Comenius University
Faculty of Mathematics, Physics and Informatics
Department of Computer Science

Algebraic attack on stream ciphers

Master's Thesis

Martin Vörös

Bratislava, 2007

Comenius University
Faculty of Mathematics, Physics and Informatics
Department of Computer Science

Algebraic attack on stream ciphers

Master's Thesis

Martin Vörös

Adviser:
RNDr. Martin Stanek, PhD.

Bratislava, 2007

I hereby declare that the work presented in this thesis is my own, except where otherwise indicated, under the careful supervision of my thesis adviser.

.....

Martin Vörös

Acknowledgment

I would like to thank my thesis adviser Martin Stanek for careful guidance, very useful advices and helpful discussions during the work on this thesis.

Abstract

An algebraic attack is a new cryptanalytic method. The main idea behind this method is finding and solving a system of multivariate polynomial equations over finite field. Recently, algebraic attacks were very successful against certain stream ciphers based on LFSRs (e.g. nonlinear filter generators and nonlinear combination generators). In this paper, we explain the main ideas of algebraic attack on stream ciphers and illustrate them with examples. We describe methods for generating systems of polynomial equations for some classes of the stream ciphers based on LFSRs. We set out to study the XL algorithm for solving these systems and run various computer simulations for verifying the behaviour of the XL algorithm. Afterwards, the method for solving the system of nonlinear equations using SAT solvers is presented, and the experimental analysis of our heuristic for speeding up the conversion from algebraic normal form to conjunctive normal form is described. Efficiency of this conversion is eminent for the use of SAT solver for solving the system of equations. We concluded that the presented heuristic is highly efficient for the systems with high density. We also explain the improvement of the algebraic attack – namely the fast algebraic attack.

Keywords: Algebraic attack, fast algebraic attack, stream ciphers, linear feedback shift register, A5/1, XL algorithm, SAT solver.

Notation

$\mathcal{A}$	System of equations
ANF	Algebraic Normal Form
CNF	Conjunctive Normal Form
$\deg(f)$	Degree of the Boolean function (polynomial) f
$\mathbb{F}_q$	Finite field with q elements
$\mathbb{F}[x_1, x_2, \ldots, x_n]$	Ring of multivariate polynomials over field $\mathbb{F}$
FAA	Fast Algebraic Attack
$\mathcal{K}$	Encryption key
LFSR	Linear Feedback Shift Register
NLCG	Nonlinear Combination Generator
NLFG	Nonlinear Filter Generator
$\mathcal{Z} = (z)_{t=0}^{\infty}$	Linear recurring sequence
S	Internal state (e.g. state of the LFSR)

Contents

Chapter 1

Introduction

The stream ciphers play really important role in modern symmetric cryptography. They are widely used due to their efficiency and hardware suitability in practice. For example, the stream cipher A5/1 is used in GSM standard, which is used by more than 2 billion people. Recently, eSTREAM - ECRYPT Stream Cipher Project is a multi-year effort to identify new stream ciphers that might become suitable for widespread adoption [1].

An algebraic attack is a very recent cryptanalytic technique which reduces the cryptanalysis of the attacked cryptosystem into problem of finding and solving a system of polynomial equations. Therefore the problem of solving a system of nonlinear equations over finite field is close to the algebraic attack. There is a discussion about applicability of the algebraic attack on various cryptographic primitives at the moment, but there is no doubt about applicability of the algebraic attack on certain stream cipher based on LFSRs.

The main objective of this thesis is to study, explain and illustrate the basic ideas and techniques related to the algebraic attack on stream ciphers. This thesis is organized as follows. In Chapter 2 we present basic ideas behind algebraic attack and describe techniques of generating system of equations for two large classes of the stream ciphers based on LFSRs – nonlinear combination generators and nonlinear filter generators. Afterwards, we try to use algebraic attack directly on the stream cipher A5/1, which is clocked irregularly. We show how to generate the system of equations for A5/1.

Unfortunately, our method is not efficient.

In Chapter 3 and Chapter 4 we present two algorithms, linearization and XL algorithm, for solving the system of nonlinear equations based on linear algebra. We have run large amount of computer simulations for experimental analysis of XL algorithm.

We examine method for solving the system of nonlinear equations using SAT solvers in Chapter 5. As its result we present our heuristic for speeding up a process of converting equations in algebraic normal form to conjunctive normal form. We present experimental results of applying this technique on various systems of equations.

Finally, in Chapter 6 we present and illustrate on the examples an enhancement of the standard algebraic attack – fast algebraic attack.

We have implemented various programs for the purposes of our experiments with algebraic attacks on various stream ciphers. Short descriptions of the implemented programs are in Appendix A. To the best of our knowledge, the program xl, which is implementation of the XL algorithm, described in Chapter 4, is a first public implementation of this algorithm.

Chapter 2

Algebraic Attack – Basic Ideas

Algebraic attack is a very recent attack on various cryptosystems. First paper which introduced algebraic attack is the paper by Kipnis, Shamir [13]. Authors defined new algorithm *"Relinearization"* for solving the system of nonlinear equations over finite field. They used Relinearization for attacking the public key cryptosystems Hidden Fields Equations and Dragon [16]. But the idea of breaking cipher as solving the system of equations is 50 years older. Claude Shannon wrote that breaking a good cipher should require *"as much work as solving a system of simultaneous equations in a large number of unknowns of a complex type"* in his paper [17].

The main principle of the algebraic attack is really simple – converting a problem of attacking cryptosystem (e.g. recovering secret key of symmetric cipher) into solving the system of polynomial equations. The algebraic attack then consists of two basic steps:

1. Find a system of equations.

2. Solve the system of equations.

However there is the fundamental problem of the algebraic attack. It is that solving the system of polynomial equations over any finite field is NP-Complete problem.

Theorem 2.1. *Solving a system of quadratic equations over any finite field (QS) is NP-Complete.*

Proof. It is easy to see that our problem $QS \in NP$ – we guess the solution and check our solution in polynomial time. We find a reduction from well known NP-Complete problem 3SAT to QS. Let $C = C_1 \wedge C_2 \wedge \ldots C_n$ be a Boolean expression where C_i is a clause of the form $C_i = l_{i_1} \vee l_{i_2} \vee l_{i_3}$ and l_{i_j} is a literal. Let us assume that our finite field is $\mathbb{F}_2$. We construct following three equations for each clause C_i:

$$\begin{cases} x_i = l_{i_1} + l_{i_2} + l_{i_3} & \text{one or three literals are true;} \\ y_i = l_{i_1} l_{i_2} + l_{i_2} l_{i_3} + l_{i_1} l_{i_3} & \text{at least two literals are true;} \\ 1 = x_i + y_i + x_i y_i & \text{one or both first equations are satisfied,} \end{cases}$$

where

$$\begin{aligned} l_i &= v_i && \text{if } l_i \text{ is a positive literal of variable } v_i; \\ l_i &= (1 - v_i) && \text{if } l_i \text{ is a negative literal of variable } v_i. \end{aligned}$$

Transformation of the C into the system of quadratic equations is done in polynomial time. Now it is easy to see that if we solve a system of new equations we also get solution for the Boolean expression C. To prove theorem for any finite field we add for each variable v_i another equation:

$$v_i(1 - v_i) = 0 \quad \text{force variable to have value 0 or 1.}$$

□

Following corollary is a trivial consequence of the previous theorem.

Corollary 2.1. *Solving a system of polynomial equations over any finite field is NP-Complete.*

Theorem 2.1 was proved by Valiant and independently by Fraenkel and Yesha. It sounds little bit paradoxical to try to break cryptosystem by converting it to NP-Complete problem. We will show some examples where algebraic attack can be very effective in the next chapters.

2.1 The Attack Scenario

Algebraic attack on the stream ciphers is known-plaintext attack. So our assumption is that the attacker knows some bits of the plaintext and also corresponding bits of the ciphertext at some known positions. For example source of these bits might be fixed header of the encrypted file. If the attacker knows format of the encrypted file and this file format has fixed header (e.g. Excel spreadsheet) it will be enough for him to get necessary bits for the attack. These bits do not have to be consecutive. We follow Kerckhoffs' principle, so the attacker has also complete description of the attacking cryptosystem. In this paper we restrict ourselves to binary stream ciphers (they generate one bit per one step) and to synchronous stream ciphers (they generate next state from the previous one independently of the plaintext), see Figure 2.1. Most of the time we work with binary finite field $\mathbb{F}_2$.

2.2 Generating Equations for Stream Ciphers

We describe how to generate a system of equations for the attacking stream ciphers in this section.

2.2.1 Stream Ciphers

Stream ciphers are very important part of symmetric cryptography. They encrypt each character of plaintext one at a time. Transformation of the cipher varies during the time. Stream ciphers are sometimes called *state ciphers* because encryption of the character depends not only on the key and the character, but also on the current state of stream cipher. Basic idea is to simulate one-time pad (originally called *Vernam cipher* [19]). We initialize the generator of pseudorandom keystream using our secret key. The part of the stream cipher called keystream generator generates pseudorandom sequence of bits. The cipher combines it with plaintext during the encryption and with ciphertext during the decryption, respectively. Generally, some of the main advantages of the stream ciphers are speed and that they are also

usually well-suited for hardware implementation. More about stream ciphers can be found in chapter 6 of [15].

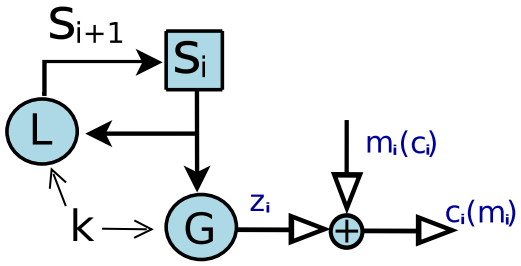

Figure 2.1: General model of synchronous stream cipher, L – next state function, G – produces keystream bits z_i, c_i – ciphertext bits, m_i – plaintext bits, k – key

Currently there is an effort to identify new stream ciphers that might become suitable for widespread adoption. The name of the project is eSTREAM - ECRYPT Stream Cipher Project and is funded by the European Union. More information about this effort can be found at [1].

2.2.2 LFSR-based Stream Ciphers

There are various building blocks for the construction of the stream ciphers. One of the well-known class of the stream ciphers is a class containing ciphers based on linear feedback shift registers.

Definition 2.1. *([15]) A linear feedback shift register (LFSR) of length L consists of L stages (or delay elements) numbered $0, 1, \ldots, L-1$, 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 a feedback function* $f(s_0, s_1, \ldots, s_{n-1})$*, which is equal to adding together modulo* 2 *the previous contents of a fixed subset of stages* $0, 1, \ldots, L-1$*.*

If the content of stage i *is* $s_i \in \{0, 1\}$ *for each* $0 \leq i \leq L-1$*, then* $S = (s_0, s_1, \ldots, s_{L-1})$ *is called the state of the LFSR.*

Note 2.1. *The feedback function* f *of LFSR can be represented as a square matrix* L_f *of dimension* n *over finite field* $\mathbb{F}_2$*. We can obtain state* S_{i+1} *from the previous state* S_i *as a result of multiplication of* S_i *and* L_f*:*

$$S_{i+1} = S_i * L_f.$$

Definition 2.2. *A connection polynomial* $C(x)$ *of LFSR* L *with length* n *is an univariate polynomial over* $\mathbb{F}_2$ *and it is equal to*

$$C(x) = 1 + c_1 x + c_2 x^2 + \ldots + c_n x^n,$$

where

$$c_i = \begin{cases} 1 & \text{if } s_{n-i} \text{ is in the fixed subset of stages of the feedback function;} \\ 0 & \text{otherwise.} \end{cases}$$

Example 2.1. *Let length of LFSR be 3, the feedback function be* $f = s_0 \oplus s_1$ *and* $S_i = (x_0, x_1, x_2)$*. Then matrix* L_f *is*

$$L_f = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix},$$

and the state S_{i+1} *is equal to*

$$S_{i+1} = S_i * L_f = (x_0, x_1, x_2) \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} = (x_1, x_2, x_0 + x_1).$$

LFSRs are well suited to hardware implementation and they are capable

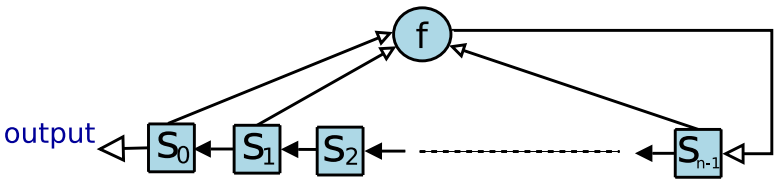

Figure 2.2: A linear feedback shift register

of producing sequences having large periods and good statistical properties. LFSR produces an output sequence with maximum period if a connection polynomial is a primitive polynomial (see Chapter 6 in [15]). Let us use LFSR directly as a keystream generator. Unfortunately, the output bit stream of any LFSR is easily predictable as the following example shows. The attack scenario at this generator is the same as it sis described in the Section 2.1 except for the known keystream bits that have to be consecutive.

Example 2.2. *Suppose a LFSR L with a length 3 and a feedback function $f(s_0, s_1, s_2) = s_0 \oplus s_1$. If attacker knows the length of L , the feedback function f and three consecutive output bits (a, b, c) of L, he can easily predict following output bits of L. The following table shows sequence of the states of L during generation of the known bits (a, b, c).*

State	L	Output
1^{st}	(a, b, c)	a
2^{nd}	$(b, c, a \oplus b)$	b
3^{rd}	$(c, a \oplus b, b \oplus c)$	c

The attacker can easily predict the values of the next two output bits which are $a \oplus b$ and $b \oplus c$.

Generally, for predicting output bit of LFSR with length N the attacker needs to know previous N consecutive output bits. There is also another very efficient attack on LFSR based on Berlekamp-Massey algorithm (see [15]), when the attacker does not even need to know the feedback function f of LFSR. Due to these facts, using LFSR as a keystream generator directly is insecure. We need to add nonlinearity into the design of the stream cipher.

2.2.3 Nonlinear Combination Generators

One of the methods of constructing the stream ciphers from LFSRs is to combine outputs of several LFSRs using a nonlinear combining function. We call the class of these ciphers nonlinear combination generators.

Definition 2.3. *A nonlinear combination generator (NLCG) consists of n LFSRs numbered $0, 1, \ldots, n-1$; a nonlinear Boolean combination function $f(l_0, l_1, \ldots, l_{n-1})$; and a clock which controls the movement of data. During each unit of time the following operations are performed:*

1. *LFSR L_i is updated to next state for $0 \leq i \leq n-1$; and*
2. *result of the function $f(l_0, l_1, \ldots, l_{n-1})$, where l_i is output of the i-th LFSR, is output and forms part of the output sequence.*

Note 2.2. *Every Boolean function $f(x_1, x_2, \ldots, x_n)$ can be written as a modulo 2 sum of distinct m^{th} order products of its variables, $0 \leq m \leq n$; this expression is called the algebraic normal form (ANF) of f (see [15]). Also every $f(x_1, x_2, \ldots, x_n)$ can be viewed as a multivariate polynomial over $\mathbb{F}_2$. Due to these facts we write Boolean functions as polynomials in this work.*

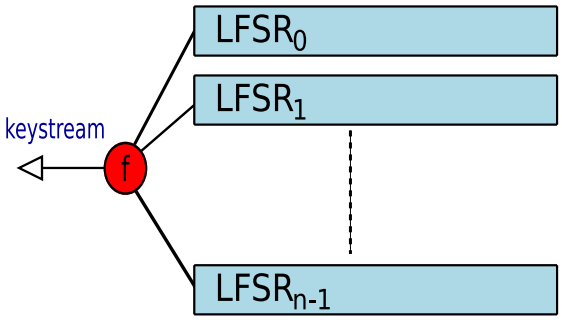

Figure 2.3: A nonlinear combination generator

Unfortunately, there are no known mathematical proofs of security of such stream ciphers so far. Yet, they can not be provably secure. A cipher is said to be provably secure if the difficulty of its defeating can be shown

to be essentially as difficult as solving a well-known and supposedly difficult (typically number-theoretic) problem, such as integer factorization or the computation of discrete logarithms (see [15]). These ciphers can only be deemed computationally secure. A proposed cipher is said to be computationally secure (see [15]) if the perceived level of computation required to defeat it (using the best attack known) exceeds, by a comfortable margin, the computational resources of the hypothesized adversary. That is another reason of our attempt to attack this class of symmetric stream ciphers by algebraic attack. We believe that in some cases the algebraic attack requires less computation than the brute force attack.

The attacker knows some bits of the keystream at some known positions and he wants to generate the system of polynomial equations $\mathcal{A}$, where unknown variables represent the key. Generating the system of polynomial equations for NLCG is straightforward. The unknown variables are initial values of each LFSR. We simulate operation of the keystream generator step by step and for each known output bit we add equation to system $\mathcal{A}$.

Let L_i be a matrix representation of the feedback function and n_i be a length of i-th LFSR, $l_i^{(k)}$ be an output bit of the i-th LFSR in k-th step and $f(l_0, l_1, \ldots, l_{n-1})$ be a nonlinear combination function of NLCG. For k-th known output bit c_k of the cipher we add following equation to our system:

$$f(l_0^{(k)}, l_1^{(k)}, \ldots, l_{n-1}^{(k)}) = c_k.$$

We get $l_i^{(k)}$ as a content of the 0-th stage of the i-th LFSR in k-th step.

$$S_0 = (s_0^{(0)}, s_1^{(0)}, \ldots, s_{n_i-1}^{(0)}), \quad S_0 * L_i^k = (s_0^{(k)}, s_1^{(k)}, \ldots, s_{n_i-1}^{(k)}), \quad l_i^{(k)} = s_0^{(k)}.$$

Example 2.3. *Consider a NLCG with a combination function* $f(l_0, l_1) = l_0 l_1 + l_1$ *and two LFSRs with matrix representations of the feedback functions* L_1 *and* L_2.

$$L_1 = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}, \quad L_2 = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}.$$

The attacker knows the first three keystream bits $(0, 1, 0)$ *of the generator. He will simulate operating of the stream cipher and will get following system of equations.*

State	$\mathbf{L_1}$	$\mathbf{L_2}$	**Equation**
Initial	(x_0, x_1)	(x_2, x_3, x_4)	-
1^{st}	$(x_1, x_0 + x_1)$	$(x_3, x_4, x_2 + x_3)$	$x_0x_2 + x_2 = 0$
2^{nd}	$(x_0 + x_1, x_0)$	$(x_4, x_2 + x_3, x_3 + x_4)$	$x_1x_3 + x_3 = 1$
3^{rd}	(x_0, x_1)	$(x_2 + x_3, x_3 + x_4, x_2 + x_3 + x_4)$	$x_0x_4 + x_1x_4 + x_4 = 0$

2.2.4 Nonlinear Filter Generators

Another class of the stream ciphers based on LFSRs is called nonlinear filter generators. The keystream generator consists of only one LFSR and a nonlinear filtering function in this class.

Definition 2.4. *A nonlinear filter generator (NLFG) consists of one LFSR* L *with length* n*; a nonlinear filtering Boolean function* $f(s_0, s_1, \ldots, s_{n-1})$*; and a clock which controls the movement of data. During each unit of time the following operations are performed:*

1. *LFSR* L *is updated to next state; and*
2. *result of the function* $f(s_0, s_1, \ldots, s_{n-1})$*, where* s_i *is content of* i*-th stage of the LFSR, is output and forms part of the output sequence.*

Generating the system of polynomial equations for this class of stream ciphers is similar to generating equations for NLCG.

Example 2.4. *Consider a NLFG with a filter function* $f(s_0, s_1, s_2) = s_0s_2 + s_1s_2 + s_1$ *and with one LFSR with matrix representation of the feedback function* L_1*.*

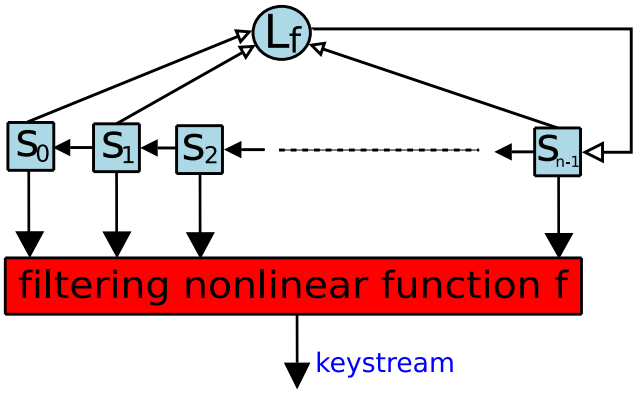

Figure 2.4: A nonlinear filter generator

$$L_1 = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}.$$

The attacker knows the first three keystream bits (0, 1, 1) of the generator. He will simulate operation of the keystream generator and get following system of equations as well as he does for the NLCG.

State	**L**	**Equation**
Initial	(x_0, x_1, x_2)	-
1^{st}	$(x_1, x_2, x_0 + x_1)$	$x_0x_1 + x_0x_2 + x_1x_2 + x_1 + x_2 = 0$
2^{nd}	$(x_2, x_0 + x_1, x_1 + x_2)$	$x_0x_1 + x_0x_2 + x_0 + x_2 = 1$
3^{rd}	$(x_0 + x_1, x_1 + x_2, x_0 + x_1 + x_2)$	$x_0x_1 + x_1x_2 + x_0 + x_1 = 1$

From the previous two examples, Example 2.3 and Example 2.4, we can see that a degree of the generated equations is upper bounded by the degree of combining or filtering function f. Content of any stage of LFSR is always linear expression and we substitute linear expressions into function f. The maximal degree of the equations is important factor of the algebraic attack. Generally, the algebraic attack is more effective for the lower degree.

The important fact is that a generation of the system of polynomial equations usually can be done as a precomputation step. The computers have

enough space to store database of equations for any cipher these days. So for the algebraic attacks, when we work with the equations in ANF, the most time consuming part of the attack is the second step – solving the system of equations. There can be exception, for example when we use SAT solver for solving equations. We have to transform equations into *conjunctive normal form* (CNF). We will discuss this case in Chapter 5.

2.3 Generating System of Equations for A5/1

A5/1 is the stream cipher used in the GSM cellular telephone standard, which is widely used technology for the mobile communication. It has not been published publicly. The algorithm was entirely described by Marc Briceno in 1999. We take the definition of A5/1 from [5].

A5/1 does not belong to any previously described class of the stream ciphers, but it is partially similar to NLCG. It contains three LFSRs L_1, L_2 and L_3 with length 19, 22 and 23, respectively (see Figure 2.5). The size of the key is 64 bits (in GSM implementation 10 bits are fixed to 0). The combining function is a linear Boolean function $f(l_0, l_1, l_2) = l_0 + l_1 + l_2$. Till this point, A5/1 looks vulnerable to algebraic attack, because the generated equations are upper bounded by the degree of the combining function $f(l_0, l_1, l_2)$ and we can solve the system of linear equations effectively. But LFSRs of A5/1 are not clocked regularly. There is used majority rule for clocking the registers. Each register has a clocking bit v_i ($v_1 = s_8^1$, $v_2 = s_{10}^2$ and $v_3 = s_{10}^3$). At each cycle, majority bit is determined (if at least two clocking bits are equal to 1, then the majority bit is equal to 1, otherwise it is set to 0). LFSR L_i is clocked at cycle, if his clocking bit is equal to majority bit.

We use the same trick, as it has been used in proof of Theorem 2.1, for generating the system of equations for A5/1.

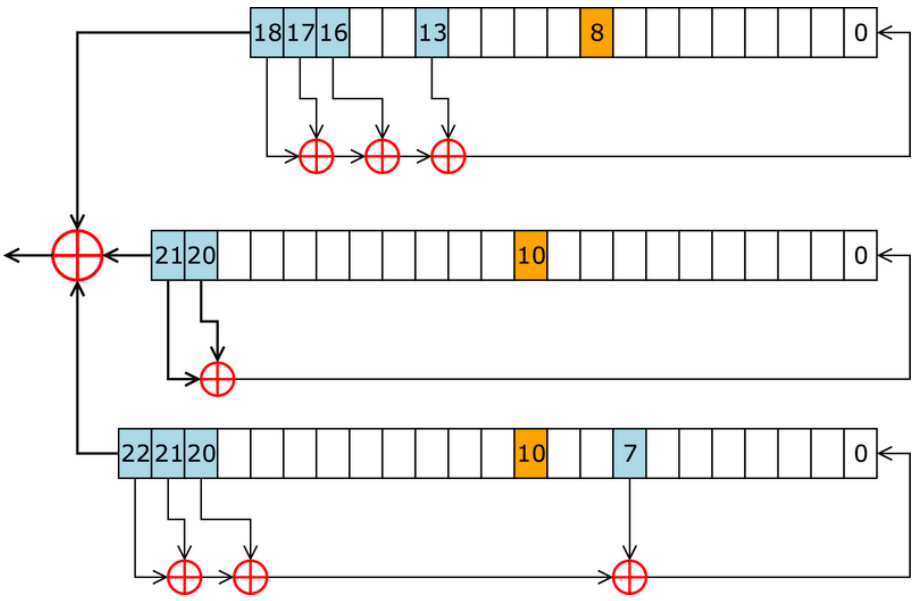

Figure 2.5: A5/1 (source: http://www.wikipedia.org)

$$V = (v_0v_1 + v_1v_2 + v_0v_2) = \begin{cases} 1 & \text{at least 2 of the clocking bits are equal to 1;} \\ 0 & \text{at least 2 of the clocking bits are equal to 0.} \end{cases}$$

Hence, V represents the value of the majority bit. Now we can easily write how to transform i-th LFSR from the current state S_j^i to the next state S_{j+1}^i for $i = 1, 2, 3$:

$$S_{j+1}^i = V(v_i S_j^i * L_i + (1 - v_i)S_j^i) + (1 - V)((1 - v_i)S_j^i * L_i + v_i S_j^i).$$

Finally, we can start generating the system of equations. We fill the LFSRs by fresh variables, which represent the initial configuration of the stream cipher, and similar to previous generating, we will simulate operation of the stream cipher and for each known keystream bit a_k, we will generate the equation of the form $f(l_0, l_1, l_2) = a_k$, where l_i is substituted by actual content of the stage s_0^i. The problem of this method is that there is no upper boundary for the degree of the generated equations. There is only upper bound by the number of variables 64. Therefore the number of the mono-

mials which may appear in the system of equations is $\sum_{k=1}^{64} \binom{64}{k} = 2^{64} - 1$. The size of the generated equations is also exponentially large. Actually, we capture the decision of clocking or not clocking the registers in the generated equations. The number of possibilities is exponential (at each cycle there are 4 possibilities of clocking the LFSRs). Hence the size of the equation has to grow exponentially.

Example 2.5. *For simplicity, let us have a stream cipher with the same design as A5/1 has. Except that, all LFSRs are equal with the length 3. Let L be a matrix representing a feedback function for each LFSR.*

$$L = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}.$$

The clocking bit for each LFSR is the first stage of LFSR. The stream cipher generates an output bit and then clock the LFSRs using majority rule at each cycle. The attacker knows first two output bits $(0, 1)$ of the keystream. He wants to recover the initial state of the LFSRs. He will fill the LFSRs by the unknown variables and will generate the following two equations:

State	$\mathbf{L_1}$	$\mathbf{L_2}$	$\mathbf{L_3}$	**Equation**
1^{st}	(x_0, x_1, x_2)	(y_0, y_1, y_2)	(z_0, z_1, z_2)	$x_0 + y_0 + z_0 = 0$
2^{nd}	(s_0^1, s_1^1, s_2^1)	$(s_0'^1, s_1'^1, s_2'^1)$	$(s_0''^1, s_1''^1, s_2''^1)$	$f(s_0^1, s_0'^1, s_0''^1) = 1$

$$s_0^1 = x_1y_1x_0 + x_1z_1x_0 + y_1z_1x_0 + x_1x_0 + x_1y_1 + x_1z_1 + y_1z_1x_1$$
$$s_1^1 = x_1y_1 + x_1z_1 + y_1z_1x_1 + x_1 + x_2 + x_1x_2 + x_1y_1x_2 + x_1z_1x_2 + y_1z_1x_2$$
$$s_2^1 = x_1y_1x_0 + x_0 + x_1x_2 + x_1x_0 + x_1z_1x_0 + y_1z_1x_1 + y_1z_1x_2 + x_1y_1 +$$
$$+ x_1z_1 + y_1z_1x_0 + x_1y_1x_2 + x_1z_1x_2$$

$$
\begin{aligned}
s_0'^1 &= x_1y_1y_0 + x_1z_1y_0 + y_1z_1y_0 + y_1y_0 + x_1y_1 + y_1z_1x_1 + y_1z_1 \\
s_1'^1 &= x_1y_1 + y_1z_1x_1 + y_1z_1 + y_1 + y_2 + y_1y_2 + x_1y_1y_2 + x_1z_1y_2 + y_1z_1y_2 \\
s_2'^1 &= y_1z_1y_0 + y_0 + y_1y_2 + y_1y_0 + x_1y_1y_0 + y_1z_1x_1 + x_1z_1y_2 + x_1z_1y_0 + \\
&\quad + x_1y_1 + y_1z_1 + x_1y_1y_2 + y_1z_1y_2 \\
s_0''^1 &= x_1y_1z_0 + x_1z_1z_0 + y_1z_1z_0 + z_1z_0 + y_1z_1x_1 + x_1z_1 + y_1z_1 \\
s_1''^1 &= y_1z_1x_1 + x_1z_1 + y_1z_1 + z_1 + z_2 + z_1z_2 + x_1y_1z_2 + x_1z_1z_2 + y_1z_1z_2 \\
s_2''^1 &= x_1z_1z_2 + y_1z_1z_2 + y_1z_1z_0 + x_1z_1z_0 + z_0 + x_1y_1z_0 + z_1z_2 + z_1z_0 + \\
&\quad + y_1z_1x_1 + x_1z_1 + y_1z_1 + x_1y_1z_2 \\
f(s_0^1, s_0'^1, s_0''^1) &= x_1y_1x_0 + y_1z_1y_0 + y_1z_1z_0 + x_1z_1z_0 + x_1y_1z_0 + x_1x_0 + y_1y_0 + \\
&\quad + z_1z_0 + x_1z_1x_0 + x_1y_1y_0 + y_1z_1x_1 + x_1z_1y_0 + y_1z_1x_0 = 1
\end{aligned}
$$

We see that the size of the generated equations will grow exponentially from our simple example. The usage of memory for storing the LFSRs will grow exponentially as well.

Precisely, the number of possible generated equations with n variables over $\mathbb{F}_2$ is upper bounded by $2^{\sum_{i=0}^{n}\binom{n}{i}}$. Hence the size of the generated equation is also upper bounded and it will not grow to infinity. Unfortunately, the size of the equations grows very fast from the beginning and the upper bound has no relevant impact on the efficiency of this technique. Therefore the size of the generated equations grows exponentially for the attacker's point of view.

We have presented method for generating equations for the stream cipher A5/1 required for algebraic attack. Unfortunately, the described method is not effective. On the other side, there have been presented other attacks [8] and [12], which are very effective. Some of presented attacks require to solve the system of equations. Therefore the effective methods for solving the system of polynomial equations discussed in this work may improve the attacks on the A5/1.

Chapter 3

Linearization

The basic and well known technique for solving the system of multivariate polynomial equations is called *linearization*. Consider the problem of solving the overdefined system of equations where the number of the equations is approximately the same as the number of the terms occurring in the equations. Then we can solve the system by using following algorithm:

Algorithm 1 Linearization

1: Substitute any product of variables by fresh variable.
2: Solve the linear system (e.g. using Gaussian elimination).
3: Plug the solution into the original system and check correctness of the solution.

Example 3.1. *Consider the problem of solving the following system of equations:*

$$\begin{aligned} xy + y &= 0 \\ xy + x + y &= 1 \\ xy &= 1 \end{aligned}$$

In step 1 we replace term xy with fresh variable z and we get the following system of linear equations:

$$\begin{aligned} z + y &= 0 \\ z + x + y &= 1 \\ z &= 1 \end{aligned}$$

By solving the system of linear equations we get the following solution:

$$x = 1, \quad y = 1, \quad z = 1.$$

At the end, we check the correctness of the solution:

$$\begin{aligned} 1 * 1 + 1 &= 0 \\ 1 * 1 + 1 + 1 &= 1 \\ 1 * 1 &= 1 \end{aligned}$$

Example 3.2. *Consider the problem of solving the following system of equations:*

$$\begin{aligned} xy + x + y &= 1 \\ x + y &= 0 \\ xy + y &= 1 \end{aligned}$$

In step 1 we replace term xy with fresh variable z and we get the following system of linear equations:

$$\begin{aligned} z + x + y &= 1 \\ x + y &= 0 \\ z + y &= 1 \end{aligned}$$

By solving the system of linear equations we get the following solution:

$$x = 0, \quad y = 0, \quad z = 1.$$

At the end, we check the correctness of the solution:

$$\begin{aligned} 0 * 0 + 0 + 0 &\neq 1 \\ 0 + 0 &= 0 \\ 0 * 0 + 0 &\neq 1 \end{aligned}$$

This solution is not correct. The example shows that step 3 of linearization method is necessary.

Solving the linear system of equations over finite field runs in polynomial time (e.g. $O(n^3)$ if we use Gaussian elimination). There are various methods for solving large sparse linear systems which are more efficient than Gaussian elimination. In this work we concentrate on the methods for solving the non-linear systems therefore we use only Gaussian elimination for simplicity and we do not try to improve algorithms using better methods for solving linear systems. Comparison of the algorithms for solving linear systems related to algebraic attack on the stream ciphers can be found in [7].

Drawback of linearization is that the attacker needs to generate strongly overdefined system. Let n be a number of variables and m be the highest degree of equations. Generally, we can end up with approximately

$$\sum_{i=2}^{m} \binom{n+i-1}{i}$$

new variables. In case of solving system over $\mathbb{F}_2$ the number of new variables is lower than in general case,

$$\sum_{i=2}^{m} \binom{n}{i}$$

because of the equation $a^2 = a$. It means that the attacker needs to know huge amount of the keystream for attacking stream cipher. The number of new variables has also impact on running time of linearization.

Example 3.3. *Let us have NLFG with LFSR of length 128 and a filtering Boolean function with degree 8. By linearizing system of equations for this cipher we can end up with*

$$\sum_{i=2}^{8} \binom{128}{i} \approx 2^{40}$$

new variables so the attacker needs to have approximately 128 GB of known keystream bits. Running time of linearization is approximately 2^{120} *steps, what is just little lower than running time of brute force attack.*

During the analysis of linearization we omitted the fact that the linearized system can contain linear dependent equations. There have been proposed several improvements as Relinearization [13] and the XL algorithm [10]. These methods try to cut back the number of known keystream bits. Relinearization is an intermediate step between linearization and the XL algorithm. In [10] authors have proved that the XL algorithm "contains" Relinearization so we leave out elaboration of this technique and we concentrate on the XL algorithm.

Chapter 4

The XL Algorithm

Another technique for solving the polynomial system of equations is called the XL algorithm (XL stands for an eXtended Linearization). The XL algorithm has been proposed by Courtois, Klimov, Patarin and Shamir in [10]. The XL tries to benefit from the fact that the number of equations exceeds the number of variables. The main idea is to increase the number of initial equations by adding the new algebraically dependent equations, which are linearly independent to initial system.

Let $\mathbb{F}$ be a field (field does not have to be finite), let $\mathcal{A}$ be a system of multivariate polynomial equations $l_k = 0 \quad (1 \leq k \leq m)$ where each l_k is multivariate polynomial $l_k = f_k(x_1, x_2, \ldots, x_n) - b_k$, and let $d_{\max}$ be a maximal degree of the polynomials in $\mathcal{A}$. The system $\mathcal{A}$ has at least one solution. The problem is to find at least one solution $x = (x_1, x_2, \ldots, x_n) \in \mathcal{K}^n$, for a given $b = (b_1, b_2, \ldots, b_m) \in \mathbb{F}^m$. In case of attacking stream cipher, b represents known keystream bits and $\mathcal{A}$ represents corresponding equations generated for example as it was described in Chapter 2. In case of $\mathbb{F}$ being a finite field, we suppose the powers of variables taken from $1, 2, \ldots q-1$, where q is the characteristic of field $\mathbb{F}$, because of the equation $a^q = a$. We denote X^k the set of all terms of degree exactly k, $X^k = \{x_{i_1}^{e_1} x_{i_2}^{e_2} \ldots x_{e_m}^{e_m} | i_j = 1 \ldots n, e_1 + e_2 + \ldots e_m = k\}$, $X^0 = \{a | a \in \mathbb{F}\}$.

Definition 4.1. *A monomial over field* $\mathbb{F}$ *is a multivariate polynomial of the following form:*

$$a * x_{i_1}^{e_1} x_{i_2}^{e_2} \dots x_{i_m}^{e_m} \quad a \in \mathbb{F}, e_j \in \mathbb{N}, i_j \in \mathbb{N}.$$

The authors of [10] have defined the XL algorithm for solving quadratic systems ($d_{\max} = 2$). Courtois has described obvious extension of the XL algorithm for solving the system with $d_{\max} \geq 2$ in [9]. We present extended version of the XL algorithm in this work.

Definition 4.2. *(The XL algorithm) Algorithm input parameters are the system of equations* $\mathcal{A}$*, which has at least one solution, and* $D \in \mathbb{N}$*, which is upper bound for the equations used in XL algorithm. Execute the following steps:*

1. **Multiply:** *Generate the new system* $\mathcal{A}'$*:*

$$\mathcal{A}' = \bigcup_{0 \leq k \leq D - d_{\max}} X^k * \mathcal{A}.$$

2. **Linearize:** *Consider each monomial in the variables* x_i *of degree* $\leq D$ *as a new variable and perform Gaussian elimination on the system* $\mathcal{A}'$*. The ordering on the monomials must be such, that all the terms containing one variable (say* x_1*) are eliminated last.*

3. **Solve:** *Assume that step 2 yields at least one univariate in the powers of* x_1*. Solve this equation over the finite field* $\mathbb{F}$ *(e.g. with Berlekamp's algorithm or Cantor-Zassenhaus's algorithm, for more information about the algorithms see [18]).*

4. **Repeat:** *Simplify the equations and repeat the process to find values of the other variables.*

All operations, except for solving univariate equation over $\mathbb{F}$, in the XL algorithm are done by linear algebra. The XL algorithm tries to find a basis of the linear vector space $V(\mathbb{F})$ (described in the Lemma 4.1), so that univariate polynomial belongs to it.

Lemma 4.1. *Let $\mathcal{A} = \{f_1, f_2, \ldots, f_m\}$ be a system of multivariate equations over finite field $\mathbb{F}$, let $D \in \mathbb{N}$ and let V be a set equal to:*

$$V = \{g \cdot f_i | f_i \in \mathcal{A}, g \in \mathbb{F}[x_1, x_2, \ldots, x_n], \deg(g \cdot f_i) \leq D\}.$$

Then a set $V(\mathbb{F})$ of the linear combinations of the elements from V is a vector space over $\mathbb{F}$, where $+ : V \times V \to V$ is a standard polynomial addition and $* : \mathbb{F} \times V \to V$ is a multiplying polynomial by the constant.

Proof. $(V(\mathbb{F}), +)$ has to be Abelian group. Let $\alpha, \beta \in V \Rightarrow \alpha = c_1 \cdot f_1 + \ldots + c_m \cdot f_m, \beta = d_1 \cdot f_1 + \ldots + d_m \cdot f_m$, $c_1, \ldots, c_m, d_1, \ldots, d_m \in \mathbb{F}[x_1, x_2, \ldots, x_n]$.

$$\alpha + \beta = c_1 \cdot f_1 + \ldots + c_m \cdot f_m + d_1 \cdot f_1 + \ldots + d_m \cdot f_m =$$

$$= (c_1 + d_1) \cdot f_1 + \ldots + (c_n + d_m) \cdot f_m \in V(\mathbb{F}).$$

Other properties of the Abelian group $(V(\mathbb{F}), +)$ can be easily shown from the properties of the addition of polynomials. Also the following properties can be easily proved, for any $c, d \in \mathbb{F}$ and $\alpha, \beta \in V(\mathbb{F})$:

1. $c * (\alpha + \beta) = c * \alpha + c * \beta$,
2. $(c + d) * \alpha = c * \alpha + d * \alpha$,
3. $(c \cdot d) * \alpha = c * (d * \alpha)$,
4. $1 * \alpha = \alpha$.

□

We remark that any solution $s = (s_1, s_2, \ldots, s_n)$ of the system $\mathcal{A}$, is also the solution for any equation of the form: $h(x) = 0, h(x) \in V(\mathbb{F})$.

$$h(s) = c_1(s) \cdot f_1(s) + \ldots + c_m(s) \cdot f_m(s) = c_1(s) * 0 + \ldots + c_m(s) * 0 = 0.$$

Hence step 1 of the XL algorithm does not add any new solution to the system $\mathcal{A}'$. Therefore the solution for the system $\mathcal{A}'$ is also the solution for the system $\mathcal{A}$.

The XL algorithm generates the elements of the vector space $V(\mathbb{F})$ in step 1. In step 2 algorithm finds one of the basis of $V(\mathbb{F})$. If at least one element of the founded basis is a univariate polynomial, we solve this equation and iterate this process.

Note 4.1. (Solving the univariate equations) *In case of solving the system over $\mathbb{F}_2$, the only possible univariate equations yielded by step 2 have form $x_i = a_i, a_i \in \mathbb{F}_2$. Hence step 3 for this case is very simple and we do not have to use any univariate solver. The univariate equations yielded by step 2 may have more than one solution for other finite fields. The authors of the XL algorithm have not described how to handle this situation.*

Note 4.2. *It is not always necessary to work with X^k for each $k \leq D - d_{\max}$, but it is sometimes more efficient to work only with a subset of the monomials. For example, if we have the quadratic homogeneous system, it is sufficient to work only with X^k for $k \leq D-2$ and k is even. We get only monomials with even degree in the system $\mathcal{A}'$. Using only subset of monomials has impact on the speed of the steps 1 and 2.*

Note 4.3. *If step 2 yields more than one univariate equation, it is obvious to solve all yielded univariate equations and speed up the process of solving the system. If step 2 does not yield any univariate equation, the XL algorithm fails.*

Example 4.1. *Consider the problem of solving the following system of equations:*

$$\begin{aligned} x_0x_1 + x_1 &= 0 \\ x_1x_2 + x_0 + x_2 &= 0 \\ x_0x_2 + x_1 &= 1 \\ x_0x_1 + x_0x_2 + x_0 &= 0 \end{aligned}$$

We set up $D = 4$ and we use only X^2 for generating the products as we have

mentioned in Note 4.2. In step 1 we get the following new equations:

$$x_0x_1 = 0$$
$$x_0x_1x_2 = 0$$
$$x_0x_1x_2 + x_1x_2 = 0$$

In step 2 we linearize the system of equations and we solve the linearized system (new variables y_i are eliminated first).

$$x_0x_1 = y_0, \quad x_1x_2 = y_1, \quad x_0x_2 = y_2, \quad x_0x_1x_2 = y_3.$$

$$\left(\begin{array}{ccccccc|c} y_0 & y_1 & y_2 & y_3 & x_0 & x_1 & x_2 & \\ \hline 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \end{array}\right) \sim \left(\begin{array}{ccccccc|c} y_0 & y_1 & y_2 & y_3 & x_0 & x_1 & x_2 & \\ \hline 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{array}\right)$$

Step 2 yields three univariate equations. Hence we get the solution for all unknown variables after one iteration:

$$x_0 = 1, \quad x_1 = 0, \quad x_2 = 1.$$

4.1 Analysis of the XL Algorithm

It is obvious that for a higher D, the XL algorithm generates more new equations so we have a better chance of finding solution. On the other side, higher D has a negative impact on the time complexity of the steps 1 and 2. Generally, we can start with the lowest possible $D = d_{\max} + 1$ and if the XL algorithm fails, we increase D by one and try again. We present boundaries for the parameter D in the following part.

In case of solving the system over $\mathbb{F}_2$, the upper bound for the parameter D is n – the number of the variables. Generally, for finite field $\mathbb{F}$ with characteristic q, the parameter D is upper bounded by $(q-1)n$.

Let R be the number of equations in the system after the step 1 and let T be the number of all monomials which may appear in the system of equations. We assume that $D \leq \frac{n}{2}$. In case of solving the system over $\mathbb{F}_2$ we have:

$$R = m + m \cdot \left(\sum_{i=1}^{D-d_{\max}} \binom{n}{i} \right) \approx m \cdot \binom{n}{D-d_{\max}},$$

$$T = \sum_{i=1}^{D} \binom{n}{i} \approx \binom{n}{D}.$$

In general case, we have:

$$R' = m + m \cdot \left(\sum_{i=1}^{D-d_{\max}} \binom{n+i-1}{i} \right) \approx m \cdot \binom{n+D-d_{\max}}{D-d_{\max}},$$

$$T' = \sum_{i=1}^{D} \binom{n+i-1}{i} \approx \binom{n+D}{D}.$$

We expect that a linearized system has a solution, if $R \geq T$. For simplicity, we use the approximation (4.1) in the following derivation of a lower bound of the parameter D:

$$\binom{n}{k} \sim \frac{n^k}{k!}. \tag{4.1}$$

For example, we have $n = 128$ in case of attacking stream cipher with 128 bit key. However the parameter D will be much less than 128, $D \ll 128$. The complexity of Gaussian elimination on the generated system $\mathcal{A}'$ by XL algorithm for $D \sim n$ will be much more than the complexity of brute force attack at the stream cipher. Therefore, the approximation for the binomial coefficients is sufficient for our derivation. We get the lower bound for the

parameter D in case of $\mathbb{F}_2$ from R and T:

$$R \geq T \Rightarrow m \geq \frac{\binom{n}{D}}{\binom{n}{D-d_{\max}}}.$$

Using (4.1) we get

$$m \geq \frac{\frac{n^D}{D!}}{\frac{n^{D-d_{\max}}}{(D-d_{\max})!}} = \frac{n^{d_{\max}}}{D(D-1)\dots(D-d_{\max}+1)},$$

therefore

$$\frac{n^{d_{\max}}}{m} \leq D(D-1)\dots(D-d_{\max}+1) \leq D^{d_{\max}}.$$

Finnally, we get desired formula:

$$D \geq \frac{n}{m^{\frac{1}{d_{\max}}}}.$$

Unfortunately, not all generated equations are linearly independent in practice. Let R_{Free} be the exact number of linearly independent equations in the system generated by the XL algorithm after step 1. We have $R_{Free} \leq R$ and $R_{Free} \leq T$. The main heuristics behind the XL algorithm is that for some values of D we get always $R \geq T$. Then we expect that $R_{Free} \approx T$. When $R_{Free} \geq T-n$, step 2 of the XL algorithm yields at least one univariate equation.

4.2 Experimental Analysis of XL for $d_{\max} \geq 2$

The exact number of linearly independent equations is somewhat complex to predict. Courtois has presented following conjecture about behaviour of XL for $D < 3d_{\max}$ and $\mathbb{F}_2$ in [9].

Conjecture 4.1. *Behaviour of XL for* $D < 3d_{\max}$

1. *For* $D = d_{\max}, \dots, 2d_{\max} - 1$ *there are no linear dependencies when* $R \geq T$ *and we have* $R_{Free} = \min(T, R) - \epsilon$ *with* $\epsilon \in \{0, 1, 2, 3\}$.

2. *For* $D = 2d_{\max}, \ldots, 3d_{\max}-1$ *there are linear dependencies and we have* $R_{Free} = \min(T, R - \left(\sum_{i=0}^{D-2d_{\max}} \binom{n}{i}\right)(\binom{m}{2} + m)) - \epsilon$ *with* $\epsilon \in \{0, 1, 2, 3\}$.

For more information about Conjecture 4.1 see [9]. The author has derived this conjecture from computer simulations. We have done various computer simulations using our programs eq-gen and xl (see Appendix A) with different parameters $d_{\max}$ and D as they have been chosen in [9]. Selected results of our simulations are presented in Appendix B. Our results confirm the Conjecture 4.1.

Chapter 5

Solving the System of Equations using SAT Solvers

We have mentioned that every Boolean function can be viewed as a multivariate polynomial over $\mathbb{F}_2$ in Chapter 2. Hence we work with Boolean expressions as the polynomials and we use algebraic methods for solving the systems (in case of algebraic attack on the stream cipher, solution represents the key used for encryption). We choose different approach of solving the system of equations over $\mathbb{F}_2$ in this chapter. We use SAT solvers for this purpose.

The SAT solver is a program for solving the Boolean satisfiability problem, also known as SAT problem. The solver takes a Boolean expression as an input and finds a satisfying assignment for the variables of the input, if that assignment exists. SAT problem is NP-Complete (k-SAT is the same problem as SAT, except that the input has to be in CNF and it has a restriction k on the maximum number of literals in one clause, we know how to solve k-SAT problem in polynomial time only for $k \leq 2$). Therefore the running time of SAT solver is exponential in the worst case. It again sounds paradoxical to try to solve the system of polynomial equations using SAT solver, because 3SAT problem is used for proving that the problem QS is NP-Complete in Theorem 2.1. There have been written many papers about theory and practice of designing SAT solver. Many practical hard problems

are solved by converting them into SAT problem, because a lot of effort has been made on implementation the good SAT solvers. The disadvantage of this approach is that converting problem into SAT may be inefficient. The design and the implementation of the good SAT solver is completely out of the scope of this work. We used MiniSAT 2.0 solver [2] in this work. MiniSAT 2.0 is the winner of the annual SAT race in 2006 [3].

The attacker generates a system of nonlinear equations $\mathcal{A}$ as it was described in Chapter 2. All generated equations are in ANF. The problem of using SAT solver for solving $\mathcal{A}$ is that SAT solvers must have the Boolean expression in CNF on the input. Therefore we have to add an extra step for converting equations in ANF into CNF.

5.1 Converting ANF to CNF

We have the system of polynomial equations

$$\mathcal{A} = \{f_i(x_1, x_2, \ldots, x_n) = 0 |\ i \in 0, 1, \ldots, m-1\}$$

over $\mathbb{F}_2$ in ANF. The system $\mathcal{A}$ can be viewed as the system of the Boolean expressions. We want to generate a Boolean expression B in CNF, where the solution for the satisfiability of B is also solution for the system $\mathcal{A}$. We will use the following algorithm:

Algorithm 2 Converting system $\mathcal{A}$ into Boolean expression B

1: Convert the $f_i(x_1, x_2, \ldots, x_n)$ into Boolean expression B_i in CNF, where the solution for B_i is also the solution for equation $f_i(x_1, x_2, \ldots, x_n) = 0$ for each $i \in 0, 1, \ldots, m-1$.
2: Put the following Boolean expression $B = B_0 \wedge B_1 \wedge \ldots \wedge B_{m-1}$ as an output of the algorithm (B is in CNF).

SAT solvers do not accept the constants on the input, but the system $\mathcal{A}$ may have a polynomial with the constant 1. We present two possibilities how to handle constants during the conversion. We can add new variable T (T

will represent the constant 1 and $\neg T$ will represent 0). We will use variable T instead of the constant 1 and then B_i is equal to:

$$B_i = CNF(\neg f_i(x_1, x_2, \ldots, x_n)) \wedge T.$$

It is easy to see that variable T is equal to 1 in the satisfying solution. The satisfying solution for B_i is also solution for the equation $f_i(x_1, x_2, \ldots, x_n) = 0$, because

$$\neg f_i(x_1, x_2, \ldots, x_n) \wedge 1 = 1 \quad \Rightarrow \quad f_i(x_1, x_2, \ldots, x_n) = 0.$$

The another approach for handling the constant 1 is the following one: we ignore constant 1 during the conversion and then B_i is equal to:

$$B_i = \begin{cases} CNF(f_i(x_1, x_2, \ldots, x_n)) & \text{if the constant 1 is in } f_i(x_1, x_2, \ldots, x_n); \\ CNF(\neg f_i(x_1, x_2, \ldots, x_n)) & \text{otherwise.} \end{cases}$$

Again, it is easy to see that the solution for B_i is also the solution for the equation $f_i(x_1, x_2, \ldots, x_n) = 0$.

There is well known standard algorithm for converting any Boolean expression into CNF for the step 1. We present his modification for converting ANF into CNF.

Algorithm 3 Converting ANF into CNF

1: Eliminate exclusive or $\oplus$ using following identity:

$$a \oplus b \iff (a \vee b) \wedge (\neg a \vee \neg b).$$

2: Drive in negation $\neg$ using De Morgan's Laws:

$$\neg(a \vee b) \iff \neg a \wedge \neg b;$$

$$\neg(a \wedge b) \iff \neg a \vee \neg b.$$

3: Distribute disjunction $\vee$ over conjunction $\wedge$:

$$a \vee (b \wedge c) \iff (a \vee b) \wedge (a \vee c).$$

Generally, the size of the expression converted to CNF may be exponential. Therefore running time of any converting algorithm is at least exponential in the worst case. The step 1 of the Algorithm 3 doubles the size of the result. Therefore converting Boolean expression in ANF into CNF is the case, where the size of the result is exponentially large.

We see that the process of creating an input for SAT solver will run in exponential time. On the other side, the attacker may generate a database of equations converted to CNF. For each equation of form $f_i(x_1, x_2, \ldots, x_n) = a_i$, he will add 2 triplets into database:

1. $(i, 0, C_{i,0})$, $C_{i,0} = CNF(\neg f_i(x_1, x_2, \ldots, x_n))$.
2. $(i, 1, C_{i,1})$, $C_{i,1} = CNF(f_i(x_1, x_2, \ldots, x_n))$.

The attacker has the knowledge of some keystream bits and their positions $I = \{(i_1, a_1), \ldots (i_k, a_k) | a_j \in \mathbb{F}_2\}$ according to the attack scenario. Hence he will generate the input B for SAT solver by reading the corresponding converted forms of equations from the database. Such process is very efficient.

$$B = \bigwedge_{(i,a)\in I} C_{i,a} \quad \text{where } (i, a, C_{i,a}) \text{ is triplet stored in the database.}$$

Example 5.1. *Let the first two equations of a system $\mathcal{A}$ be:*

$$x_0x_1 + x_0 = a_0$$
$$x_1x_2 + x_0 = a_1$$

where $a_0, a_1 \in \mathbb{F}_2$ represent the keystream bits. The attacker will store the following four triplets into his database for the system $\mathcal{A}$.

1. $(0, 0, C_{0,0}) = (0, 0, (\neg x_0 \vee x_1))$.
2. $(0, 1, C_{0,1}) = (0, 1, (x_0 \vee x_1) \wedge (x_0 \vee x_2) \wedge (\neg x_0 \vee \neg x_1 \vee \neg x_2))$.
3. $(1, 0, C_{1,0}) = (1, 0, (\neg x_0 \vee x_1) \wedge (\neg x_0 \vee x_2) \wedge (x_0 \vee \neg x_1 \vee \neg x_2))$.
4. $(1, 1, C_{1,1}) = (1, 1, (x_0 \vee x_1) \wedge (x_0 \vee x_2) \wedge (\neg x_0 \vee \neg x_1 \vee \neg x_2))$.

Let the first two output bits be $a_0 = 0$ and $a_1 = 1$, the attacker will put on the input of the SAT solver the Boolean expression B which equals to:

$$B = C_{0,0} \wedge C_{1,1} = (\neg x_0 \vee x_1) \wedge (x_0 \vee x_1) \wedge (x_0 \vee x_2) \wedge (\neg x_0 \vee \neg x_1 \vee \neg x_2).$$

The solver will return the following solution:

$$x_0 = 0, \quad x_1 = 1, \quad x_2 = 1.$$

5.2 Improving the Efficiency of ANF to CNF Conversion

Let $n_{\mathcal{A}}$ denote the number of appearances of the xor operation $\oplus$ in the system $\mathcal{A}$ and let $n_{\mathcal{A}}^{\max}$ denote the maximal number of appearances of the xor operation in one equation from $\mathcal{A}$. We have mentioned that the first step of the conversion algorithm has the biggest impact on the size of the output in previous section. For each appearance of the operation xor, the first step of Algorithm 3 will double the size of the output. Hence the system $\mathcal{A}$, which has lower parameters $n_{\mathcal{A}}$ and $n_{\mathcal{A}}^{\max}$ than the system $\mathcal{B}$, will be converted into CNF more efficiently (although the running time will be still exponential). We present our simple technique how to find a new system $\mathcal{A}'$, which has the same solution as the original system $\mathcal{A}$, but for the parameters of the new system may stand $n_{\mathcal{A}} < n_{\mathcal{A}'}$ and $n_{\mathcal{A}}^{\max} < n_{\mathcal{A}'}^{\max}$ in this section.

The main idea behind our heuristic is very simple. Let us have a system $\mathcal{A}$, which contains the following two equations:

$$x_0x_1 + x_1x_2 + x_2 + 1$$
$$x_1x_2 + x_2 + x_3$$

If we add the second equation to the first one, we get the new system $\mathcal{A}'$, which has both parameters lower than the original $\mathcal{A}$ ($n_{\mathcal{A}} = 4 < 5 = n_{\mathcal{A}'}$ and

$n_{\mathcal{A}}^{\max} = 2 < 3 = n_{\mathcal{A}'}^{\max}$).

$$x_0x_1 + x_3 + 1$$

$$x_1x_2 + x_2 + x_3$$

Therefore, if we have the overdefined system $\mathcal{A}$ of nonlinear equations we can try to combine the equations and reduce both parameters.

We use systematic approach using Gaussian elimination instead of looking for the good equations for combining "by hand". We linearize the system $\mathcal{A}$ (same as it is done for linearization, each monomial is substituted by a fresh variable) and then eliminate the linearized system $\mathcal{A}$. We get the $\mathcal{A}'$ by backward substitution from the eliminated system. We preprocess the system $\mathcal{A}$ by Gaussian elimination, therefore preprocessing will run in polynomial time ($O(n^3)$). The new system $\mathcal{A}'$, may have parameters $n_{\mathcal{A}'}$ and $n_{\mathcal{A}'}^{\max}$ lower than the original $\mathcal{A}$. Let T be the number of variables in the linearized system and r be a rank of the matrix M representing the linearized system $\mathcal{A}$. After Gaussian elimination, the matrix M will look as (5.1):

$$\begin{pmatrix} 1 & 0 & 0 & \dots & 0 & a_{1,r+1} & \dots & a_{1,T} \\ 0 & 1 & 0 & \dots & 0 & a_{2,r+1} & \dots & a_{2,T} \\ \vdots & \vdots & \vdots & & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \dots & 1 & a_{r,r+1} & \dots & a_{r,T} \\ 0 & 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \dots & 0 & 0 & & 0 \end{pmatrix} \tag{5.1}$$

where $a_{i,j} \in \mathbb{F}_2$. We can easily write the formulas for the parameters of the new system $\mathcal{A}'$.

$$n_{\mathcal{A}'} = \sum_{1 \le j \le r, r+1 \le i \le T} a_{i,j}, \qquad n_{\mathcal{A}'}^{\max} = \max_{1 \le j \le r} \Big(\sum_{r+1 \le i \le T} a_{i,j} \Big).$$

We can see that the success of the proposed method depends on the rank r of the matrix M. We do not have any influence on the values of $a_{i,j}$ and we also do not have any influence on the parameter T. We can only try

to increase the rank of the matrix M by adding the new equations into the system $\mathcal{A}$ (it means to know more keystream bits in our case). Adding the new equations into $\mathcal{A}$ has a negative impact on the time complexity of the conversion algorithm, because we increase the number of equations which has to be converted into CNF. However we do not have to work with the whole system $\mathcal{A}'$. We can choose just a subset of $\mathcal{A}'$ (e.g. first j equations, which contain all unknown variables, but j has to be equal or higher than the number of the unknown variables) and convert only this subset of equations into CNF.

Example 5.2. *Let $\mathcal{A}$ be a system of nonlinear equations.*

$$\begin{aligned}\mathcal{A} = \{&x_0x_1 + x_0x_3 + x_1x_2 + x_2x_3 + 1,\\ &x_2x_3 + x_0,\ x_0x_1 + x_1x_3,\ x_1x_2 + x_2x_3 + 1\}.\end{aligned}$$

We have:

$$n_{\mathcal{A}} = 8, \quad n_{\mathcal{A}}^{\max} = 4.$$

We linearize the system $\mathcal{A}$

$$M = \left(\begin{array}{cccccccccc} x_0x_1 & x_1x_2 & x_0x_3 & x_2x_3 & x_1x_3 & x_0 & x_1 & x_2 & x_3 & \\ \hline 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right)$$

and then use Gaussian elimination.

$$M \sim \left(\begin{array}{cccccccccc} x_0x_1 & x_1x_2 & x_0x_3 & x_2x_3 & x_1x_3 & x_0 & x_1 & x_2 & x_3 & \\ \hline 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \end{array}\right)$$

Hence we get the new system $\mathcal{A}'$.

$$\mathcal{A}' = \{x_0x_1 + x_1x_3 = 0, x_1x_2 + x_0 = 1, x_0x_3 + x_1x_3 = 0, x_2x_3 + x_0 = 0\}.$$

The proposed heuristic reduced both paremeters by factor 2.

$$n_{\mathcal{A}'} = 4, \quad n_{\mathcal{A}'}^{\max} = 2.$$

We also run our conversion program poly2cnf (see Appendix A) on both systems $\mathcal{A}$ and $\mathcal{A}'$. The running time of conversion for $\mathcal{A}'$ was roughly half of the time for the original system $\mathcal{A}$.

We call the density $0 < p < 1$ of the system of equations as a probability, that the coefficient of any monomial in equation is not equal to 0. The selected experimental results of this simple technique for reducing the time complexity of the conversion algorithm from ANF to CNF are presented in Appendix C. Examining Figure 5.1 we can see that presented technique may reduce the size of the parameters $n_{\mathcal{A}}$ and $n_{\mathcal{A}}^{\max}$ by factor 3 to 5 for the strongly overdefined dense systems ($p > \frac{1}{4}$, for more information see also tables C.1, C.2 and C.3). The technique does not reduce the parameters $n_{\mathcal{A}}$ and $n_{\mathcal{A}}^{\max}$ for the sparse systems ($p < \frac{1}{10}$, for detailed information see tables C.4, C.5 and C.6). For example for the density $p = \frac{1}{12}$, the heuristic reduces the parameters only for the strongly overdefined systems and the factor is between 1 and 1.5. The results of our experiments show that both parameters, $n_{\mathcal{A}}$ and $n_{\mathcal{A}}^{\max}$, are reduced approximately by the same factor as well. After various experiments it seems that the presented technique is highly efficient for the systems with high density.

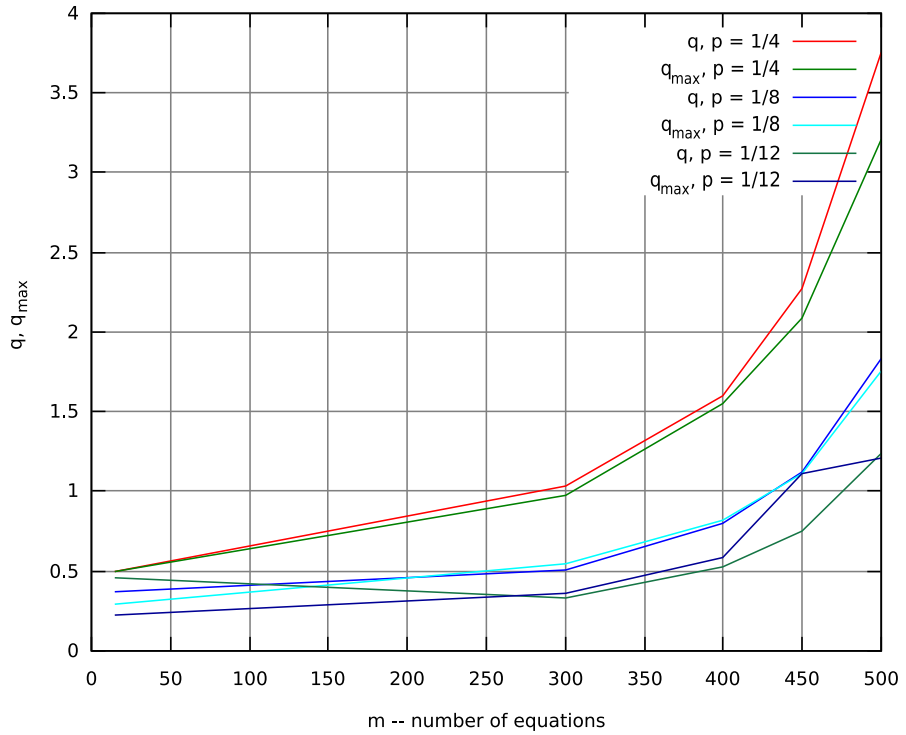

Figure 5.1: Graphs for the tables C.4, C.5 and C.6 from Appendix C. $q = \frac{n_{\mathcal{A}}}{n_{\mathcal{A}'}}$, $q = \frac{n_{\mathcal{A}}^{\max}}{n_{\mathcal{A}'}^{\max}}$

Chapter 6

Fast Algebraic Attack

We have concentrated on the second step of the algebraic attack – solving the system of equations in the previous chapters. Theorem 2.1 shows that solving the quadratic system over finite field is NP-Complete. We have also seen that the maximal degree of the system has impact on the time complexity of solving the system. For example, for linearization method the higher degree of the equations means the higher number of the possible monomials, which has impact on the number of the new variables. It has also impact on the number of required known keystream bits for successful attack. Hence if we know how to find a system with a lower degree, the algebraic attack can be accelerated. We present method known as Fast Algebraic Attack (FAA) in this chapter.

FAA was introduced by Courtois [11] at Crypto 2003. The idea of finding the new system is based on the principle that the equations of original system are linearly combined to get new equations. The new equations have lower maximal degree. So FAA contains the following three steps, where the first and the third one are the same as the standard algebraic attack has.

1. Find a system of equations.
2. Reduce the overall degree of equations.
3. Solve the system of equations.

Unfortunately, Courtois did not prove the correctness of the second step of FAA. Later, Armknecht has proved that the second step is applicable for cryptographically reasonable LFSR-based stream ciphers in [6].

Let $\mathcal{C} = \{c_i\}_{i=0}^{\infty}$ be the keystream of the attacked stream cipher, let L be the next state function of the cipher and let $\mathcal{K}$ be the key used during the encryption. Generally, the attacker has to find a Boolean function $F \neq 0$ such that for $\delta \geq 0$ the equation

$$F(L^t(\mathcal{K}), \ldots, L^{t+\delta}(\mathcal{K}), c_t, \ldots, c_{t+\delta}) = 0. \tag{6.1}$$

is true for all clocks t. Then he can easily find a system $\mathcal{A}$ of equations using F according to the attack scenario (Section 2.1). The degree of the system is concerning the variables representing the key $\mathcal{K}$, because the variables c_i are substituted by captured bits of the keystream during the attack.

Note 6.1. *We denote by F_t the Boolean function F for some fixed t and $\mathcal{C}_{t,\delta}$ represents the part of the keystream $\mathcal{C}$, $\mathcal{C}_{t,\delta} = (c_t, c_{t+1}, \ldots, c_{t+\delta})$.*

The function F is easily derivable from combining or filtering function in case of generating equations for a stream cipher from NLCG or NLFG class as it was described in Chapter 2 (for simplicity, we did not use this notation in Chapter 2). The δ is equal to 0 in this case.

Example 6.1. *Let $f(l_0, l_1)$ be a combining Boolean function for NLCG with two LFSRs with total length n ($l_i^{(t)}$ is equal to the t^{th} output of the i^{th} LFSR). Then $f(l_0, l_1)$ can be viewed as the Boolean function* (6.1) *required for the algebraic attack where $\delta = 0$:*

$$F(L^t(\mathcal{K}), c_t) = f(l_0^{(t)}, l_1^{(t)}) - c_t = 0.$$

The arguments $l_0^{(t)}, l_1^{(t)}$ contain only linear combinations of the variables x_i, $0 \leq i \leq n-1$, which represent the key $\mathcal{K}$ used during the encryption.

Suppose that (6.1) can be written as well as addition of the two Boolean

functions $G(\mathcal{K})$ and $H(\mathcal{K}, \mathcal{C})$

$$\begin{aligned} F_t(\mathcal{K}, \mathcal{C}_{t,\delta}) &= G_t(\mathcal{K}) + H_t(\mathcal{K}, \mathcal{C}_{t,\delta}) = \\ &= G(L^t(\mathcal{K}), \ldots, L^{t+\delta}(\mathcal{K})) + \\ &+ H(L^t(\mathcal{K}), \ldots, L^{t+\delta}(\mathcal{K}), c_t, \ldots, c_{t+\delta}) = 0, \end{aligned} \tag{6.2}$$

where $\deg(H) < \deg(F)$. We also assume that the attacker knows coefficients $a_0, a_1, \ldots a_{T-1} \in \mathbb{F}_2$ such that

$$\sum_{i=0}^{T-1} a_i . G_{t+i}(\mathcal{K}) = 0 \quad \forall \mathcal{K}, \forall t. \tag{6.3}$$

Then the attacker can get a new equation $E'(\mathcal{K}, \mathcal{C})$ with $\deg(E') < \deg(F)$ by linear combination of the T consecutive equations.

$$\begin{aligned} 0 = E'(\mathcal{K}, \mathcal{C}) &= \sum_{i=0}^{T-1} a_i \cdot E_{t+i}(\mathcal{K}, \mathcal{C}) \\ &= \sum_{i=0}^{T-1} a_i \cdot F_{t+i}(\mathcal{K}, \mathcal{C}_{t+i,\delta}) \\ &= \sum_{i=0}^{T-1} (a_i \cdot G_{t+i}(\mathcal{K}) + a_i \cdot H_{t+i}(\mathcal{K}, \mathcal{C}_{t+i,\delta})) \\ &= 0 + \sum_{i=0}^{T-1} a_i \cdot H_{t+i}(\mathcal{K}, \mathcal{C}_{t+i,\delta}) \\ &= \sum_{i=0}^{T-1} a_i \cdot H_{t+i}(\mathcal{K}, \mathcal{C}_{t+i,\delta}). \end{aligned}$$

The degree of the new equation $E'(\mathcal{K}, \mathcal{C})$ is lower than the $\deg(F)$. It is upper bounded by $\deg(H)$. The attacker can reduce the maximal degree of the system $\mathcal{A}$ using this technique. It needs to know more keystream bits c_i than the standard algebraic attack and also the attacker needs to know T consecutive bits for each new equation. Generally, the standard algebraic attack does not require the consecutive keystream bits.

Example 6.2. *Let us have NLCG with two LFSRs and let* $f(l_0, l_1) = l_0 l_1 + l_0 + l_1$ *be a combining Boolean function. Let* L_0 *and* L_1 *be matrices representing the feedback functions of LFSRs.*

$$L_1 = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}, \quad L_2 = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}.$$

The Boolean function $F_t(\mathcal{K}, \mathcal{C}_{t,0})$ *is equal to*

$$F_t(\mathcal{K}, \mathcal{C}_{t,0}) = f(l_0^{(t)}, l_1^{(t)}) - c_t = l_0^{(t)} l_1^{(t)} + l_0^{(t)} + l_1^{(t)} - c_t = 0.$$

Let $(c_0, \ldots, c_7)$ *be the first 8 keystream bits of the generator. Then the first 8 generated equations for our stream cipher using the technique described in Chapter 2 are:*

$$E_0 : x_1x_3 + x_1 + x_3 - c_0$$
$$E_1 : x_0x_4 + x_1x_4 + x_0 + x_1 + x_4 - c_1$$
$$E_2 : x_0x_2 + x_0x_3 + x_0 + x_2 + x_3 - c_2$$
$$E_3 : x_1x_3 + x_1x_4 + x_1 + x_3 + x_4 - c_3$$
$$E_4 : x_0x_2 + x_1x_2 + x_0x_3 + x_1x_3 + x_0x_4 + x_1x_4 + x_0 + x_1 + x_2 + x_3 + x_4 - c_4$$
$$E_5 : x_0x_2 + x_0x_4 + x_0 + x_2 + x_4 - c_5$$
$$E_6 : x_1x_2 + x_1 + x_2 - c_6$$
$$E_7 : x_0x_3 + x_1x_3 + x_0 + x_1 + x_3 - c_7$$

We can rewrite the Boolean function $F_t(\mathcal{K}, \mathcal{C}_{t,0})$ *as an addition of the two Boolean functions* $G(l_0^{(t)}, l_1^{(t)}) = l_0^{(t)} l_1^{(t)}$ *and* $H(l_0^{(t)}, l_1^{(t)}, c_t) = l_0^{(t)} + l_1^{(t)} - c_t$.

$$F_t(\mathcal{K}, \mathcal{C}_{t,0}) = G(l_0^{(t)}, l_1^{(t)}) + H(l_0^{(t)}, l_1^{(t)}, c_t) = l_0^{(t)} l_1^{(t)} + l_0^{(t)} + l_1^{(t)} - c_t.$$

Let $(a_0, a_1, a_2, a_3, a_4, a_5, a_6) = (1, 1, 1, 0, 1, 0, 1)$ *be a sequence of the coefficients required for FAA. We are able to eliminate the quadratic monomials*

according to (6.3) *using these coefficients.*

$$\sum_{i=0}^{6} a_i \cdot G_i(l_0^{(i)}, l_1^{(i)}) = x_1x_3 + x_0x_4 + x_1x_4 + x_0x_2 + x_0x_3 + x_0x_2 + \\ + x_1x_2 + x_0x_3 + x_1x_3 + x_0x_4 + x_1x_4 + x_1x_2 = 0;$$

$$\sum_{i=0}^{6} a_i \cdot G_{i+1}(l_0^{(i+1)}, l_1^{(i+1)}) = x_0x_4 + x_1x_4 + x_0x_2 + x_0x_3 + x_1x_3 + x_1x_4 + \\ + x_0x_2 + x_0x_4 + x_0x_3 + x_1x_3 = 0.$$

Therefore we are able to get the new system of equations. The new system contains only linear equations. For demonstration, the first two new equations belonging to the new system are:

$$E_0' := \sum_{i=0}^{6} a_i \cdot (G_i(l_0^{(i)}, l_1^{(i)}) + H_i(l_0^{(i)}, l_1^{(i)}, c_i)) = \\ = x_0 + x_2 + x_3 - c_0 - c_1 - c_2 - c_4 - c_6 = 0;$$

$$E_1' := \sum_{i=0}^{6} a_{i+1} \cdot (G_{i+1}(l_0^{(i+1)}, l_1^{(i+1)}) + H_{i+1}(l_0^{(i+1)}, l_1^{(i+1)}, c_{i+1})) = \\ = x_1 + x_3 + x_4 - c_1 - c_2 - c_3 - c_5 - c_7 = 0.$$

6.1 How to Find Coefficients $a_0, \ldots, a_{T-1}$

Let us start with formal definition of a linear recurring sequence.

Theorem 6.1. *(Lidl, Niederreiter [14]) A sequence $\mathcal{Z} = (z_t)_{t=0}^{\infty}$ over $\mathbb{F}_2$ is called a linear recurring sequence if coefficients $a_0, \ldots, a_{T-1} \in \mathbb{F}_2$ (not all zero) exist such that $\sum_{i=0}^{T-1} a_i z_{t+i} = 0$ is true for all values $t \geq 1$. In this case, $\sum_{i=0}^{T-1} a_i x^i \in \mathbb{F}_2[x]$ is called a characteristic polynomial of the sequence $\mathcal{Z}$. Amongst all characteristic polynomials of $\mathcal{Z}$ exists one unique polynomial $\min(\mathcal{Z})$ which has the lowest degree. We will call it the minimal polynomial of $\mathcal{Z}$. A polynomial $f(x) \in \mathbb{F}_2[x]$ is a characteristic polynomial of $\mathcal{Z}$ if and only if $\min(\mathcal{Z})$ divides $f(x)$.*

Note 6.2. *The connection polynomial of LFSR defined in Definition 2.2 is a*

characteristic polynomial for a linear recurring sequence generated by LFSR.

Definition 6.1. *Let $\overline{\mathbb{F}}$ be the smallest field such that $\mathbb{F} \subset \overline{\mathbb{F}}$ and each polynomial $f(x) \in \mathbb{F}[x]$ has at least one root in $\overline{\mathbb{F}}$. We will call the field $\overline{\mathbb{F}}$ an algebraic closure of the field $\mathbb{F}$.*

It is crucial to find the coefficients $a_0, \ldots, a_{T-1} \in \mathbb{F}_2$ efficiently for the success of FAA. Courtois proposed Algorithm 6.1 in [11].

Algorithm 4 Finding coefficients a_i

1: Choose a reasonable key $\mathcal{K}'$ and compute c_i for $0 \leq i \leq 2T$
2: Apply the Berlekamp-Massey algorithm to find $a_0, \ldots, a_{T-1}$ where

$$\sum_{i=0}^{T-1} a_i \cdot F_{t+i}(\mathcal{K}') = 0 \quad \forall t. \tag{6.4}$$

It is well known that the Berlekamp-Massey algorithm finds coefficients $a_0, \ldots, a_{T-1}$ efficiently with the smallest value of T and the coefficients fulfill (6.4) (for more information see [15]). Algorithm 6.1 needs $O(T^2)$ steps. The exact value of T is unknown, but T is upper bounded by the number of possible different monomials occurring in the generated equations. The reasonable key $\mathcal{K}'$ is key from $\{0,1\}^n$ where each LFSR is not filled with zeros during initialization. If one of the LFSRs is filled with zero during the initialization, the described algorithm usually returns the wrong result. Hence it is recommended to choose the key, where all LFSRs are not in all-zero state. The correctness of Algorithm 6.1 was not proved in [11]. The problem is that the Algorithm 6.1 returns coefficients for the concrete reasonable choice of the key $\mathcal{K}'$. We need coefficients $a_0, a_1, \ldots a_{T-1}$, which are good for any choice of the key $\mathcal{K}$, for the successful FAA. There is no obvious reason that (6.3) implies (6.4). As the following example shows, this implication does not hold for all cases. In other words, Algorithm 6.1 does not always work properly.

Example 6.3. *Let us have NLCG with two LFSRs L_1, L_2, let $\mathcal{Z}_1 = (z_t^{(1)})_{t=0}^{\infty}$ and $\mathcal{Z}_2 = (z_t^{(2)})_{t=0}^{\infty}$ be the output sequences of LFSRs. Let $x^2 + x + 1$*

be the characteristic polynomial of $\mathcal{Z}_1$ *and* $x^4 + x + 1$ *be the characterstic polynomial of* $\mathcal{Z}_2$*. Let* $f(l_0, l_1) = l_0 l_1 + l_0 + l_1$ *be a combining Boolean function. Firstly, we rewrite the combining function* $f(l_0, l_1)$ *as an addition of the two Boolean functions* $G(l_0, l_1) = l_0 l_1$ *and* $H(l_0, l_1) = l_0 + l_1$*,* $f(l_0, l_1) = G(l_0, l_1) + H(l_0, l_1)$*. Let us try to use Algorithm 6.1 on our cipher. If Algorithm 6.1 is correct, for any choice of reasonable key, the coefficients calculated by algorithm should be the same (coefficients represents* $\min(\mathcal{Z})$*, where* $\mathcal{Z}$ *is an output sequence of the stream cipher using* $G(l_0, l_1)$ *as the combining function). Firstly, we choose the following initial states of LFSRs:* $\mathcal{K}_1 = (1, 0)$*,* $\mathcal{K}_2 = (1, 1, 1, 1)$*. We get the coefficients* $(1, 0, 0, 0, 0, 0, 1)$*, respectively* $\min(\mathcal{Z}) = 1 + x^6$*. Secondly, we choose the different initial states of LFSRs:* $\mathcal{K}_1' = (0, 1)$*,* $\mathcal{K}_2' = (1, 1, 1, 1)$*. Unfortunately, we get the different coefficients* $(1, 1, 1, 1, 1, 1)$*,* $\min(\mathcal{Z}) = 1 + x + x^2 + x^3 + x^4 + x^5$ *respectively. Hence Algorithm 6.1 does not work properly in this case.*

It is clear that the Algorithm 6.1 does not work for every NLCG from the previous example. Armknecht has formulated and proved the following theorem in [6], which describes a subclass of the ciphers, where Algorithm 6.1 works properly.

Definition 6.2. *(Armknecht [6]) Let* $R_1, \ldots, R_k \subseteq \overline{\mathbb{F}_2}$ *be pairwise disjunct,* $R := R_1 \cup \ldots \cup R_k$*. We say that a pair of vectors* $(\alpha_1, \ldots, \alpha_n) \in R^n$*,* $(\beta_1, \ldots, \beta_m) \in R^m$ *factorizes uniquely over* $R_1, \ldots, R_k$ *if the following holds*

$$\alpha_1 \ldots \alpha_n = \beta_1 \ldots \beta_m \Rightarrow \prod_{\alpha_i \in R_l} \alpha_i \prod_{\beta_j \in R_l} (\beta_i)^{-1} = 1,\ 1 \leq l \leq k.$$

For a monomial $\mu = \prod_{j=1}^{k} x_{i_j} \in \mathbb{F}_2[x_1, \ldots, x_n]$ *with* $\{i_1, \ldots, i_k\} \subseteq 1, \ldots, n$ *and* $\alpha = (\alpha_1, \ldots, \alpha_n) \in R^n$*, we define vector* $\overrightarrow{\mu}(\alpha) := (\alpha_{i_1}, \ldots, \alpha_{i_k}) \in R^k$*.*

Theorem 6.2. *(Armknecht [6]) Let* $\mathcal{Z}_1 = (z_t^{(1)})_{t=0}^{\infty}, \ldots, \mathcal{Z}_k = (z_t^{(k)})_{t=0}^{\infty}$ *be sequences with pairwise co-prime minimal polynomials which have only non-zero roots. Let* R_i *denote the set of roots of* $\min(\mathcal{Z}_i)$ *in* $\overline{\mathbb{F}_2}$*,* $F : {\mathbb{F}_2}^n \to \mathbb{F}_2$ *be an arbitrary Boolean function and* $I := (i_1, \ldots, i_n) \in \{1, \ldots, k\}^n$ *and* $\delta := (\delta_1, \ldots \delta_k) \in \mathbb{N}^k$ *be two vectors. We set* $R := R_{i_1} \times \ldots \times R_{i_n}$ *and*

divide $F = \sum \mu$ *into sum of monomials. Further on, for arbitrary* $d := (d_1, \ldots, d_k) \in \mathbb{N}^k$ *the sequences* $\mathcal{Z} := (z_t)_{t=0}^{\infty}$ *and* $\mathcal{Z}^{(d)} := (z_t^{(d)})_{t=0}^{\infty}$ *are defined by*

$$z_t := F(z_{t+\delta_1}^{(i_1)}, \ldots, z_{t+\delta_n}^{(i_n)}), \quad z_t^{(d)} := F(z_{t+\delta_1+d_{i_1}}^{(i_1)}, \ldots, z_{t+\delta_n+d_{i_n}}^{(i_n)}).$$

If all pairs of vectors $\overrightarrow{\mu_i(\alpha)}, \overrightarrow{\mu_j(\alpha')}$ *with* $\alpha, \alpha' \in R$ *factorize uniquely over* $R_1, \ldots, R_k$*, the* $\min(\mathcal{Z}) = \min(\mathcal{Z}^{(d)})$.

What is the connection between Theorem 6.2 and the LFSR-based stream ciphers? The output sequence of the NLCG or NLFG is a linear recurring sequence $\mathcal{Z}$. So Algorithm 6.1 looks for the coefficients $a_0, \ldots, a_{T-1}$ which correspond to the minimal polynomial of $\mathcal{Z}$ for some reasonable key $\mathcal{K}$. This minimal polynomial is unique according to Theorem 6.1. Theorem 6.2 shows that if we shift each of the sequences produced by LFSRs individually (shift of LFSRs is defined in vector d), the minimal polynomial of $\mathcal{Z}$ remains the same. Because the period of the generated sequence is maximal, the minimal polynomial is the same for any choice of the key $\mathcal{K}$.

How large is the class of the LFSR-based stream ciphers which corresponds to Theorem 6.2? The following two theorems describe the large subclasses belonging to the class of ciphers vulnerable to FAA.

Theorem 6.3. *Algorithm 6.1 works properly for any NLFG, where the characteristic polynomial of NLFG has only non-zero roots.*

Proof. Let $R' = \bigcup_{i=1}^{k} R_i$ and let $\mathcal{P} = \{\alpha_1 \ldots . \alpha_n | \alpha_i \in R', n \in \mathbb{N}\}$ be the set of all possible multiple products of elements in R'. We show that for all pairs of vectors $\overrightarrow{\alpha} = (\alpha_1, \ldots, \alpha_n)$, $\overrightarrow{\beta} = (\beta_1, \ldots, \beta_m)$ such that $\alpha_1 \ldots \alpha_n$, $\beta_1 \ldots \beta_m$ are elements in $\mathcal{P}$ factorize uniquely over $R_1, \ldots, R_k$. We have only one LFSR in case of NLFG.

$$\alpha_1 \ldots \alpha_n = \beta_1 \ldots \beta_k \iff \prod_{i=1}^{n} \alpha_i \prod_{j=1}^{m} (\beta_j)^{-1} = 1 \iff \prod_{\alpha_i \in R_1} \alpha_i \prod_{\beta_j \in R_1} (\beta_j)^{-1} = 1.$$

We showed that any two vectors factorize uniquely over R_1 for the set $\mathcal{P}$, which is the superset of all possible vectors $\mu(\alpha)$, therefore the conditions of

Theorem 6.2 are satisfied. □

Theorem 6.4. *Algorithm 6.1 works properly for any NLCG, where the degrees of the minimal polynomials of LFSRs are pairwise co-prime and characteristic polynomials have only non-zero roots.*

Proof. Let d_i denote the degree of the minimal polynomial $m_i(x)$. Then $\alpha, \alpha^{-1}, \alpha \in R_1$ and all multiple products are elements of $\mathbb{F}_{2^{d_i}}$. Let us choose l with $1 \leq l \leq k$ and $S_l = d_1 \dots d_{l-1} d_{l+1} \dots d_k$. Then

$$\alpha_1 \dots \alpha_n = \beta_1 \dots \beta_k \Rightarrow \gamma_l = \prod_{\alpha_i \in R_1} \alpha_i \prod_{\beta_j \in R_1} (\beta_j)^{-1} = \prod_{\alpha_i \notin R_1} \alpha_i \prod_{\beta_j \notin R_1} (\beta_j)^{-1}.$$

Therefore γ_l belongs to $\mathbb{F}_{2^{d_i}} \cap \mathbb{F}_{2^{S_i}}$. We have to show, that $\gamma_l = 1$. The degrees of the minimal polynomials are pairwise co-prime by assumption. Therefore $\gcd(d_i, S_i) = 1$. Hence $\gamma_l \in \mathbb{F}_2$. The roots are non-zero, so γ_l has to be equal to 1. We showed that for any NLCG, where the degrees of minimal polynomials are pairwise co-prime, the conditions of Theorem 6.2 are satisfied. □

Chapter 7

Conclusion

The main purpose of this thesis was to present and explain basic ideas of the very recent attack on the stream ciphers - algebraic attack. We have dedicated careful attention to both main steps of algebraic attack: finding a system of equations and solving this system. We have described methods of generating equations for two large classes of the stream ciphers based on LFSRs. We have also presented method of generating equations for the stream cipher A5/1, which is irregularly clocked. Unfortunately, our method is not efficient. On the other side, our idea may be helpful in applying algebraic attack on the other stream ciphers based on irregular clocking. We have studied two approaches for solving the systems of nonlinear equations over finite field. We have presented two algorithms, linearization and XL algorithm, which represent standard approach based on linear algebra. We have run various experiments with our implementation of XL algorithm. The second approach of solving these systems was based on converting problem to the SAT problem. The key part of this approach is how to effectively convert Boolean expression in ANF into CNF. We have presented and experimentally analyzed our heuristic for speeding up this process. We concluded that defined technique works very well for the systems with high density. In this work we have presented improved algebraic attack – fast algebraic attack as well.

The algebraic attack is related to various problems, from effective al-

gorithms of solving the large systems of linear equations to the theory of Boolean function. Due to this fact, there are many directions in which this thesis can be extended. For example, there are more sophisticated methods for solving the system of nonlinear equations at the moment. It may be interesting to study impact of the algebraic attack on the design criteria of Boolean functions used in various cryptosystems. Also studying possibility of applying developed methods used for attacking stream ciphers on other cryptographic primitives such as block ciphers or hash functions could be worth to study.

Appendix A

Programs Descriptions

A.1 Program cipher-gen

Program cipher-gen generates a system of equations for attacking a stream cipher belonging to the classes NLFG or NLCG as it was described in Chapter 2. The program is implemented in C++ programming language. Library Synaps [4] was used for the finite field arithmetic, the linear algebra and for representing multivariate polynomials. The stream cipher has to be defined in a configuration file. The configuration file has a simple format. For each LFSR the file contains two lines. First line contains the length of the LFSR and the initial state of the LFSR (the initial state is not required). The second line corresponding to the LFSR contains a matrix L_f written by rows. The matrix L_f represents a feedback function of the LFSR.

Example A.1. *Example of the configuration file for the program cipher-gen for NLCG described in Example 2.1.*

```
2 0 1
0 1 1 1
3 1 1 1
0 0 1 1 0 1 0 1 0

SYNOPSIS:
$ cipher-gen [OPTIONS]
```

Switch	Description	Default
-c	combining or filtering function of the cipher (required)	–
-d	maximal degree of equations	2
-h	print help	–
-i	configuration file of the stream cipher	–
-l	switch on logic output format, input format for the program poly2cnf	off
-o	name of the output file (required)	–
-r	range of the generated equations ('1,3-7,9,11-13')	1
-v	verbose mode	off
-V	print program's version	-

A.2 Program eq-gen

Program eq-gen generates a random system of multivariate polynomial equations over finite field $\mathbb{F}$. The program is implemented in C programming language.

```
SYNOPSIS:
$ eq-gen [OPTIONS]
```

Switch	Description	Default
-h	print help	–
-d	maximal degree of equations	2
-f	finite field	$\mathbb{F}_2$
-l	switch on logic output format, input format for the program poly2cnf	off
-n	number of generated equations	0
-o	name of the output file	stdout

Switch	Description	Default
-p	density of the generated system	1/4
-s	generated system will have at least one solution	off
-v	set the number of variables	6
-V	print program's version	-

A.3 Program elim

Program elim performs Gaussian elimination. It is implemented in C++ using library Synaps [4] for representing multivariate polynomials over $\mathbb{F}_2$.

```
SYNOPSIS:
$ elim [OPTIONS] [INPUT] [OUTPUT]
```

Switch	Description	Default
-h	print help	–
-H	size of the hash table used for linearization	127
-v	verbose mode	off
-V	print program's version	–

A.4 Program poly2cnf

Program poly2cnf is an implementation of the Algorithm 3 – Converting ANF into CNF. The main algorithm is implemented in Prolog. The input file has to be in logic input format. Each equation has a dot at the end and variables are represented by numbers starting with 1. The output of the algorithm is in the DIMACS format. The DIMACS format is widely accepted format for representing Boolean expressions in CNF.

Example A.2. *Example of the DIMACS format for the Boolean expression* $B = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_2)$.

```
c Boolean expression B
p cnf 3 2
```

```
1 2 3 0
-1 2 0
```

The first line is a comment and it will be ignored by SAT solver. The second line defines the number of variables and the number of clauses in B. Every following line specifies one clause of the Boolean expression B. A positive literal is represented by the corresponding number, a negative literal is represented by the corresponding number with minus sign. The last number in the line representing the clause should be zero.

```
SYNOPSIS:
$ poly2cnf [INPUT] [OUTPUT]
```

A.5 Program xl

Program xl is an implementation of the XL algorithm described in Chapter 4. It is implemented in C++ programming language. We used Synaps library [4] for representing multivariate polynomials over $\mathbb{F}_2$ and for linear algebra.

```
SYNOPSIS:
$ xl [OPTIONS] [INPUT]
```

Switch	**Description**	**Default**
-D	parameter D	4
-h	print help	–
-H	size of the hash table used for linearization	127
-m	X^k used in generation of the new equations a: $k = 1 \dots D - d_{\max}$ e: $1 \leq k \leq D - d_{\max}$ and k is even o: $1 \leq k \leq D - d_{\max}$ and k is odd l: $k = D - d_{\max}$	a
-r	maximum number of the iterations	unlimited
-s	print statistics	off
-v	verbose mode	off
-V	print program's version	–

Appendix B

Results for the Simulations of the XL Algorithm

In the following simulations, we generate a random system of polynomial equations with at least one solution satisfying defined parameters using program eq-gen. Then we use program xl for solving the system.

Notation:

$d_{\max}$ – maximal degree of equations
n – number of variables
m – number of equations
D – parameter D
R – number of generated equations
T – number of monomials of degree $\leq D$
R_{Free} – number of linearly independent equations
R'_{Free} – number of linearly independent equations estimated by Conjecture 4.1

$d_{\max}$	3	3	3	3	3	3	3	3	3	3
n	10	10	10	10	10	15	15	15	15	15
m	10	14	20	35	40	15	25	35	125	140
D	4	4	4	4	4	4	4	4	4	4
R	110	154	220	385	440	240	400	560	2000	2240
T	385	385	385	385	385	1940	1940	1940	1940	1940
R_{Free}	110	154	220	385	385	240	400	560	1940	1940
R'_{Free}	110	154	220	385	385	240	400	560	1940	1940

Table B.1: Table of results for $d_{\max} = 3$ and $D = 4$

$d_{\max}$	3	3	3	3	3	3	3	3	3	3
n	10	10	10	10	10	15	15	15	15	15
m	10	14	20	35	40	15	17	20	40	50
D	5	5	5	5	5	5	5	5	5	5
R	560	784	1120	1960	2240	1815	2057	2420	4840	6050
T	637	637	637	637	637	4943	4943	4943	4943	4943
R_{Free}	560	637	637	637	637	1815	2057	2420	4840	4943
R'_{Free}	560	637	637	637	637	1815	2057	2420	4840	4943

Table B.2: Table of results for $d_{\max} = 3$ and $D = 5$

$d_{\max}$	2	2	2	2	2	2	2	2
n	10	10	10	15	15	15	32	24
m	10	14	18	15	20	30	32	24
D	5	5	5	5	5	5	3	4
R	1760	2464	3168	8640	11520	17280	1056	7224
T	637	637	637	4943	4943	4943	5488	12950
R_{Free}	634-637	637	637	4943	4943	4943	1056	7224
R'_{Free}	637	637	637	4943	4943	4943	1056	7224

Table B.3: Table of results for $d_{\max} = 2$,$D = 5$

Appendix C

Results for Reducing Time Complexity of ANF to CNF Conversion

In the following simulations, we generate a random system of equations with predefined parameters and apply Gaussian elimination on the generated system. We use our programs eq-gen and elim for simulations (see Appendix A).

Notation:

$d_{\max}$ – maximal degree of equations

n – number of variables

m – number of equations

p – probability that a randomly selected monomial has a non-zero coefficient in equation

$n_{\mathcal{A}}$ – number of xor operation in $\mathcal{A}$

$n_{\mathcal{A}}^{\max}$ – maximal number of xor operation in one equation from $\mathcal{A}$

$q = \frac{n_{\mathcal{A}}}{n_{\mathcal{A}'}}$,

$q_{\max} = \frac{n_{\mathcal{A}}^{\max}}{n_{\mathcal{A}'}^{\max}}$

$d_{\max}$	2	2	2	2	2
p	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$
n	10	10	10	10	10
m	10	20	30	40	50
q	0.55 - 0.81	0.69 - 0.82	0.97 - 1.1	1.64 - 1.91	5.04 - 5.83
$q_{\max}$	0.54 - 0.92	0.8 - 0.95	1.05 - 1.29	1.33 - 1.83	3.16 - 5

Table C.1: Table of results for $d_{\max} = 2$ and $n = 10$.

$d_{\max}$	3	3	3	3	3
p	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$
n	10	10	10	10	10
m	10	30	100	120	150
q	0.54 - 0.69	0.58 - 0.63	0.97 - 1.1	1.5 - 1.59	3.36 - 3.52
$q_{\max}$	0.48 - 0.62	0.61 - 0.71	1.05 - 1.29	1.44 - 1.8	2.5 - 3.39

Table C.2: Table of results for $d_{\max} = 3$, $n = 10$ and $p = 0.25$.

$d_{\max}$	3	3	3	3	3
p	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$
n	20	20	20	20	20
m	20	200	800	1200	1600
q	0.5 - 0.53	0.58 - 0.59	1.22 - 1.23	4.46 - 4.49	13.4 - 13.75
$q_{\max}$	0.48 - 0.57	0.61 - 0.62	1.2 - 1.27	4.03 - 4.34	9.82 - 11.14

Table C.3: Table of results for $d_{\max} = 3$ and $n = 20$

$d_{\max}$	3	3	3	3	3
p	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$
n	15	15	15	15	15
m	15	300	400	450	500
q	0.5 - 0.59	1.03 - 1.04	1.6 - 1.64	2.27 - 2.29	3.76 - 3.81
$q_{\max}$	0.5 - 0.6	0.98 - 1.17	1.55 - 1.7	2.09 - 2.38	3.21 - 3.76

Table C.4: Table of results for $d_{\max} = 3$, $n = 15$ and $p = \frac{1}{4}$.

$d_{\max}$	3	3	3	3	3
p	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$
n	15	15	15	15	15
m	15	300	400	450	500
q	0.37 - 0.5	0,51 - 0.52	0.8 - 0.81	1.12 - 1.14	1.83 - 1.9
$q_{\max}$	0.29 - 0.35	0.55 - 0.63	0.82 - 1.04	1.11 - 1.25	1.76 - 2

Table C.5: Table of results for $d_{\max} = 3$, $n = 15$ and $p = \frac{1}{8}$.

$d_{\max}$	3	3	3	3	3
p	$\frac{1}{12}$	$\frac{1}{12}$	$\frac{1}{12}$	$\frac{1}{12}$	$\frac{1}{12}$
n	15	15	15	15	15
m	15	300	400	450	500
q	0.46 - 0.66	0.33 - 0.34	0.53 - 0.54	0.75 - 0.76	1.24 - 1.27
$q_{\max}$	0.22 - 0.28	0.36 - 0.44	0.59 - 0.68	1.11 - 1.35	1.21 - 1.36

Table C.6: Table of results for $d_{\max} = 3$, $n = 15$ and $p = \frac{1}{12}$

Bibliography

[1] eSTREAM - The ECRYPT Stream Cipher Project. http://www.ecrypt.eu.org/stream/index.html.

[2] MiniSAT - SAT solver. http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/MiniSat.html.

[3] SAT-Race 2006. http://fmv.jku.at/sat-race-2006/.

[4] Synaps (SYmbolic Numeric ApplicationS). http://www-sop.inria.fr/galaad/software/synaps/.

[5] Wikipedia - A5/1. http://en.wikipedia.org/wiki/A5/1.

[6] Armknecht, F. Improving fast algebraic attacks. In *FSE* (2004), pp. 65–82.

[7] Bard, G. V. Achieving a log(n) speed up for boolean matrix operations and calculating the complexity of the dense linear algebra step of algebraic stream cipher attacks and of integer factorization methods. Cryptology ePrint Archive, Report 2006/163, 2006.

[8] Biryukov, A., Shamir, A., and Wagner, D. Real time cryptanalysis of A5/1 on a pc. In *FSE* (2000), pp. 1–18.

[9] Courtois, N. Higher order correlation attacks, xl algorithm and cryptanalysis of toyocrypt. In *ICISC* (2002), pp. 182–199.

[10] Courtois, N., Klimov, A., Patarin, J., and Shamir, A. Efficient algorithms for solving overdefined systems of multivariate polynomial

equations. *Lecture Notes in Computer Science : Advances in Cryptology - EUROCRYPT 2000: International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 2000. Proceedings* (2000), 392+.

[11] Courtois, N. T. Fast algebraic attacks on stream ciphers with linear feedback. 176–194.

[12] Ekhdal, P., and Johansson, T. Another attack on A5/1. *IEEE Transactions on Information Theory 49*, 1 (2003), 284–289.

[13] Kipnis, A., and Shamir, A. Cryptanalysis of the HFE public key cryptosystem by relinearization. *Lecture Notes in Computer Science 1666* (1999), 19–30.

[14] Lidl, R., and Niederreiter, H. Finite fields and their applications. In *Handbook of algebra, Vol. 1.* North-Holland, Amsterdam, 1996, pp. 321–363.

[15] Menezes, A. J., Vanstone, S. A., and Oorschot, P. C. V. *Handbook of Applied Cryptography.* CRC Press, Inc., Boca Raton, FL, USA, 1996.

[16] Patarin, J. Cryptanalysis of the Matsumoto and Imai public key scheme of Eurocrypt '88. 248–261.

[17] Shannon, C. E. Communication theory of secrecy sytems. *Bell Systems Tech. J. 28* (1949), 657–715.

[18] Shoup, V. *A Computational Introduction to Number Theory and Algebra.* Cambridge University Press, June 2005.

[19] Vernam, G. S. Cipher printing telegraph systems for secret wire and radio telegraphic communications. *Journal American Institute of Electrical Engineers XLV* (1926), 109–115.

Abstrakt

Algebraický útok patrí medzi nové kryptoanalytické metódy súčastnosti. Jeho hlavnou myšlienkou je hľadanie a následné riešenie systému polynomiálnych rovníc nad konečným poľom. V súčastnosti sú algebraické útoky veľmi úspešné pri aplikácii na vybrané prúdové šifry založené na posuvných registroch s lineárnou spätnou väzbou (LFSR) ako napríklad nelineárne filtrovacie generátory a nelineárne kombinačné generátory. V tejto práci prezentujeme hlavné myšlienky algebraického útoku a ilustrujeme ich na príkladoch. Podrobne popisujeme metódy generovania systému rovníc pre vybrané triedy prúdových šifier založených na LFSR. Taktiež v tejto práci študujeme XL algoritmus na riešenie systému rovníc nad konečným poľom. Experimentálne sme overili vlastnosti tohto algoritmu. Následne prezentujeme iný prístup riešenia týchto systémov, a to prístup založený na využití programu pre riešenie SAT problému. V tejto práci prezentujeme našu heuristiku na zrýchlenie konverzie boolovských výrazov v algebraickej normálnej forme do konjuktívnej normálne formy. Efektivita tejto konverzie je veľmi dôležitá pre úspech prístupu, ktorý je založený na konverzii na SAT problém. Z našich experimentov vyplýva, že navrhovaná heuristika je veľmi efektívna pre husté systémy. Nakoniec prezentujeme vylepšenie štandardného algebraického útoku a to rýchly algebraický útok.

Kľúčové slová: algebraický útok, rýchly algebraický útok, prúdové šifry, posuvný register s lineárnou spätnou väzbou (LFSR), A5/1, XL algoritmus, program pre riešenie SAT problému.