute-force method means trying every possible combinations of elements of key. It is used to find user name and password or find password to encrypted archive for example, but it is useless for realtime attacks or one-time pad, the unbreakable Vernam cipher. This type of attack is simple, usable to almost every cipher but very demanding on hardware.

Due to the simplicity of brute-force attack there are some ways how to improve the performance. Typical computers in homes are equipped with dual-core or quad-core, in special cases hexa-core or octo-core processors and graphic card with one core, in special cases two graphic cards combined together using SLI or CrossFire technologies, depends on manufacturer of graphics processing unit (GPU). The rest of the equipment of computer is not mentioned because the impact on performance of brute-force attacks on ciphers is minimal.

Brute-force method is such a simple that it can be easily parallelized. Computation of all keys is accelerated almost linearly according to number of threads computer can offer. This means computer with 8 cores is almost exactly 4 times faster than computer with 2 cores. Due to trend of multi-core CPUs (central processing unit) and GPUs and even multi-CPU and multi-GPU there is huge increase in parallel performance and thus need to increase size of keys used in ciphers. Computation time of brute-force attack depends on the size of key used in cipher. It increases exponentially with every bit of the key, it expands the key space. So the computation time of 64 bit key and 128 bit key is not twice as great but 2^{64} times greater.

Brute-force attack is limited by physical limits according to Landauer's principle called Landauer's limit. The principle describes minimum theoretical limit of power consumption required to erase one bit of information:

$$kT\ln(2) \tag{2.1}$$

where k is Boltzmann constant $k = 1.3806488 \cdot 10^{-23}\mathrm{m}^2\mathrm{kgs}^{-2}\mathrm{K}^{-1}$ and T is temperature in Kelvin. [4] According to this law 128 bit key is safe for current applications. The device cracking the 128 bit key using brute-force method would consume at minimum 10 GW for 100 years only for computation of all values of the key. The operations of checking if the generated key is the solution would consume many times more energy. This limit is dependant on the actual architecture and multi-core, multi-CPU and multi-GPU trend affects the final consumption of energy and efficiency of the brute-force implementation.

The reverse type of attack is called reverse brute-force attack. It takes part when attacker know the key or multiple keys and tries to guess another credentials, for example username or corresponding encrypted archive. It is often used as a part of social engineering. The attacker identifies the victims and tries the most common passwords such as names (John, Black, JohnBlack, etc.) or numerical series easy to remember (12345, 147258369, 112233445566778899, etc.).

2.1.2 Dictionary attack

Problem of computation of key values is solved by precomputation and storing keys in so called dictionary. The basic type of dictionary attack is using list of most likely used words as a passphrases. Typical dictionary attack consists of simple loop through the whole dictionary, trying every record inside. This type is effective only for simple passwords and common users who do not consider the security of their property. But there are some advanced techniques of dictionary attacks. [13]

2.1.2.1 Time-memory tradeoff attack

More technique than attack. It describes problem of balance between time work and memory usage so it is balance between true dictionary attack and brute force attack. This type of attack can be done in many ways. Most simle is precomputation of hash tuples (password and adequate hash value) of the most used passwords. We know the hash of password and we know the hash mechanism. There is possibility of occurence of the password we are looking for in the table. There is possibility we will not find the right hash and therefore we will have to find password by exhaustive search over all possible keys. [13]

2.1.2.2 Rainbow tables

These are one of the alterations to the time-memory tradeoff attack. Design of rainbow tables solves the problem of huge amount of storaged data by usage of reduction function. The one-way known function $H(k)$ is used to hash password and we create new reduction function $R(h)$ which generates possible password from set of possible passwords. The principle of the rainbow tables is repeating of hashing and reduction of given password. Reduction funcion $R(h)$ does not recover the key, it is used to chain more hashes to one start point (password) and one end point (hash). The table consists of tuples of start points and end points. [13]

If we search the password we seek the hash in the set of end points. If the hash is not found we use the reduction function to calculate a new password. We seek for this new password in table again if we fail, repeat the whole iteration again. If the hash is found we look into table for the start point and we go through the whole chain. There is possibility of collisions between hashes and passwords so if we determine the found password false, we continue in seeking chain of reduction and hash functions. This techinque is ment to balance the usage of memory and computing time.

The simples method to make password hashes resistant to dictionary attacks is usage of cryptographic hash salt. It is one-way function to change the original password. It can be addition of some bits to the password of switching some bits in the password.

2.1.3 Divide and conquer attack

Refers to an algorithm to solve complex problems dividing them to simplier ones. In the case of cryptoanalysis, this attack is based on dividing a key (problem) to smaller units and gathering information we can get from these. The units have to be small that we are able to calculate keys. There are limitations for this type of attack. We need to be able to guess parts of the key separately and to confirm or reject the validity of these guesses with reasonable probability. Many attacks belongs to divide and conquer attacks, the most famous are correlation attacks on stream ciphers which will be discused in chapter (2). [17]

2.1.4 Side-Channel Attacks

Completely different principle of attacks compared to brute-force and dictionary methods mentioned before. Side-channel attacks aims to physical phenomena such as fluctuations in temperature, electromagnetic field, sound phenomena, power consumption or the fault induction attack.[3]

Timing attack It is based on the fact that every computation and every operation takes different amount of time. This attack is statistical so the

difference between single operations can be really small so it is important to measure carefully. It is possible to find fixed parameters of ciphers, for example Diffie-Hellman exponents or factor RSA keys. The time difference in operations is caused by performance optimalizations, RAM cache hit, branching, etc.

Power consumption The attacker monitors fluctuations in power consumption during the encryption operation. Transistors are basic component of electronic devices including encryption units. "The motion of electric charge consumes power and produces electromagnetic radiation, both of which are externally detectable."[3]. This information are used as an addition to information derived by other cryptoanalytic methods. This method is more important for chip cards due to their lack of own power supply.

2.2 Known attacks on A5/1

Ciphers are designed to be resistant to brute-force attacks. The resistance is ensured by usage of so long key that the computation of all possible combinations using current computers takes inadequate time, e.g. 10 000 years. Because key is 64 bits long, we get to 2^{64} combinations for every frame. If we take into account that every frame is 4.6 miliseconds of call, it will take about 3 - 6 years of searching for right set of keystreams and this is unacceptable for real-time decryption.

The second option is precomputation of all possibilities of frame content. That means preparation of all 2^{64} keys and for every key another 2^{228} possible contents of communication. If we want to realize this dictionary attack, we need to prepare tremendous ammount of empty space to storage the content of our dictionary. This one takes about 18446744 TB and also huge computing power to search in and manipulate with these data during the creation phase.

One of the feasible attacks was introduced by Anderson and Roe. It was based on guessing of 41 bits of the smaller registers $R1$ and $R2$ followed by calculationg the content of register $R3$. It was importent to take into account guessing of bits to find out value of majority function. In the end, it produced mechanism with complexity 2^{45}. It takes about month to find out the keystream.

In this days, we know few other types of attack on A5/1 cipher. One of them is "divide-and-conquer" attack. The assumption is knowledge of piece of plaintext and corresponding piece of ciphertext. It is based on deducing of initialization phase of cipher. Initialization phase depends on secret key which we can derive.[7]

Let

$$S(t) = (S_1(t), S_2(t), S_3(t)) \tag{2.2}$$

denotes the inner state of cipher in time $t \geq 0$ and $S(0)$ denotes initial state. At first, we have to get inner state $S(101)$, which denotes the first inner state after initialization. Then, we have to get initial state

$$S(0) = (S_1(0), S_2(0), S_3(0)). \tag{2.3}$$

That transition function is not one-to-one. There is no way to reach all of 2^{64} possible states so ammount of states is only subset of set of states $S(0)$. We can reach only $5 \cdot 2^{61} \approx 2^{63.32}$ in time $t = 1$. It results in possibility that different states generates the same inner states or even the same keystream. [7]

Golic proved that we need approximately 64 consecutive bits of plaintext and ciphertext.

This 64 bits represents vector boolean funcion of initial state of cipher. The idea is the determination of values of two shorter LFSR registers and due to knowledge of final bit we can calculate values of the third LFSR. The complexity is approximately 2^{40}. [7]

However computation of matrix of equations is in this case slower than design from Anderson and Roe even though it has lower complexity.

But Golic suggests improvement of his attack using Time-Memory TradeOff method.

Chapter 3

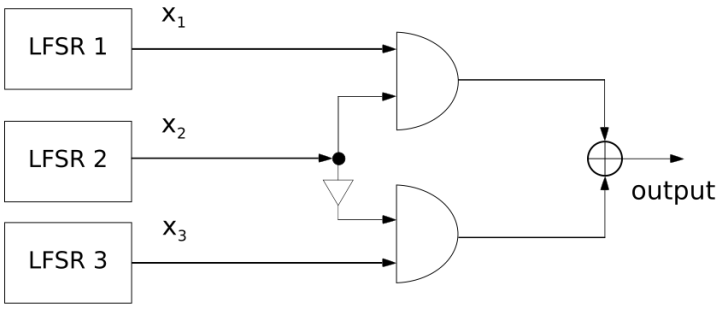
Correlation Attack

Our chosen attack belongs to known plain-text attack designed for stream ciphers. The main idea is to use statistical attributes of cipher, correlation in this case, for example correlation between value of LFSR and value of keystream bit.

3.1 Geffe generator

It is a synchronous stream cipher which consists of three LFSRs and non-linear combining function. This function combines LFSRs and generates output.

Figure 3.1: Design of Geffe generator



3.1.1 Attributes

Let the first LFSR is called $R1$, second LFSR is called $R2$ and third LFSR is called $R3$ respectively with lengths $L1, L2, L3$. The lengths are relatively prime. We denote x_1, x_2, x_3 as the output bits of $R1$, $R2$ and $R3$. Then the nonlinear combining function used in Geffe generator to provide output is given by

$$f(x_1, x_2, x_3) = x_1x_2 \oplus (1 + x_2)x_3 = x_1x_2 \oplus x_2x_3 \oplus x_3 \tag{3.1}$$

[2, p. 206].

The generated keystream has period $2^{L1-1} \cdot 2^{L2-1} \cdot 2^{L3-1}$ with linear complexity $L = L_1L_2 + L_2L_3 + L_3$.

Although this generator is weak and vunerable to correlation. There is correlation between output of register $R2$ and generator output. The probability of mutual value is 75 %. The same probability applies to correlation between output of $R3$ and generator output. We can express it as a *correlation probability.* "Let $x_1(t), x_2(t), x_3(t), z(t)$ denote the t^th output bit of LFSRs 1, 2, 3 and the keystream. Then the correlation probability of the sequence $x_1(t)$ to the output sequence $z(t)$ is"

$$\begin{aligned} P(z(t) = x_1(t)) &= P(x_2(t) = 1) + P(x_2(t) = 0) \cdot P(x_3(t) = x_1(t)) \\ &= \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} = \frac{3}{4} \end{aligned} \tag{3.2}$$

.

Table 3.1 clarifies these correlations.

Table 3.1:

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Value of x_2 is equal to value of $F(x_1, x_2, x_3)$ in six of eight possible situations, so probability of mutual value is 75 %. The same situation occurs in case of the x_3 value. Knowledge of taps is essential, so we can not use this type of attack. But if we know these, we can deduce the initial value of $R2$ and $R3$ and generate the output sequences of these registers. Then we are able to

count every matching bits from guessed registers and output of the generator. Let o be output bit of generator. If percentage of equalities between o and $F(x_1, x_2, x_3)$ is around 50%, our guess was wrong. If we guessed right, the percentage of equalities is around 75%.

3.2 Basic correlation attack

First of all, let us remind necessary information about A5/1 and denote basic variables. Every conversation is based on frame F and encryption using session key K. Frame contains of 228 bits of information. Subjects A sends 114 bits to B and B sends 114 bits to A. Session key is unique for every conversation. Session key K is loaded to registers and mixed with *frame counter* F_n for every recieved frame. Then registers are clocked 100 times to mix the values and make possible attacks more difficult. The result is the initial state of the shift registers. Then 228 bits of running key are produced and xored with 228 bits plaintext, producing 228 bits of keystream.

Cipher consists of three LFSRs denoted by $R1$, $R2$ and $R3$. Each of them has got primitive feedback polynomial. Production of running key is provided by XOR of the most significant bit from each LFSR. Clocking of LFSRs is irregular due to non-linear stop/go function with majority rule. Clocking taps are indicated by C_1, C_2 and C_3.

Let us denote the running key by $z = z_1, z_2, ...$ This is known-plaintext attack so we have access to N bits of ciphertext and corresponding plaintext as well thus we are able to deduce the N first bits of z. This attack is based on the assumption that we got m different frames, each initialized with session key K but with different F_n. The initial state of A5/1 is linear function of K and F_n. Let

$$K = (k_1, k_2, ..., k_n) \tag{3.3}$$

and

$$F_n = (f_1, f_2, ..., f_{22}), \tag{3.4}$$

where k_i, $f_i \in \mathbb{F}$. Let u_0^1 be the first bit of LFSR output produced by *regular* clocking and $u_0^1, u_0^2, ...$ be the whole LFSR output sequence. According to previous, let u_0^2 be the first bit of $R2$ output and $u_0^2, u_1^2, ...$ be the sequence and finally let u_0^3 be the first bit of $R3$ output and $u_0^3, u_1^3, ...$ be the sequence. It follows that $(u_0^1, u_0^2, ..., u_0^{18})$ represents the initial state of $R1$, $(u_1^1, u_1^2, ..., u_1^{21})$ represents the initial state of $R2$ and $(u_3^1, u_3^2, ..., u_3^{22})$ represents the initial state of $R3$.[12]

Using linearity of the initial state of the cipher, composed of key K and frame number F_n we are able to claim that every LFSR initial state can be

described as a linear function of K and F_n. So then every LFSR output symbol from $R1$ (and similarly for $R2$ and $R3$) can be described as

$$u_t^1 = \sum_{i=1}^{64} c_{it}^1 k_i + \sum_{i=1}^{22} d_{it}^1 f_i, \tag{3.5}$$

$$u_t^2 = \sum_{i=1}^{64} c_{it}^2 k_i + \sum_{i=1}^{22} d_{it}^2 f_i, \tag{3.6}$$

$$u_t^3 = \sum_{i=1}^{64} c_{it}^3 k_i + \sum_{i=1}^{22} d_{it}^3 f_i, \tag{3.7}$$

$c_{it}^1 k_i, c_{it}^2 k_i, c_{it}^3 k_i, i = 0, \ldots, 64, t \geq 0$ and $d_{it}^1 f_i, d_{it}^2 f_i, d_{it}^3 f_i, i = 0, \ldots, 22, t \geq 0$ are known binary constants. They are known because they are given by known regular clocking. Let

$$s_t^1 = \sum_{i=1}^{64} c_{it}^1 k_i \tag{3.8}$$

be the secret or the key part of u_t^1 and let

$$\hat{f}_t^1 = \sum_{i=1}^{22} d_{it}^1 f_i, t \geq 0 \tag{3.9}$$

be the frame part of output symbol u_t^1 so we can claim

$$u_t^1 = s_t^1 + \hat{f}_t^1, t \geq 0. \tag{3.10}$$

s_t^1 could be also written

$$s_t^1 = \sum_{i=0}^{18} \hat{c}_{it}^1 s_i^1 \tag{3.11}$$

for known binary constants $\hat{c}_{it}^1, i = 0, \ldots, 18, t \geq 0$. [12]

We know that sequence $s_0^1, s_1^1, s_2^1, \ldots$ is a constant sequence for the whole conversation. It means every single frame has this sequence the same although it is unknown. This feature is caused by dependence only on K, which is constant during the whole conversation.

We also know the $\hat{f}_1^1, \hat{f}_2^1, \hat{f}_3^1, \ldots$ sequence although frame counter is different for every frame but always known. This means that sequence $\hat{f}_1^1, \hat{f}_2^1, \hat{f}_3^1, \ldots$ can be calculated for each frame. According to the notation we have to introduce s_t^2, s_t^3 as a secret part and $\hat{f}_t^2, \hat{f}_t^3$ as a frame part of u_t^2, u_t^3 of registers $R2, R3$

$$u_t^2 = s_t^2 + \hat{f}_t^2, t \geq 0, \tag{3.12}$$

$$u_t^3 = s_t^3 + \hat{f}_t^3, t \geq 0. \tag{3.13}$$

The correlation attack is based on statistical behavior of A5/1. At the beginning, registers are 100 times irregularly clocked with no output and then once more for the first bit of keystream. This means registers are 101 times irregularly clocked with 75% chance to been clocked. It follows that every LFSR has been clocked about 76 times. We can use this information and assume that every LFSR has been clocked exactly 76 times. We denote $z_1, z_2, \ldots, z_{228}$ as the running key we are looking for. Then z_1 is the product of XOR for each LFSR,

$$z_1 = u_{76}^1 + u_{76}^2 + u_{76}^3. \tag{3.14}$$

This equation can be decomposed using previous equations (3.9), (3.11) and (3.12)

$$z_1 = s_{76}^1 + \hat{f}_{76}^1 + s_{76}^2 + \hat{f}_{76}^2 + s_{76}^3 + \hat{f}_{76}^3. \tag{3.15}$$

We know $\hat{f}_1, \hat{f}_2, \ldots$ because frame number is generally known and $\hat{f}_1, \hat{f}_2, \ldots$ can be calculated. Now we modify equation (3.14) to preferablet form.

$$s_{76}^1 + s_{76}^2 + s_{76}^3 = \hat{f}_{76}^1 + \hat{f}_{76}^2 + \hat{f}_{76}^3 + z_1. \tag{3.16}$$

Left side contains only unknown variables and right side consists of known output value z_1 and known values $\hat{f}_{76}^1$, $\hat{f}_{76}^2$ and $\hat{f}_{76}^3$. The right side of equation will be denoted as $O_{76,76,76,1}^j$ for the frame j. Now we have one bit of information about the key in frame j under the assumption that every LFSR was clocked exactly 76 times was correct. There is possibility that our assumption was not correct but because it is independent event we still have probability 50 % that our bit of information is correct. So in this point, correlation is described by equation[12]

$$\begin{aligned} P(s_{76}^1 + s_{76}^2 + s_{76}^3 = O_{(76,76,76,1)}^j) & \\ = & P(assumption\ correct) \cdot 1 \\ + & P(assumption\ wrong) \cdot \frac{1}{2}. \end{aligned} \tag{3.17}$$

P. Ekdahl and T. Johansson calculated the probability of all three LFSRs being clocked exactly 76 times about 10^{-3}. So equation

$$P(s_{76}^1 + s_{76}^2 + s_{76}^3 = O_{(76,76,76,1)}^j) = \frac{1}{2} + \frac{1}{2} \cdot 10^{-3} \tag{3.18}$$

is the probability the assume was correct.

$s_{76}^1 + s_{76}^2 + s_{76}^3$ remains constant for every frame so in the case of access to a few million frames we can calculate $O_{76,76,76,1}^j$ for every one of them and

identify $s^1_{76} + s^2_{76} + s^3_{76}$ with high confidence. Usage of this idea for other triplets allows us to derive more information about running key. With at least 64 bits of information about the key we are able to recover the running key.

3.3 A Refinement of the Attack

Attack we mentioned before is practical but not effective. It requires many captured frames which could be difficult to obtain. This condition degrades this attack so Ekdahl and Johannson refined it.[12]

We use the notation as before (K for secret key, F_n for frame counter, etc.) and add another. Let $w_0, w_1, \ldots$ be the produced keystream including the first 100 discarded values. So running key $z_1 = w_{101}$, $z_2 = w_{102}$, $\ldots$, $z_{228} = w_{328}$. Let us have a certain clocking triple (cl_1, cl_2, cl_3) which describes clocking of three registers $R1$,$R2$,$R3$. It is not necessary to assign this triple to only one position. It might occur in more positions. Triple $(79, 79, 79)$ might occur at position 101, 102, etc. but not earlier. Keystream positions before 101 are not aproachable because first 100 positions are discarded. We are not certain that triple will occur at this position. So let

$$P((cl_1, cl_2, cl_3) \text{ in } v\text{th position}) \tag{3.19}$$

be the probability of occurence of triple (cl_1, cl_2, cl_3) at the position v. Let us note, position v symbolize keystream symbol w_v. The probability of occurrence is not the same throughout the entire interval. We are able to calculate interval of non-negligible possibility of occurrence of our chosen triple. This means we can use all positions with probability of occurrence of triple. Denote this interval $\mathcal{I}$ for v. Let denote $p^j_{(cl_1,cl_2,cl_3)}$ the probability of occurrence of triple (cl_1, cl_2, cl_3) in frame j over the interval $\mathcal{I}$. So

$$p^j_{(cl_1,cl_2,cl_3)} = P(s^1_{cl_1} + s^2_{cl_2} + s^3_{cl_3} = 0) \tag{3.20}$$

is equation we want to use as a weighted voting over the interval $\mathcal{I}$. We reach it using equation:

$$\begin{aligned} p^j_{(cl_1,cl_2,cl_3)} = \sum_{v\in\mathcal{I}} & P((cl_1, cl_2, cl_3) \text{ in } v\text{th position}) \\ & \cdot[O^j_{(cl_1,cl_2,cl_3),v-100} = 0] \\ + \frac{1}{2} \cdot (1 - \sum_{v\in\mathcal{I}} & P((cl_1, cl_2, cl_3) \text{ in } v\text{th position})) \end{aligned} \tag{3.21}$$

where $[x = 0]$ is an indicator funcion which equals 1 if $x = 0$ and 0 otherwise. $O^j_{(cl_1,cl_2,cl_3),v-100}$ was mentioned as notation of $\hat{f}^2_{cl_1} + \hat{f}^2_{cl_2} + \hat{f}^3_{cl_3} + z_{v-100}$ and $\mathcal{I}$ is an interval we chose to iterate over due to non-negligible posibility of occurrence of clocking triple (cl_1, cl_2, cl_3).

If we assume that the bits entering the clock controlling device of A5/1 are uniformly distributed independent bits, we can write the probability (13) as a recursive formula,

$$P((cl_1, cl_2, cl_3) \text{ in } v\text{th position}) = F((cl_1, cl_2, cl_3), v) \tag{3.22}$$

where

$$\begin{aligned}
F((cl_1, cl_2, cl_3), 0) &= 1 \text{ if } cl_1 = 0,\ cl_2 = 0\ , cl_3 = 0,\\
F((cl_1, cl_2, cl_3), v) &= 0 \text{ if } cl_1 < 0 \text{ or } cl_2 < 0 \text{ or } cl_3 < 0,\\
F((cl_1, cl_2, cl_3), v) &= 0 \text{ if } cl_1 > v \text{ or } cl_2 > v \text{ or } cl_3 > v,\\
F((cl_1, cl_2, cl_3), v) &= 0.25F(cl_1 - 1, cl_2 - 1, cl_3 - 1, v - 1)\\
&\quad +0.25F(cl_1, cl_2 - 1, cl_3 - 1, v - 1)\\
&\quad +0.25F(cl_1 - 1, cl_2, cl_3 - 1, v - 1)\\
&\quad +0.25F(cl_1 - 1, cl_2 - 1, cl_3, v - 1)
\end{aligned} \tag{3.23}$$

This is applicable under assumption of independent uniform distribution of clocking bits. The first part describes situation when clocking triple consists of $(0, 0, 0)$ and position $(v = 0)$. This is the first position before initialization so none of the registers is clocked yet and it is certain that triple is $(0, 0, 0)$ right. Next piece of recursive equation solves the situation of smaller number of clocks than number of clocked of registers. So if cl_1, cl_2 or cl_3 is smaller number than 0 then there is no chance of occurence of triple, value is 0 and recursion is over. The third situation is analogical. Clocking of any single register can not be higher than main clock of the cipher. If cl_1, cl_2 or cl_3 is bigger number than v then there is no chance of occurence of triple, value is 0 and recursion is over. The last part takes care of recursion. We separated weight equally between four equations with different triples. There are all possible combinations of triples included and the end of recursion is ensured by decreasing of clocking triples in every step of recursion.

But this recursive formula is not the only one usable in our situation. These are used to approximate the case of A5/1 but the approximation works well when the probability is high. A simplier, non-recursive formula brings the same result. The assumptions are the same as before so expression applies in the case of cl_1, cl_2 and cl_3 are bigger than 0 and smaller than v. The expression is following:

$$P((cl_1, cl_2, cl_3) \text{ in } v\text{th position}) = \frac{\binom{v}{v-cl_1}\binom{v-(v-cl_1)}{v-cl_2}\binom{v-(v-cl_1)-(v-cl_2)}{v-cl_3}}{4^v} \tag{3.24}$$

The next step is an introduction of a log-likelihood ratio. Remind us that $p^j_{(cl_1,cl_2,cl_3)}$ describes probability that $s^1_{cl_1} + s^2_{cl_2} + s^3_{cl_3} = 0$ in the frame j and we introduce

$$\hat{p}_{(cl_1,cl_2,cl_3)} = P(s^1_{cl_1} + s^2_{cl_2} + s^3_{cl_3} = 0) \tag{3.25}$$

as the same expression used over all frames. Now we define the log-likelihood ratio as

$$\Lambda_{(cl_1,cl_2,cl_3)} = \ln \frac{\hat{p}_{(cl_1,cl_2,cl_3)}}{1 - \hat{p}_{(cl_1,cl_2,cl_3)}} \tag{3.26}$$

as estimation for one frame. We can calculate estimation of $\Lambda_{(cl_1,cl_2,cl_3)}$ over all frames using interval of all available frames (m).

$$\Lambda_{(cl_1,cl_2,cl_3)} = \sum_{j=1}^{m} \ln \frac{\hat{p}_{(cl_1,cl_2,cl_3)}}{1 - \hat{p}_{(cl_1,cl_2,cl_3)}} \tag{3.27}$$

We can summarize knowledge about log-likelihood ratio Λ in following table (3.2). It describes the link between Λ and the probability of $s^1_{cl_1} + s^2_{cl_2} + s^3_{cl_3}$ being zero. If Λ is less than zero the probability of being zero is lower than 50% so the value is most likely one. On the other hand if Λ is greater than zero the probability of being zero is greater than 50% so the value is really most likely zero.

Table 3.2:

Λ	$P(s^1_{cl_1} + s^2_{cl_2} + s^3_{cl_3} = 0)$
0	0
> 1	$> \frac{1}{2}$
< 1	$< \frac{1}{2}$

The final stage of our attack is choosing the parameters. We choose position $(79, 79, 79)$ and interval of size 8, denoted $\mathcal{C}_1 = \{79, \ldots, 86\}$. Then we want to calculate $\Lambda_{(cl_1,cl_2,cl_3)}$ so we start with all linear combinations of $s^1_{cl_1} + s^2_{cl_2} + s^3_{cl_3}$ for $(cl_1, cl_2, cl_3) \in \mathcal{C}_1$. These combinations are calculated for every frame $j = 1, \ldots, m$ to calculate $\hat{p}_{(cl_1,cl_2,cl_3)}$ using (3.26). This provides us information about every possible triple. As the last step we decide the value of $s^1_{cl_1} + s^2_{cl_2} + s^3_{cl_3}$ using $\Lambda_{(cl_1,cl_2,cl_3)}$ and the table (3.2).

In result we get system of equations with unknown variables. The number of equations and unknown variables is based on a size of the interval we choose. In our case we chose interval of size 8 and because triple (cl_1, cl_2, cl_3) goes through all values from the interval we get $8^3 = 512$ equations and $8 + 8 + 8$ unknown variables. This system is described as decoding of a length 512 linear code of dimension 24 so the 512 bits of word are calculated values of $s^1_{cl_1} + s^2_{cl_2} + s^3_{cl_3}$ and the corresponding equations are used as a parity check equations. This system of equations is soluble only by exhaustive search, i.e. brute-force method. We can not claim that estimated solution of equations is the right one, so we save $\mathcal{T} = 1000$ best solutions.

At the end, we are able to get 24 bits of information about the key for every interval. If the interval has an empty intersection with interval $\mathcal{C}_1$ we

get another 24 bits of the key so we need to choose at least 3 intervals. It is usefull to choose overlapping intervals, overlapp of our intervals can be used as a checking part. This checking part is based on filling the register $R1$ with estimated values $s_1, s_2, \ldots, s_{19}$. Than if we denote $f(R, i)$ as a clocking function of register R, which clocks the register i-times and returns newly created bit.

```
1)      Initialization.
        Load the most significant bits to registers.
2)      Comparison.
        Create for cycle for registers R1, R2 and R3 from
        1 to (24 - size of chosen register).
                Call f(R,i) for R1, R2, R3 and
                corresponding i.

                If f(R,i) returns the same value as
                s_{size of register + i}, check matches,
                chosen solution is wrong otherwise.
```

Checking of solution is made by running the cipher reverse. At first, the computed values are loaded into the registers and then running the cipher 79 + 22 (for frame counter) clocks backwards. Then the frame counter is loaded as usual and cipher is clocked 100 times. At the end, check the generated output against the recieved ciphertext.

Chapter 4

Realisation

4.1 Preparation

At first we prepare our test data. "The implemented attack needs the 40 first bits from about 2^{16} (possible non-consecutive) frames, which corresponds to about 5min GSM conversation." [6, conclusions]. So we prepare program written in C++ to generate one secret key, 2^{16} frame numbers and corresponding keystreams using the secret key. C++ language is chosen because rand() function for generation of pseudo-random numbers in C generates non-uniformly distributed numbers. [5]

The generated data are stored into file. First 8 bytes (64 bits) of file contains key value and then there are pairs of frame numbers and plaintext. These pairs consists of 3 bytes for frame number and 30 bytes for plaintext. These 30 bytes were chosen to simplify the implementation.

Marc Briceno, Ian Goldberg, and David Wagner published "A pedagogical implementation of A5/1". So we can use their program as a template and use their clocking and setup functions. The while cycle with inner logic for reading from the generated file we mentioned before. After reading one whole frame (frame number and plaintext) we call functions keysetup() and run() to create encrypted data. These data are written to new file. The moment of generating data has no impact on the performance, it took only 3 - 4 seconds, generation and encryption.

One of the optimalizations was precomputation of
$P((cl_1, cl_2, cl_3)$ in vth position). There is no primitive data type which can store number 2^{202}. Another problem is computation of $P((cl_1, cl_2, cl_3)$ in vth position) during the attack. The last thing is precision of the operations during computation according to (3.24). This problems were solved by SageMath.

SageMath is free open-source mathematic software. It builts on existing open-source packages, including Python. Small script was written to compute $P((cl_1, cl_2, cl_3)$ in vth position). It is not fast but speed was not purpose. This script is able to do computations with almost unlimited precision. We use this

script to precomputate $P((cl_1, cl_2, cl_3)$ in vth position) for all combinations of cl_1, cl_2, cl_3 and v.

4.2 Main Program

The implementation was written in C++ without using objective design. It consists of functions from "A pedagogical implementation of A5/1", my function for loading of encrypted data and the main function. At the beginning, we allocate array for frame numbers and corresponding ciphertext. Then other temporary structures such as arrays for log-likelihood ratio for every interval or arrays of $P((cl_1, cl_2, cl_3)$ in vth position) for every interval.

At first, we load log-likelihood ratios into arrays. This accelerates our program because we do not have to access files and access hard drive. Then we use the function loadEncrypted() to load file of encrypted data. We generated 2^16 frames so the size of file is only 2.2 MB. Next step is cycle for computation of $p^j_{(cl_1,cl_2,cl_3)}$. This cycle is crucial because it runs over all frames, over every combination of (cl_1, cl_2, cl_3) in interval $\mathcal{I}$ and over every postion $v - 100$. In the end, there five cycles of final $2^{16} \cdot 8^3 \cdot 40$ iterations.

There is another optimalization in this cycles. We calculate $p^j_{(cl_1,cl_2,cl_3)}$ for log-likelihood ratio. This means we need to calculate $O^j_{(76,76,76,1)}$ so also $\hat{f}^1_t, \hat{f}^2_t, \hat{f}^3_t$. The computation of $\hat{f}^1_t, \hat{f}^2_t, \hat{f}^3_t$ has the highest impact on the performace. But, values are dependant only on frame numbers. Thus we precomputed values of $\hat{f}^1_t, \hat{f}^2_t, \hat{f}^3_t$ and save $\hat{f}^1_t \oplus \hat{f}^2_t \oplus \hat{f}^3_t$ to file.

We have saved $p^j_{(cl_1,cl_2,cl_3)}$ to array of structures. This structures consists of three-dimensional array for every combination of $p^j_{(cl_1,cl_2,cl_3)}$ in every frame and every clocking. We use it in another cycle. We calculatel og-likelihood ratio and decide by Hard Decision (table 3.2) what bit will we save to array.

Anoher crucial part is solving of equations. Exhaustive search of solution for set of 512 equations with 24 variables means at least one cycle of 2^{24} iterations. But the operations inside are simple. Estimated values of s_1, s_2, s_3 are compared to calculated log-likelihood of given clocking. We can calculate minimal Hamming distance between given word and value of chosen $s_1+s_2+s_3$. $\mathcal{T}$ possible solutions are stored to the array.

The last part consists of iteration through the whole array of possible solutions and checking consistency. We use bits of possible solution to load registers, clock them and compare resulting bits from clocking with overlapping bits from solution. If this bits does not match, the solution is wrong and we continue.

4.3 Technologies

Only basic technologies were used during the implementation of attack.

g++ Compilator of C and C++ code. The "-g -Ofast" flags were used during compilation. Version 4.9.2 20150304

Sublime Text 3 Text editor writen in Python. Build 3083

SageMath Mathematical package with Python packages. It was used for difficult precomputations. Version 6.6

Inkscape Editor of vector graphics. Used for making of illustration figures. Version 0.91 r13725

Opera Browser used to search for information about the attack. Version 29.0.1795.47

lshw *nix script used to learn about laptop configuration. Version B.02.17

The simulations run on laptop with configuration

CPU Intel(R) Core(TM) i5-2410M CPU @ 2.30GHz

RAM 3867 MB

OS Arch Linux 64bit, 4.0.1-1-ARCH

4.4 Measurements

The most difficult part to computate is finding the most probable solutions according to overlapping bits. The program run with different numbers of saved solutions and the difference between performance was watched. Table (4.1) shows n as a number of saved solutions and $\hat{n}$ as a number of solutions chosen according to overlapping bits.

Table 4.1:

n	$\hat{n}$
100	67
150	88
200	111
250	150

Figure 4.1: Graph of computation time needed to solve bit overlapping for 100 saved solutions.

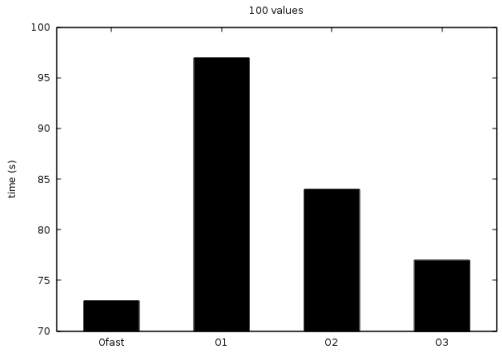


Figure 4.2: Graph of computation time needed to solve bit overlapping for 200 saved solutions.

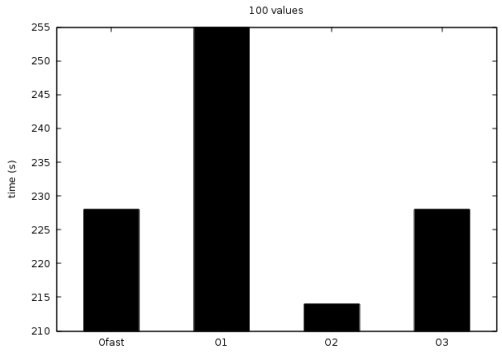
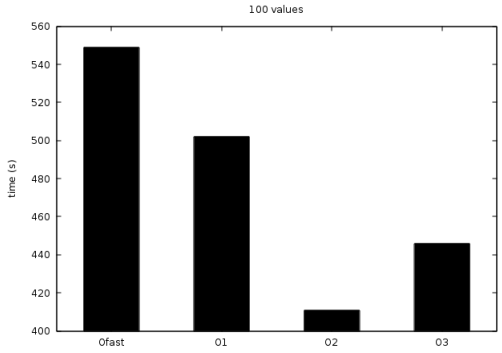


Figure 4.3: Graph of computation time needed to solve bit overlapping for 250 saved solutions.



As we can see at figures above (4.1, 4.2, 4.3), the x axis denotes optimalization rate and y axis denotes time. It is crucial to pick the right number of saved solution and choose the suitable optimalization rate. The time needed for computations is almost twice bigger for every 50 more solutions.

Conclusion

The correlation attack on A5/1 is ideal way to demonstrate idea of correlation attacks in deeper way. Our simulation of attack was not successfull. We was forced to reduce number $\mathcal{T}$ of possible results due to enormous time requirements. Optimalizations were made and brought significant improvement in performance but our chosen design turned out inefficient. This is the drawback of this method. The implementation is very design and language dependant.

Bibliography

[1] Steve Babbage Alexander Maximov Thomas Johansson. "An Improved Correlation Attack on A5/1". In: *Graduate School in Personal Computing and Communication PCC++* (2005).

[2] Scott A. Vanstone Alfred Menezes Paul C. van Oorschot. *Handbook of applied cryptography. Discrete mathematics and its applications*. 1st ed. Boca Raton: CRC Press, 1997. ISBN: 0849385237.

[3] Hagai Bar-El. *Introduction to Side Channel Attacks [online]*. Discretix Technologies Ltd. 2014. URL: `www.discretix.com` (visited on 28/04/2015). Israel.

[4] Bart Preneel Christof Paar Jan Pelzl. *Understanding Cryptography. A Textbook for Students and Practitioners*. Springer, 2010. ISBN: 3-642-04100-0.

[5] cplusplus.com. *rand [online]*. cplusplus.com. 2015. URL: `http://www.cplusplus.com/reference/cstdlib/rand/` (visited on 28/04/2015).

[6] Ondřej Dudek. *Struktura sítě GSM [online]*. Katedra radioelektroniky K 13137, ČVUT FEL, Praha. 23rd Mar. 2005. URL: `http://radio.feld.cvut.cz/personal/mikulak/MK/MK05_semestralky/Struktura_GSM_Ondrej_Dudek.pdf` (visited on 28/04/2015). Česká Republika, Praha.

[7] Jovan Dj. Golic. "Cryptanalysis of Alleged A5 Stream Cipher". In: *Advances in Cryptology - EUROCRYPT '97* (1997).

[8] GSA. *Fast Facts [online]*. The Global mobile Suppliers Association. 23rd Apr. 2015. URL: `http://www.gsacom.com/news/gsa_fastfacts.php4` (visited on 28/04/2015). United Kingdom (Sawbridgeworth).

[9] Martin Jureček. *Pokročilá kryptologie. A5/1*. Praha, 2013.

[10] Michael Miller. *How Mobile Networks Work [online]*. Pearson Education, Que Publishing. 14th Mar. 2013. URL: `http://www.quepublishing.com/articles/article.aspx?p=2021961` (visited on 28/04/2015). 800 East 96th Street, Indianapolis, Indiana 46240.

[11] Eliška Ochodková. *Matematické základy kryptografických algoritmů [online]*. Vysoká škola báňská, Západočeská univerzita. 2011. URL: `www.cs.vsb.cz/ochodkova/courses/kpb/mzka.pdf` (visited on 28/04/2015). Česká Republika, Ostrava, Plzeň.

[12] Thomas Johansson Patrik Ekdahl. "Another Attack on A5/1". In: *Submission to IEEE Transactions on Information Theory* (2002).

[13] I. Piller. "Hashovací funkce a jejich využití při autentizaci". master. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2009. 68 pp.

[14] Bruce Schneier. *Applied Cryptography. Second Edition: Protocols, Algorthms, and Source Code in C*. 2nd ed. Wiley Computer Publishing, John Wiley & Sons, Inc., 1996. ISBN: 0471128457.

[15] Dmitry Sklyarov. *Hidden Keys to Software Break-ins and Unauthorized Entry*. 1st ed. Wayne: A-List Publishing, 2004. ISBN: 1931769303.

[16] Dr. Vlastimil Klíma. *Základy moderní kryptologie - Symetrická kryptografie II [online]*. 11th Nov. 2014. URL: `http://cryptography.hyperlink.cz` (visited on 28/04/2015). Česká Republika.

[17] Jean-Jaques Quisquater Werner Schindler Francois Koeune. *Improving Divide and Conquer Attacks Against Cryptosystems by Better Error Detection / Correction Strategies*. article. Université catholique de Louvain, 2001.

Appendix A

Acronyms

UFD Unique Factorization Domain

LFSR Linear Feedback Shift Register

XOR Exclusive bitwise OR

CPU Central Processing Unit

GPU Graphic Processing Unit

RAM Random-Access Memory

***nix** Set of all systems based on Unix

Appendix B

Contents of enclosed CD

- `readme.txt` ..... documentation file
- `BPHolecMartin2014.tex` ..... LaTeX source file
- `BPHolecMartin.pdf` ..... thesis in PDF
- `encryption` ..... directory for encryption files
  - `encryption02` ..... encryption binary file
  - `encryption02.c` ..... encryption source file
  - `generator` ..... generator binary file
  - `generator.cpp` ..... generator source file
  - `randmaterial.txt` ..... precomputated random material
  - `randmaterialencrypted.txt` ..... encrypted random material
- `decryption` ..... directory for decryption files
  - `decryption11` ..... decryption binary file
  - `decryption11.c` ..... decryption source file
  - `binary.csv` ..... binary precomputated file
  - `probability.sage` ..... Sage script
  - `pcl1cl2cl3invthposition` ..... precomputated file of probabilities
    - `79th.txt` ..... precomputated file of probabilities
    - `84th.txt` ..... precomputated file of probabilities
    - `89th.txt` ..... precomputated file of probabilities
    - `94th.txt` ..... precomputated file of probabilities
    - `fhat.txt` ..... precomputated file of bit values