Rose-Hulman Institute of Technology
Rose-Hulman Scholar

Mathematical Sciences Technical Reports (MSTR) Mathematics

5-2011

Algebraic Solutions to Overdefined Systems with Applications to Cryptanalysis

Eric Crockett
Rose-Hulman Institute of Technology

Follow this and additional works at: http://scholar.rose-hulman.edu/math_mstr

Part of the Algebra Commons, Discrete Mathematics and Combinatorics Commons, and the Theory and Algorithms Commons

Recommended Citation
Crockett, Eric, "Algebraic Solutions to Overdefined Systems with Applications to Cryptanalysis" (2011). *Mathematical Sciences Technical Reports (MSTR)*. Paper 10.
http://scholar.rose-hulman.edu/math_mstr/10

This Article is brought to you for free and open access by the Mathematics at Rose-Hulman Scholar. It has been accepted for inclusion in Mathematical Sciences Technical Reports (MSTR) by an authorized administrator of Rose-Hulman Scholar. For more information, please contact bernier@rose-hulman.edu.

Algebraic Solutions to Overdefined Systems with Applications to Cryptanalysis

Eric Crockett

Adviser: Joshua Holden

Mathematical Sciences Technical Report Series
MSTR 11-06

May 2011

Department of Mathematics
Rose-Hulman Institute of Technology
http://www.rose-hulman.edu/math

Fax (812)-877-8333 **Phone (812)-877-8193**

Algebraic Solutions to Overdefined Systems with Applications to Cryptanalysis

Eric Crockett
Advisor: Dr. Joshua Holden
Rose-Hulman Institute of Technology

May 2011

Abstract

Cryptographic algorithms are based on a wide variety of difficult problems in mathematics. One of these problems is finding a solution to a system of multivariate quadratic equations (MQ). A generalization of this problem is to find a solution to a system of higher order non-linear equations. Both of these problems are NP-hard over any field [2, p. 202]. Many cryptosystems such as AES, Serpent, Toyocrypt, and others can be reduced to some form of the MQ problem. In this paper we analyze the relinearization and XL algorithms for solving overdetermined systems of non-linear equations, as well as two variations of the XL algorithm.

We then introduce the Toyocrypt stream cipher and show how it can be reduced to a system of non-linear equations. We analyze an attack introduced by Courtois [7] using a variation of XL. We also implemented the attack for a smaller version of Toyocrypt, and made many interesting observations that have not been explained by our analysis.

1 Introduction

Claude Shannon said in his famous 1949 paper that ciphers should be constructed "in such a way that breaking it is equivalent to...the solution of some problem known to be laborious, [such as] solving a system of simultaneous equations in a large number of unknowns [15]." The problem of finding one solution (but not necessarily every solution) to a system of quadratic multivariate polynomial equations is known as the MQ problem, and is NP-hard over any field [2].

We consider a non-linear system in n variables and m equations. The MQ problem has been studied for the underdefined case and is polynomial in n when sufficiently underdefined [4]. We will show that the problem becomes polynomial in n if it is adequately overdefined as well. Thus the hardest case of MQ is when $m \approx n$.

1.1 A Motivating Example

The idea of an algebraic attack on a cryptosystem can be motivated by a simple example. Suppose Alice wants to send a secret message to Bob. One possible method is for Alice and Bob to use an affine cipher. They agree on a secret key which is composed of two integers α and β with $\gcd(\alpha, 26) = 1$. To send a message to Bob, Alice encrypts her message one letter at a time by converting each letter to a number (for example, $a = 0$ and $z = 25$) and then encrypting each number with the linear function $c(x) = \alpha x + \beta \bmod 26$. Alice can send this encrypted message over an untrusted channel, and for the purposes of this example, we may assume that no one is able to decrypt the message besides Bob who has the secret key. [1]

When Bob receives the encrypted message, he first computes $\gamma = \alpha^{-1} \bmod 26$. He can then decrypt the message one letter at at time using $d(y) = (y - \beta)\gamma$.

As an example, let $\alpha = 3$ and $\beta = 5$. The plaintext letter e corresponds to the integer "4", so the ciphertext would be $c(4) = 3 \cdot 4 + 5 \equiv 17 \bmod 26$. If Alice sends the message "17" to Bob, Bob computes $\gamma = 9$ (since $\alpha\gamma = 3 \cdot 9 = 27 \equiv 1 \bmod 26$). Then $d(17) = (17 - 5) \cdot 9 = 108 \equiv 4 \bmod 26$, which Bob interprets as the letter e.

Now suppose that we have intercepted the message [20, 25, 18, 10, 11, 23, 20] encrypted with this scheme, but with an unknown key. If we also happen to have discovered through fortune or subterfuge that the plaintext corresponding to the ciphertext "18" was "6" and that the plaintext corresponding to the ciphertext "23" was "17" (i.e. $c(6) = 18$ and $c(17) = 23$), then can we figure out the rest of the message?

We do not know the secret key, but we do know

$$\begin{cases} 18 \equiv 6\alpha + \beta \\ 23 \equiv 17\alpha + \beta \end{cases} \quad \bmod 26$$

This can be modeled by the linear system

$$\begin{bmatrix} 6 & 1 \\ 17 & 1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \equiv \begin{bmatrix} 18 \\ 23 \end{bmatrix} \bmod 26$$

We can solve this system using Gaussian elimination to obtain $\alpha = 17$ and $\beta = 20$. $\gamma = \alpha^{-1} \bmod 26 \equiv 23$. Then $d(20) = (20 - 20) \cdot 23 \equiv 0$, so the first letter of the message is "a". Similarly we decrypt the rest of the message to get [a l g e b r a].

[1] The affine cipher has numerous weaknesses, so Alice and Bob should not expect any degree of privacy over an untrusted channel with this cipher.

This is the basic premise of algebraic attacks on cryptosystems. We will model the cryptosystem as a (possibly non-linear) function. By some means we will obtain some input/output pairs for the function which results in a system of equations. We can then try to solve the system and determine the secret key. For linear systems like the example above, this is trivial (given enough equations relative to unknowns) but the problem becomes much more challenging when the function is non-linear.

This kind of attack is called a "known-plaintext" attack. Although it may seem unlikely that an attacker could obtain the plaintext and corresponding ciphertext necessary to launch such an attack, there are many cases where it is feasible. For example the message header may be constant or predictable. In some cases it may be enough to know the language in which the message is written (see Section 5.1).

2 Linear Feedback Shift Registers (LFSR)

2.1 Stream Ciphers

Now we introduce linear feedback shift registers (LFSR) as a basic component of more sophisticated encryption schemes such as stream ciphers. Later we will describe Toyocrypt, a stream cipher that is vulnerable to an algebraic attack based on the same concept as the attack above. Stream ciphers are a type of symmetric key cipher in which plaintext bits are XORed with a pseudo-random bit stream, called the keystream. Stream ciphers are used in applications that require encryption of large quantities of data very quickly, which makes them a good candidate for algebraic attacks where a large amount of known plaintext is required.

2.2 Galois LFSRs

The pseudo-random bit stream used in stream ciphers is often obtained using a LFSR because of their speed when implemented in hardware. There are many varieties of LFSRs; we will consider a Galois LFSR. Let L be a Galois LFSR of length n, and let $C = \sum_{i=0}^{n} c_i x^i$ be polynomial in $GF(2)[x]$, with $deg(C) = n$, i.e. $c_i \in GF(2)$, $c_n = 1$. Alternatively, we may view C as a vector in $GF(2)^n$: $C = [c_{n-1}, ..., c_1, c_0]$, note that the leading coefficient is dropped in the vector notation because it is always 1. C is called the feedback polynomial for L. For most cryptographic purposes and for the purposes of this paper we want C to be a primitive polynomial so that the LFSR achieves its maximum possible period of $2^n - 1$ [13].

The contents of the shift register is known as the state, and at any time $t \geq 0$, the contents can be represented as a vector $L(t) \in GF(2)^n$. Individual bits of the state at time t are denoted by $L_i(t)$, $i = 0, 1, ..., n-1$. C is used to generate the next state of the shift register every time it is clocked. In this notation, we write $L(t) =< L_0(t), L_1(t), ..., L_{n-1}(t) >$. We define the transition function

$$\begin{aligned} CLK : GF(2)^n \times GF(2)^n &\mapsto GF(2)^n \\ (C, L(t)) &\mapsto L(t+1) \end{aligned}$$

where

$$L_0(t+1) = c_0 L_{n-1}(t)$$

$$L_i(t+1) = L_{i-1}(t) + c_i L_{n-1}(t), \quad i = 1, 2, ..., n-1$$

and addition is performed mod 2. Figure 1 shows visually how the next state of a Galois LFSR is computed.

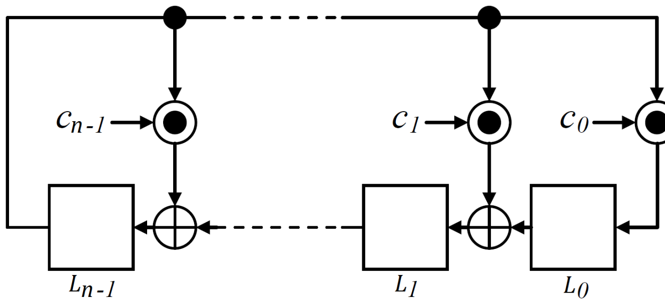

Figure 1: A Galois LFSR, adapted from [17]

The primary weakness of using the output of an LFSR as a pseudo-random bit stream is that given L_{n-1} for $2n$ states, we can easily discover the feedback polynomial and then determine past and future states of the LFSR [18, p. 47]. We describe this attack below because the same idea is used in the attack on Toyocrypt.

2.3 Basic LFSR Attack

Consider a 4-bit Galois LSFR. Suppose we know the values of $L_3(0), L_3(1), ..., L_3(7) \in GF(2)$. Using the recurrence relation given by CLK, we have (known values are in bold):

$$\begin{aligned}
\mathbf{L_3(4)} &= L_2(3) + c_3\mathbf{L_3(3)} \\
&= (L_1(2) + c_2\mathbf{L_3(2)}) + c_3\mathbf{L_3(3)} \\
&= ((L_0(1) + c_1\mathbf{L_3(1)}) + c_2\mathbf{L_3(2)}) + c_3\mathbf{L_3(3)} \\
&= c_0\mathbf{L_3(0)} + c_1\mathbf{L_3(1)} + c_2\mathbf{L_3(2)} + c_3\mathbf{L_3(3)}
\end{aligned}$$

Similarly, we can get algebraic equations for $L_3(5)$, $L_3(6)$, and $L_3(7)$:

$$\begin{aligned}
\mathbf{L_3(5)} &= c_0\mathbf{L_3(1)} + c_1\mathbf{L_3(2)} + c_2\mathbf{L_3(3)} + c_3\mathbf{L_3(4)} \\
\mathbf{L_3(6)} &= c_0\mathbf{L_3(2)} + c_1\mathbf{L_3(3)} + c_2\mathbf{L_3(4)} + c_3\mathbf{L_3(5)} \\
\mathbf{L_3(7)} &= c_0\mathbf{L_3(3)} + c_1\mathbf{L_3(4)} + c_2\mathbf{L_3(5)} + c_3\mathbf{L_3(6)}
\end{aligned}$$

Substituting in the known values, we obtain a linear system of equations which can be easily solved, for example using Gaussian elimination. This allows us to recover the feedback polynomial. As we will show, knowing the feedback polynomial allows us to reconstruct the initial state of the

LFSR, and thus the entire output sequence. It is not necessary for the known output bits to be consecutive as in this example, so this attack applies to more general situations.

We will now show how the initial state can be recovered if we know just n (not necessarily consecutive) output bits and the feedback polynomial. For simplicity, we will use some results from the previous example. Assume that we know $L_3(4)$, $L_3(5)$, $L_3(6)$, and $L_3(7)$ but no other state bits. We can continue to rewrite $L_3(4)$, keeping in mind that we are now assuming knowledge of c_0, c_1, c_2, and c_3 but not $L_3(0)$, $L_3(1)$, $L_3(2)$, or $L_3(3)$.

$$\begin{aligned}\mathbf{L_3(4)} =&\mathbf{c_0}L_3(0) + \mathbf{c_1}L_3(1) + \mathbf{c_2}L_3(2) + \mathbf{c_3}L_3(3)\\ =&\mathbf{c_0}L_3(0) + \mathbf{c_1}(L_2(0) + \mathbf{c_3}L_3(0)) + \mathbf{c_2}(L_2(1) + \mathbf{c_3}L_3(1)) + \mathbf{c_3}(L_2(2) + \mathbf{c_3}L_3(2))\\ =&(\mathbf{c_0} + \mathbf{c_1c_3})L_3(0) + \mathbf{c_1}L_2(0) + \mathbf{c_2}L_2(1) + \mathbf{c_2c_3}L_3(1) + \mathbf{c_3}L_2(2) + \mathbf{c_3^2}L_3(2)\\ =&...\\ =&(\mathbf{c_0} + \mathbf{c_2} + \mathbf{c_3} + \mathbf{c_2c_3})L_3(0) + (\mathbf{c_1} + \mathbf{c_3})L_2(0) + (\mathbf{c_2} + \mathbf{c_3})L_1(0) + \mathbf{c_3}L_0(0)\end{aligned}$$

after taking all coefficients mod 2 and reducing variable exponents to 1 due to the fact that $c_i^2 = c_i$ over $GF(2)$.

Similarly, we can write $L_3(5)$, $L_3(6)$, and $L_3(7)$ in terms of the initial LFSR state:

$$\begin{aligned}\mathbf{L_3(5)} =&(\mathbf{c_3} + \mathbf{c_1c_3} + \mathbf{c_2c_3})L_3(0) + (\mathbf{c_0} + \mathbf{c_2} + \mathbf{c_3} + \mathbf{c_2c_3})L_2(0) + (\mathbf{c_1} + \mathbf{c_3})L_1(0) + \mathbf{c_2} + \mathbf{c_3}L_0(0)\\ \mathbf{L_3(6)} =&(\mathbf{c_1} + \mathbf{c_2} + \mathbf{c_3} + \mathbf{c_0c_3} + \mathbf{c_2c_3})L_3(0) + (\mathbf{c_3} + \mathbf{c_1c_3} + \mathbf{c_2c_3})L_2(0)\\ &+ (\mathbf{c_0} + \mathbf{c_2} + \mathbf{c_3} + \mathbf{c_2c_3})L_1(0) + (\mathbf{c_1} + \mathbf{c_3})L_0(0)\\ \mathbf{L_3(7)} =&(\mathbf{c_3} + \mathbf{c_1c_2})L_2(0) + (\mathbf{c_1} + \mathbf{c_2} + \mathbf{c_3} + \mathbf{c_0c_3} + \mathbf{c_2c_3})L_2(0)\\ &+ (\mathbf{c_3} + \mathbf{c_1c_3} + \mathbf{c_2c_3})L_1(0) + (\mathbf{c_0} + \mathbf{c_2} + \mathbf{c_3} + \mathbf{c_2c_3})L_0(0)\end{aligned}$$

We now have a linear system of four equations and four unknowns, so we can solve to find the initial state of the LFSR. Once we have the initial state of the LFSR, we can generate the entire output sequence by using the known feedback polynomial. Thus for LFSRs if we have no knowledge of the feedback polynomial or the initial state, we can recover both if we know $2n$ bits of the output stream. If we know the feedback polynomial in advance (as we will assume in a later attack), we can recover the initial state of the LFSR if we know just n bits of the output stream. Thus it is not sufficient to use a LFSR for generating pseudo-random bits for cryptographic applications because knowing a few output bits allows us to recover the initial state, even without knowledge of the feedback polynomial.

3 Algebraic Solutions to Non-Linear Systems

We first consider solving a system of non-linear equations algebraically (rather than numerically) over any field. Throughout the rest of this paper, we will refer to a system of equations with m equations in n variables, where n refers to the number of linear variables (i.e. $x_1, x_2, ..., x_n$, not all of which must appear in the system as linear terms) rather than the number of monomials in the system. We say that the system is overdefined (overdetermined) if $m > n$ and underdefined (underdetermined) if $m < n$. We say an equation is homogeneous if every non-constant monomial has the same degree. For example, $x_1x_2+x_3^2 = 5$ is a homogeneous equation while $x_1+x_2x_3+x_3^2 = 0$ is a non-homogeneous equation. We refer to a system as homogeneous if every equation in the system is homogeneous.

3.1 Relinearization

3.1.1 Linearization

For a non-homogeneous quadratic system over a general finite field $GF(q)$, there are $\binom{n}{1} = n$ possible distinct linear terms and $\binom{n+1}{2}$ distinct quadratic terms. [2] Thus the total number of monomials in the system is no more than $n + \binom{n+1}{2}$. We will assume that each of these monomials appear in the system.

The m quadratic equations form a non-homogeneous non-linear system, which cannot directly be solved using Gaussian elimination. One approach to solving a non-linear system is to *linearize* the non-linear monomials. We rewrite the system with new variables $y_i = x_i$ and $y_{ij} = x_i x_j$, $1 \leq i \leq j \leq n$, which does not change the number of monomials, but does convert the quadratic system to a linear system that can be solved by Gaussian elimination. The linearized system has m equations and $n + \binom{n+1}{2}$ unknowns, so we can find a solution (if one exists) when the linearized system has full column rank. The solution to the linearized system contains values for the linearized y_i variables which correspond to the solution to the original system. It is not strictly necessary that every possible monomial appears in the system; for example we could consider a homogeneous quadratic system where every quadratic term appeared. Then the solution to the original x_i variables would be the square roots of the y_{ii} variables. In either case, linearization requires $O(n^2)$ equations, which may not be available in many cases.

For example, consider the non-linear system

$$\begin{bmatrix} 2 & 5 & 9 & 3 & 6 \\ 10 & 7 & 5 & 6 & 10 \\ 9 & 0 & 8 & 1 & 6 \\ 9 & 9 & 10 & 9 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 \\ x_1x_2 \\ x_2^2 \end{bmatrix} \equiv \begin{bmatrix} 7 \\ 3 \\ 1 \\ 8 \end{bmatrix} \bmod 11$$

After linearization, this system becomes

$$\begin{bmatrix} 2 & 5 & 9 & 3 & 6 \\ 10 & 7 & 5 & 6 & 10 \\ 9 & 0 & 8 & 1 & 6 \\ 9 & 9 & 10 & 9 & 2 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_{11} \\ y_{12} \\ y_{22} \end{bmatrix} \equiv \begin{bmatrix} 7 \\ 3 \\ 1 \\ 8 \end{bmatrix} \bmod 11$$

3.1.2 Degree Four Relinearization

We will analyze relinearization method in the homogeneous case [12]. If we have $m < \binom{n+1}{2}$ linearly independent homogeneous equations, we would like a method that allows us to solve the system. When every quadratic monomial is present, we can use linearization to find a parametric solution in $t = \binom{n+1}{2} - m$ free variables z_i, $1 \leq i \leq \binom{n+1}{2} - m$. However during the linearization step we lose information about the relationship between linearized variables. For example, the linearized system does not enforce $y_{ij} = y_i y_j$. All of the solutions to the original non-linear system are also solutions

[2] We are assuming that $q \geq 3$. In this case the number of monomials is found using the formula for selection with repetition $\binom{n+r-1}{r}$, where $r = 1$ to count linear monomials and $r = 2$ to count quadratic monomials. Over $GF(2)$ repetition is not allowed, so there are only $\binom{n}{2}$ quadratic monomials.

to the linearized system, but the linearized system may contain extraneous solutions that do not correspond to solutions of the original system. One method for eliminating extraneous solutions is to check each solution to see if it satisfies every non-linear relationship, throwing out solutions that do not.

If we had a few more original non-linear equations, we could drastically reduce the number of solutions to the linearized system. Our goal is to introduce more equations that reflect the information lost in the linearization step to reduce the number of extraneous solutions. For degree four relinearization, we consider the product of four arbitrary x variables and notice $x_a x_b x_c x_d = (x_a x_b)(x_c x_d) = (x_a x_c)(x_b x_d) = (x_a x_d)(x_b x_c)$, which can be rewritten as $y_{ab} y_{cd} = y_{ac} y_{bd} = y_{ad} y_{bc}$. We would like to find $\binom{n+1}{2} - m$ linearly independent equations of this form so that we may uniquely determine a solution to the y_{ij}'s.

There are only three possible forms for the degree four relinearization equations:

1. $y_{ij} y_{kl} = y_{ik} y_{jl}$
2. $y_{ij} y_{jk} = y_{ik} y_{jj}$
3. $y_{ii} y_{jj} = y_{ij} y_{ij}$

where i, j, k, and l are distinct indices.

For equations of the first type, we can write $y_{ij} y_{kl} = y_{ik} y_{jl} = y_{il} y_{jk}$ which leads to the three equations

$$eq_1 := y_{ij} y_{kl} = y_{ik} y_{jl}$$
$$eq_2 := y_{ij} y_{kl} = y_{il} y_{jk}$$
$$eq_3 := y_{ik} y_{jl} = y_{il} y_{jk}$$

but we see that $eq_1 - eq_2$ is $0 = y_{ik} y_{jl} - y_{il} y_{jk}$ which is the same as eq_3. Thus only two of the three equations are linearly independent, so each choice of four variables leads to two new equations of the first type. There are $2\binom{n}{4}$ linearly independent equations of the first type. For any set of three unique indices, we can make three linearly independent equations of the second type by changing the x_i variable that appears twice for a total of $3\binom{n}{3}$ equations. Similarly there are $\binom{n}{2}$ linearly independent equations of the third type. Thus the number of potential relinearization equations depends only on the number of original variables. The analysis assumes that the relinearization equations are all linearly independent; we will comment on this assumption below.

After using Gaussian elimination on the linearized original system, we can express each y_{ij} as a linear combination of free z_i variables and a constant term, $0 \le i < t$. Thus the relinearization equations may contain linear or quadratic monomials of the z_i variables. There are t linear monomials and $\binom{t+1}{2}$ quadratic monomials as above. We expect that in most cases, every possible linear and quadratic monomial in the z_is will appear in the relinearization equations. If we get lucky and some monomials are missing, we will need fewer linearly independent relinearization equations to solve the second system. However for the purposes of this analysis we do not consider this favorable case. The result is a new quadratic system where the z_i's are the linear variables, and every monomial appears.

This system has a unique solution (or no solution) when the number of relinearization equations is at least the number of possible monomials in the z_i variables, that is, when $2\binom{n}{4} + 3\binom{n}{3} + \binom{n}{2} \ge t + \binom{t+1}{2}$. Solving this inequality for m in terms of n (recall $t = \binom{n+1}{2} - m$) gives

$$\left\lceil \frac{1}{2}n^2 + \frac{1}{2}n + \frac{3}{2} - \frac{1}{6}\sqrt{6n^4 - 6n^2 + 81} \right\rceil \leq m \leq \left\lfloor \frac{1}{2}n^2 + \frac{1}{2}n + \frac{3}{2} + \frac{1}{6}\sqrt{6n^4 - 6n^2 + 81} \right\rfloor$$

However, we are assuming that $m \leq \binom{n+1}{2}$ because otherwise we can solve the system using a single linearization step. $\binom{n+1}{2} < \left\lfloor \frac{1}{2}n^2 + \frac{1}{2}n + \frac{3}{2} + \frac{1}{6}\sqrt{6n^4 - 6n^2 + 81} \right\rfloor$ for all n, so in fact the quadratic system can be solved by relinearization for small m or linearization for large m when $m \geq \left\lceil \frac{1}{2}n^2 + \frac{1}{2}n + \frac{3}{2} - \frac{1}{6}\sqrt{6n^4 - 6n^2 + 81} \right\rceil$.

When m is large enough, we can solve the new quadratic system in t (linear) variables using linearization. The solution to this system gives values for the free variables of the first linearized system, $z_i, 0 \leq i < t$. Substituting in the values for the z_i's, the first linearized system can be solved. Since the solution to the first linearized system determines the y_{ii} variables and $y_{ii} = x_i^2$, we may simply find the square root of each y_{ii}, which exists if the system has a solution. However, many of these solutions to the first linearized system may not be solutions to the original non-linear system because neither the linearized system nor the degree four relinearization equations enforce degree two relationships such as $x_i x_j = y_{ij}$. If a solution exists, each y_{ii} will have two square roots over a general finite field. There are n y_{ii} variables, so there are 2^n solutions to the linearized system that need to be checked. Checking a solution requires computing up to $\binom{n}{2}$ quadratic terms (we do not need to check y_{ii} terms) to see if they match the solution to the relinearization system. We can filter through the 2^n solutions by:

1. Choose a square root x_1 of y_{11} (or any other y_{ii} if $y_{11} = 0$).
2. For $2 \leq i \leq n$ compute $x_i = y_{1i}/x_1$.
3. For $2 \leq i < j \leq n$ compute $x_i x_j$ and make sure it is the same as the value of y_{ij} found by solving the relinearization system.
4. If every y_{ij} is correct, the solutions for *both* square roots of y_{11} are solutions to the non-linear homogenous system, otherwise neither are and there is no solution to the system.

Although we are assuming that every quadratic monomial appears in the original system, this is not always the case. If quadratic monomials are missing, we may view the system in two ways. On one hand, the system has fewer than $\binom{n+1}{2}$ monomials, so we have fewer free variables after the first linearization step. However, this severely restricts the choices for the relinearization equations, so there may not be enough to relinearize this smaller number of free variables. On the other hand, we may think of the system as still having $\binom{n+1}{2} - m$ free variables by assigning each missing monomial to a z_i. We may then use all of the relinearization equations, but these may not be enough to solve the relinearization system in more free variables. It may be possible to solve a system when quadratic monomials are missing, but this case has not been classified to our knowledge. Note that the unfavorable situation caused by monomials missing from the original system is different from the favorable case mentioned above of monomials missing from the relinearization equations.

We now consider the possibility of linear dependencies among the relinearization equations. In [5], Courtois et al. give empirical evidence that degree four relinearization equations are linearly independent except when the field is “very small”. Due to the simplicity of the relinearization equations listed above, it is not difficult to see that, as written, these equations are linearly independent. However it is not clear that these equations are always linearly independent when the representation in the free z_i variables is used. Certainly if there were dependencies when the relinearization equations were written in the y_{ij} representation, there would be dependencies after the substitution was made, but this does not necessarily hold in the reverse direction (at least over "very small" fields). This interesting question could be investigated in future research.

We can determine how many original linearly independent equations are necessary for degree four relinearization to work (for homogeneous systems) to see the savings compared to linearization. We will assume that all $\binom{n+1}{2}$ possible monomials appear in the original equations and then determine the minimum value of m for several values of n. We are still making the critical assumption that the relinearization equations are linearly independent.

This table shows for various n the number of degree four relinearization equations available, the smallest number of original equations needed for relinearization to succeed, and the number of relinearization equations that are needed of the equations available for that choice of n. The last column shows how many original equations would be needed for normal linearization to succeed.

n	Relinearization equations available	Minimal m	Relinearization equations needed	Minimal m without relinearization
3	6	4	5	6
4	20	5	20	10
5	50	7	44	15
6	105	8	104	21
7	196	10	189	28
8	336	12	324	36
9	540	14	527	45
10	825	16	819	55
50	520,625	257	519,689	1275
100	8,332,500	970	8,329,320	5050

Table 1: Degree Four Relinearization Summary

From the table above we can see that degree four relinearization creates considerable savings in terms of the number of original equations needed to solve the system over linearization.

3.1.3 Degree Four Relinearization Example

We will demonstrate degree four relinearization using a small homogeneous system in the original variables x_1, x_2, and x_3 over the field $GF(11)$ (equations mod 11):

$$\begin{bmatrix} 2 & 5 & 9 & 3 & 6 & 5 \\ 10 & 7 & 5 & 6 & 10 & 0 \\ 9 & 0 & 8 & 1 & 6 & 1 \\ 9 & 9 & 10 & 9 & 2 & 7 \end{bmatrix} \begin{bmatrix} x_1^2 \\ x_1x_2 \\ x_1x_3 \\ x_2^2 \\ x_2x_3 \\ x_3^2 \end{bmatrix} \equiv \begin{bmatrix} 7 \\ 3 \\ 1 \\ 8 \end{bmatrix} \text{ mod } 11$$

After linearization, this system becomes

$$\begin{bmatrix} 2 & 5 & 9 & 3 & 6 & 5 \\ 10 & 7 & 5 & 6 & 10 & 0 \\ 9 & 0 & 8 & 1 & 6 & 1 \\ 9 & 9 & 10 & 9 & 2 & 7 \end{bmatrix} \begin{bmatrix} y_{11} \\ y_{12} \\ y_{13} \\ y_{22} \\ y_{23} \\ y_{33} \end{bmatrix} \equiv \begin{bmatrix} 7 \\ 3 \\ 1 \\ 8 \end{bmatrix} \bmod 11$$

Since all four original homogeneous equations are linearly independent, we expect $t = \binom{n+1}{2} - m = 2$ free variables. Solving these simultaneous equations mod 11, we obtain

$$\begin{cases} y_{11} = 10 + 9z_1 \\ y_{12} = 9 + 8z_1 \\ y_{13} = 3z_1 + 6z_2 \\ y_{22} = 10 + 10z_1 + 6z_2 \\ y_{23} = z_1 \\ y_{33} = z_2 \end{cases}$$

We need $t + \binom{t+1}{2} = 5$ linearly independent relinearization equations to solve this system. Per Table 1, there are six linearly independent degree four relinearization equations:

$$\begin{cases} y_{11}y_{23} = y_{12}y_{13} \\ y_{22}y_{13} = y_{12}y_{23} \\ y_{33}y_{12} = y_{13}y_{23} \\ y_{11}y_{22} = y_{12}y_{12} \\ y_{11}y_{33} = y_{13}y_{13} \\ y_{22}y_{33} = y_{23}y_{23} \end{cases}$$

After substituting in the solutions from the linearization step and expanding the relinearization equations become

$$\begin{cases} 5z_1 + 1z_2 + 7z_1^2 + 7z_1z_2 = 0 \\ 10z_1 + 5z_2 + +z_1z_2 + 3z_2^2 = 0 \\ 9z_2 + 8z_1^2 + 2z_1z_2 = 0 \\ 2z_1 + 5z_2 + 4z_1^2 + 10z_1z_2 + 8 = 0 \\ 10z_2 + 2z_1^2 + 6z_1z_2 + 8z_2^2 = 0 \\ 10z_2 + 10z_1^2 + 10z_1z_2 + 6z_2^2 = 0 \end{cases}$$

Now we linearize again by substituting $w_1 = z_1, w_2 = z_2, w_3 = z_1^2, w_4 = z_1z_2, w_5 = z_2^2$ to get the linearized system (of relinearization equations)

$$\begin{bmatrix} 5 & 1 & 7 & 7 & 0 \\ 10 & 5 & 0 & 1 & 3 \\ 0 & 9 & 8 & 2 & 0 \\ 2 & 5 & 4 & 10 & 0 \\ 0 & 10 & 2 & 6 & 8 \\ 0 & 10 & 10 & 10 & 6 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \end{bmatrix} \equiv \begin{bmatrix} 0 \\ 0 \\ 0 \\ 3 \\ 0 \\ 0 \end{bmatrix} \bmod 11$$

which has full rank. Solving this system we find that $w_1 = z_1 = 8, w_2 = z_2 = 9, w_3 = 9, w_4 = 6, w_5 = 4$. Substituting the values of z_1 and z_2 back into the original linearization solution, we find that

$$\begin{bmatrix} y_{11} \\ y_{12} \\ y_{13} \\ y_{22} \\ y_{23} \\ y_{33} \end{bmatrix} = \begin{bmatrix} 5 \\ 7 \\ 1 \\ 1 \\ 8 \\ 9 \end{bmatrix}$$

We determine the values of the original variables x_1, x_2, x_3 by finding the square roots (mod 11) of y_{11}, y_{22}, and y_{33} respectively. We find that $x_1 \equiv \pm 4 \bmod 11$, $x_2 \equiv \pm 1 \bmod 11$, and $x_3 \equiv \pm 3 \bmod 11$ are solutions to the linearized original system and the relinearization system.

To filter out extraneous solutions, we arbitrarily choose $x_1 = 4$ and then compute $x_2 = y_{12}/x_1 \equiv 10 \bmod 11$ and $x_3 = y_{13}/x_1 \equiv 3 \bmod 11$. To check that this is a solution, we also need to compute the rest of the cross terms: $x_2 x_3 = 8 = y_{23}$. Since all cross terms are correct, one solution to the non-linear system is $\{x_1 \equiv 4 \bmod 11,\ x_2 \equiv 10 \bmod 11,\ x_3 \equiv 3 \bmod 11\}$. Since the system is homogeneous, we can negate every x_i to obtain a different solution, so $\{x_1 \equiv 7 \bmod 11,\ x_2 \equiv 1 \bmod 11,\ x_3 \equiv 8 \bmod 11\}$ also solves the original system.

3.1.4 Generalized Relinearization

There is no reason we are restricted to degree four relinearization. This idea will be revisited with the XL algorithm and its variants, but for now we will show that degree six relinearization can have benefits over degree four relinearization, an idea introduced in [5].

Degree six relinearization works by linearizing and solving the original quadratic system as described above. The y_{ij} variables are expressed as $t = \binom{n+1}{2} - m$ free z_i variables, $0 \le i < t$. Rather than considering the product of two y_{ij} terms, we now consider products of three y_{ij} terms which is really a product of six original x_i variables. Since the y_{ij}s are expressed as a linear combination of the free z_i variables and a constant, these relinearization equations could contain up to $\binom{t}{1} + \binom{t+1}{2} + \binom{t+2}{3}$ monomials in the z_i variables.

There are nine different types of degree six relinearization equations which can be classified according to how many times each original x variable appears in the product on one side of the equation. One side of each degree six relinearization equation has the following form (which we will associate to equations of that type):

1. $y_{ij} y_{kl} y_{ab}$

2. $y_{ii} y_{jk} y_{ab}$

3. $y_{ii}y_{ij}y_{kl}$
4. $y_{ii}y_{ii}y_{jk}$
5. $y_{ii}y_{jj}y_{kl}$
6. $y_{ii}y_{ij}y_{jk}$
7. $y_{ii}y_{jj}y_{kk}$
8. $y_{ii}y_{ij}y_{jj}$
9. $y_{ii}y_{ii}y_{jj}$

where i, j, k, l, a, and b are distinct indices.

Now we will count the number of degree six relinearization equations of each type. For products of the first type, given a set of six distinct original variables, there are $\binom{6}{2}$ choices for the first term y_{ij}, $\binom{4}{2}$ choices for the second term y_{kl} and $\binom{2}{2} = 1$ choice for the third term y_{ab}. Since $y_{ab}y_{cd} = y_{cd}y_{ab}$ and $y_{ab} = y_{ba}$, and order does not matter, there are $\binom{6}{2}\binom{4}{2}/3! = 15$ "non-trivial" permutations of products of the first type. From these 15 permutations we can create 14 "non-trivial" equations, so there are $14\binom{n}{6}$ degree six relinearization equations of the first type. For products of the second type we can rearrange the indices to get the following nine non-trivial permutations:

$$\begin{cases} y_{ii}y_{jk}y_{ab} \\ y_{ii}y_{ja}y_{kb} \\ y_{ii}y_{jb}y_{ak} \\ y_{ia}y_{ib}y_{jk} \\ y_{ij}y_{ik}y_{ab} \\ y_{ij}y_{ia}y_{kb} \\ y_{ij}y_{ib}y_{ka} \\ y_{ik}y_{ia}y_{jb} \\ y_{ik}y_{ib}y_{ja} \end{cases}$$

Given 5 indices, there are $\binom{5}{1}$ choices for the quadratic term, so there are $8\binom{5}{1}\binom{n}{5}$ equations of the second type.

Similarly we can count the number of non-trivial equations of the other types. The total number of non-trivial equations is $14\binom{n}{6} + 8\binom{5}{1}\binom{n}{5} + 3\binom{4}{1}\binom{n}{4} + \binom{3}{1}\binom{n}{3} + 5\binom{4}{2}\binom{n}{4} + 2\binom{3}{1}\binom{2}{1}\binom{n}{3} + 4\binom{n}{3} + \binom{n}{2} + \binom{2}{1}\binom{n}{2} = (7n^6 + 15n^5 + 25n^4 - 15n^3 - 32n^2)/360$.

The idea of degree six relinearization is that it should be more powerful than degree four relinearization since there are more relinearization equations. However, there are more monomials in the relinearization system, so we also require more linearly independent relinearization equations to solve the system. We did observe that degree six relinearization is more powerful for a homogeneous system in five original variables: degree four relinearization required seven original equations while degree six relinearization required only six original equations. See Appendix A for systems that demonstrate the power and linear dependencies of degree six relinearization. Thus it appears that degree six relinearization works in cases when degree four relinearization does not, and in all cases we observed that degree six relinearization worked whenever degree four relinearization worked.

3.1.5 Linear Dependencies for Generalized Relinearization

Although there are many more possible equations for degree six relinearization than for degree four relinearization, there are many dependencies among the degree six equations. These dependencies have not been completely classified, but [5] identifies two sources of linear dependencies among equations from generalized relinearization.

We call a relinearization equation special if exactly two y_{ij} variables are different. For example, $y_{12}y_{34}y_{56} = y_{13}y_{24}y_{56}$ is special while $y_{12}y_{34}y_{56} = y_{13}y_{25}y_{46}$ is not special. We can compare the number of non-trivial equations to the number of special equations generated from each type above:

Type	Non-trivial equations	Special equations
1	$14\binom{n}{6}$	$6\binom{n}{6}$
2	$40\binom{n}{5}$	$20\binom{n}{5}$
3	$12\binom{n}{4}$	$12\binom{n}{4}$
4	$3\binom{n}{3}$	$3\binom{n}{3}$
5	$30\binom{n}{4}$	$18\binom{n}{4}$
6	$12\binom{n}{3}$	$12\binom{n}{3}$
7	$4\binom{n}{3}$	$3\binom{n}{3}$
8	$\binom{n}{2}$	$\binom{n}{2}$
9	$2\binom{n}{2}$	$2\binom{n}{2}$

There are $(n^6 + 5n^5 + 35n^4 - 65n^3 + 24n^2)/120$ special degree six relinearization equations.

Lemma 3.1. *The set of special equations spans the set of non-trivial equations.*

Proof. Let $(a, b, ..., e, f) \sim (a', b', ..., e', f')$ be permuted tuples each with an even number of elements in $\{1, ..., n\}$ and consider the non-trivial equation $y_{ab}...y_{ef} = y_{a'b'}...y_{e'f'}$. Without loss of generality, we assume that the indices are sorted so that the first index of each y_{ij} is no more than the second index and that the first index of consecutive y_{ij}'s are non-decreasing. Given a set of permuted tuples $p_i, 1 \leq i \leq r$ and equations $\{p_1 = p_2,\ p_2 = p_3,\ ...,\ p_{r-1} = p_r\}$, we define a chain of equations to be the equality $p_1 = p_2 = p_3 = ... = p_{r-1} = p_r$ that equates the first permutation with the last. We will show that given a chain of equations, the equation $p_1 = p_r$ is linearly dependent on the equations in the chain. We will use induction on the size of the tuples to show that any permutation can be obtained by creating a chain of special equations, which shows that the set of special equations spans the non-trivial equations.

Base Case: If there are four elements in each tuple, there are only two possibilities. Either the permutations are the same, or they are off by exactly one permutation. If the both permutations are p_1, we can choose any special equation of the form $p_1 = p_i$ and then chain it with the special equation $p_i = p_1$. In the second case, there is a special equation equating the two permutations since every product of y_{ij}'s with exactly two y_{ij}'s permuted yields a special equation.

Induction Step: Assume that for tuples of size $2k$ that we can always create a special equation chain from the first tuple to the second permuted version. We will show the result holds for tuples of size $n = 2(k + 1)$.

Since the two tuples contain the same indices, we can find y_{st} and y_{uv} such that (without loss of generality) $s = a'$ and $u = b'$. Use the special equation $y_{ab}y_{st}y_{uv}...y_{ef} = y_{ab}y_{su}y_{tv}...y_{ef}$ and reorder as $y_{ab}y_{st}y_{uv}...y_{ef} = y_{a'b'}y_{ab}y_{tv}...y_{ef}$. Now the right side of the special equation has one term in common with the second tuple. The remaining indices of both tuples are the same, so we consider the two new tuples $(a, b, t, v, ...e, f) \sim (c', d', ..., e'f')$ of size $2k$. By the induction hypothesis, we can create a chain connecting these two tuples. If we multiply both sides of each equation in the chain by $y_{a'b'}$, each equation in the chain is still special and the chain links the two original permuted tuples. $\square$

This shows that we need only consider special equations. However, there are further dependencies among these equations. After the first linearization step, we express each y_{ij} as a linear combination of $t = \binom{n+1}{2} - m$ z_i variables (plus a constant). Rather than writing a degree six special relinearization equation as a product of three y_{ij}s, we can instead think of it as a z_i-linear combination of degree four relinearization equations, where we multiply each degree four equation by either zero or one z_i variables. Since the y_{ij}s are linear combination of the z_is, these equations span the set of special degree six relinearization equations.

For example, if there are 2 free variables z_1 and z_2, and $y_{56} = 1 + z_1 - 2z_2$, $y_{12}y_{34}y_{56} = y_{13}y_{24}y_{56}$ can be written as the sum of $y_{12}y_{34} = y_{13}y_{24}$, $y_{12}y_{34}z_1 = y_{13}y_{24}z_1$, and $-2y_{12}y_{34}z_2 = -2y_{13}y_{24}z_2$.

We call $\{$degree four relinearization equations$\} \bigcup z_i \cdot \{$degree four relinearization equations$\}$ the set of degree six z-equations. Since there were $2\binom{n}{4} + 3\binom{n}{3} + \binom{n}{2}$ degree four relinearization equations and there are $t = \binom{n+1}{2} - m$ z_i variables, there are $(2\binom{n}{4} + 3\binom{n}{3} + \binom{n}{2}) \cdot (1 + \binom{n+1}{2} - m)$ degree six z-equations.

There are fewer degree six z-equations than degree six special equations when $(2\binom{n}{4} + 3\binom{n}{3} + \binom{n}{2}) \cdot (1 + \binom{n+1}{2} - m) < (n^6 + 5n^5 + 35n^4 - 65n^3 + 24n^2)/120$. Solving this inequality for m, we find that we should use z-equations when $\frac{119n^3+234n^2+319n+264}{240(n+1)} < m < \binom{n+1}{2}$ because there are fewer z-equations.[3] The point of using the smaller number of equations is to get a better estimate of the number of linearly independent degree six relinearization equations.

We conjecture that similar results hold for higher degree relinearization. For example, if we consider degree eight relinearization, we expect that there are fewer z-equations of the forms $y_{ab}y_{cd} = y_{ac}y_{bd}$, $y_{ab}y_{cd}z_i = y_{ac}y_{bd}z_i$, and $y_{ab}y_{cd}z_iz_j = y_{ac}y_{bd}z_iz_j$ than there are special equations when $m \approx n^2$.

The authors of [5] claims to have found other sources of linear dependence, but we did not investigate other potential dependencies. We also point out that even if we could show that equations of the form above are linearly independent, there is the possibility of "random" linear dependence, based on the linear combinations of particular y_{ij}s; see Appendix B.

For example, suppose we have a system in five original variables and that after linearization we have:

[3]Recall that when $m \geq \binom{n+1}{2}$, there are no free variables after the first linearization and so there is no need for relinearization.

$$\begin{bmatrix} y_{23} = 1 + z_1 + z_2 \\ y_{35} = z_1 - z_2 \\ y_{25} = z_2 + 1 \\ y_{33} = z_1 - 1 \\ y_{11} = z_2 \\ y_{12} = z_1 + z_2 - 1 \\ y_{15} = z_2 - 1 \\ y_{13} = z_1 \\ y_{14} = z_1 + 1 \\ y_{34} = 2z_1 + z_2 - 3 \end{bmatrix}$$

The relinearization equations $y_{23}y_{35} = y_{25}y_{33}$, $y_{11}y_{25} = y_{12}y_{15}$, and $y_{13}y_{14} = y_{11}y_{34}$ appear linearly independent. But after substituting in the linear combinations in terms of the free variables, we find that the sum of the first two equations is the third equation. Thus even when all systematic linear dependencies have been identified, linear dependencies may still exist. The complete categorization of the systematic dependencies is still an open question.

3.1.6 Generalization to Non-Homogeneous Case

We have seen that relinearization can be used to solve a system of homogeneous quadratic equations. There seems to be a relatively simple extension to a system of non-homogeneous quadratic equations. The results in Table 1 show when relinearization works based on the number of free variables after the first relinearization. If we introduce a linear monomial into the original system (making it non-homogeneous), one way we can ensure that relinearization still works is to add one more original equation as well to keep the number of free variables the same after linearization. Thus for non-homogeneous systems, if $1 \leq l \leq n$ linear monomials appear [4], we need exactly l more original equations than the table above indicates. In the homogeneous case we solve for the original x_i variables by taking the square root of y_{ii}. Over a general field this has two square roots, but in the non-homogeneous system we directly solve for the value of $y_i = x_i$ for any linear monomials that appear in the original system.

An alternate approach may not require more original equations in the non-homogeneous case. Introducing linear monomials into the original system increases the number of linearized variables as well. However we also get $\binom{n+1}{2}$ more relinearization equations of the form $x_i x_j = y_{ij}$. Note that unlike the approach above, this alternate approach depends on how many linear monomials appear in the original system. The more linear monomials that appear in the original system, the more of the new relinearization equations we can use. These extra relinearization equations (which do not appear to contain any systematic linear dependencies) may allow us to solve the original system with the same number of original equations as the homogeneous case, and in any case will likely allow us to use fewer than the n extra original equations that we would need to solve the non-homogeneous system using only the homogeneous relinearization equations.

This case has not been thoroughly analyzed and this would be an excellent area for future work.

[4]Unlike the situation for quadratic monomials, every linear monomial need not appear because they do not affect the number of relinearization equations.

3.2 XL

3.2.1 Basic XL

We now consider the XL method for solving generic systems of non-linear equations [5]. The basic XL method can be used to solve a system of overdetermined equations of degree d. The idea is to multiply each original equation by every possible monomial of degree less than or equal to $D-d$, where D is the XL parameter. We will denote by x^k the set of all monomials of degree k in the n variables x_0 through x_{n-1}. Each of the original m equations in n variables is of the form $f_i(x_0, x_2, ..., x_{n-1}) = b_i$. We will refer to $l_i = f_i(x_1, x_2, ..., x_n) - b_i = 0$, and l will be the set of these m equations. We will denote by $x^k l$ the set of all equations multiplied by every monomial of degree k and will refer to $\bigcup\limits_{k=0}^{D-d} x^k l$ as the XL system.

For example, if $n = 3$, $x^2 = \{x_0x_1,\ x_0x_2,\ x_1x_2,\ x_0^2,\ x_1^2,\ x_2^2\}$ and $x^2l = \{x_0x_1l_1,\ x_0x_2l_1,\ x_1x_2l_1,\ x_0^2l_1,\ x_1^2l_1,\ x_2^2l_1,\ x_0x_1l_2,\ x_0x_2l_2,\ x_1x_2l_2,\ ...\ \}$.

The algorithm consists of four steps [5]:

1. **Multiply** Choose an XL parameter D and compute $x^i l, 0 \le i \le D-d$.
2. **Linearize** Linearize the system of all of the equations from step 1. Order the monomials so that monomials in one variable (i.e. powers of a particular x_i) are eliminated last.
3. **Solve** If Gaussian elimination yields an equation in monomials in one variable, we can use Berlekamp's algorithm [13] to solve the equation over the finite field $GF(q)$.
4. **Repeat** Go back to step 2 and solve for a different variable until every original x_i has been found.

Over "large" fields ($q > D$) there are D monomials in one variable (x_i, x_i^2, ..., x_i^D). In this case there are $R = m \cdot \sum\limits_{i=0}^{D-d} \binom{n+i-1}{i}$ equations and $T = \sum\limits_{i=0}^{D} \binom{n+i-1}{i}$ monomials (including the constant monomial) in the XL system. The goal of XL (step 3) is to get an equation in these D monomials (and a constant). In general the XL system can have at most $D-1$ free variables. Let I be the number of linearly independent equations in the XL system. Clearly $I \le R$, the total number of equations, and $I < T$ since the rows of the XL system are in a $T-1$-dimensional vector space. These conditions imply $I \le min(R, T-1)$. $T-1$ is the number of non-constant monomials and $(T-1)-I$ is the number of free variables after performing Gaussian elimination on this system. Then for XL to succeed over a large field we need $(T-1)-I \le D-1$ which can be simplified to $I \ge T-D$.

We now consider XL when the field is "small" ($q \le D$). Over any finite field $GF(q)$ we have $x^q = x$ and since our equations have monomials of degree $\ge q$, we can reduce some terms. When working over small fields, we will always reduce exponents to the range $1, ...q-1$. Since the monomials in one variable are now x_i, x_i^2, ..., x_i^{q-1}, we can solve the XL system if there are at most $q-2$ free variables, so the new condition on I becomes $I \ge T-(q-1)$. We will be concerned with the analysis when $q = 2$. In this case we need $I \ge T-1$, and since $T-1$ is the number of non-constant monomials, this implies that the XL system can have no free variables. Since $x_i = x_i^2$, no x_i appears in a monomial to a power larger than 1. The number of equations over $GF(2)$ is $R = m \cdot \sum_{i=0}^{D-d} \binom{n}{i}$ and the number of monomials is $T = \sum_{i=0}^{D} \binom{n}{i}$.

Now we explain how to order the monomials of the XL system in step 2 of the XL algorithm. Rearrange the columns of the XL system so that monomials that are multiples of x_1 come first (in any order). Next put any remaining monomials that are multiples of x_2 (in any order). Continue in this way until all monomials have been ordered. The important property of this order is that when placing monomials that are multiples of x_i, none of the monomials are multiples of any of the variables $\{x_1, x_2, ..., x_{i-1}\}$. In particular, at the last step of the ordering, the only remaining monomials will be powers of x_n.

For example, consider a system with three variables over a large field, with $D = 4$. The monomials in the resulting XL system are

$$\{x_1, x_2, x_3, x_1x_2, x_1x_3, x_2x_3, x_1^2, x_2^2, x_3^2, x_1x_2x_3, x_1^2x_2, x_1^2x_3, x_2^2x_1, x_2^2x_3, x_3^2x_1, x_3^2x_2, x_1^3, x_2^3, x_3^3\}$$

To order these monomials for the XL algorithm, first choose any monomials containing x_1 in any order:

$$\{x_1x_2, x_1x_3, x_1^2, x_1x_2x_3, x_1^2x_2, x_1^2x_3, x_2^2x_1, x_3^2x_1, x_1^3, x_1\}$$

Next add monomials in x_2:

$$\{x_1x_2, x_1x_3, x_1^2, x_1x_2x_3, x_1^2x_2, x_1^2x_3, x_2^2x_1, x_3^2x_1, x_1^3, x_1, x_2, x_2x_3, x_2^2, x_2^2x_3, x_3^2x_2, x_2^3\}$$

Finally add the monomials in x_3:

$$\{x_1x_2, x_1x_3, x_1^2, x_1x_2x_3, x_1^2x_2, x_1^2x_3, x_2^2x_1, x_3^2x_1, x_1^3, x_1, x_2, x_2x_3, x_2^2, x_2^2x_3, x_3^2x_2, x_2^3, x_3^2, x_3^3, x_3\}$$

When the condition on I is satisfied ($I \geq T - D$ for large fields or $I \geq T - (q-1)$ for small fields), this ordering makes step 4 of the XL algorithm unnecessary. Since there are D (or $q-1$) variables in x_n but at most $D-1$ (or $q-2$) free variables, at least one of the monomials in x_n is determined by free variables. By our ordering, it can only depend on free variables corresponding to other powers of x_n. This gives us an equation in one variable which can be solved using Berlekamp's algorithm. We can then compute every monomial in the single variable x_n, so the set of monomials in the two variables x_{n-1} and x_n now become monomials (with constants) in one variable. There are more than D (or $q-1$) of these monomials, so again at least one of them is determined by monomials only in the variables x_{n-1} and x_n. We can write a new equation in one variable (since x_n is determined) and solve with Berlekamp's algorithm to determine the value of x_{n-1}. Continuing in this way, we can find the value of every variable. Thus when the condition on I is satisfied, there is no need to repeat any of the algorithm with step 4 and step 3 is guaranteed to work.

If the condition on I is not satisfied, XL may still work if we get lucky. It would probably be possible to order the monomials (depending on the system) so that step 3 succeeds, but this has not been investigated.

We also note that relinearization equations of degree D are equivalent to a subset of equations from the XL system with parameter D. Thus XL is at least as powerful as relinearization with the same D. In particular when $D = d$, XL reduces to linearization. For an outline of the proof, see [5].

3.2.2 Results and Complexity

In this section we summarize the empirical findings in [5]. We only consider systems where $m \geq n$. The XL system contains R equations in T (linearized) variables. To find one solution to the original system, the complexity of XL is essentially the complexity of solving a linear system in T variables via Gaussian reduction. Thus step 2 of the algorithm requires about T^{ω} operations, where ω is the exponent of Gaussian reduction. This exponent is at most 2.376 in theory [5], but in practice the fastest implementation of Strassen's algorithm [16] requires about $7/64T^{\log_2 7}$ operations.

As we will see, the time complexity of XL increases with D, so we would like to know how large D needs to be in several interesting cases. The following estimations for D are based on empirical evidence over a large field from [5]. When $m = n$, empirical evidence shows XL requires $D = 2^n$ to succeed, so XL only works for very small n in this case.[6] If we add just one more equation so that $m = n + 1$, we only need $D = n$. When $m = n + 2$, we only require $D \approx \sqrt{n}$. Thus we see that the complexity of XL decreases rapidly when we add even a single equation and $m \approx n$.

3.2.3 Analysis Over Small Fields

We can estimate D in terms of m and n over small fields using the asymptotic analysis of XL in [7]. If we assume that $D << n$ and that most of the R equations are linearly independent (we will justify these important assumptions), we have $T \approx \binom{n}{D}$ so the complexity of XL is about $7/64\binom{n}{D}^{\log_2 7}$. In this case we expect to succeed when $R \geq T$. Taking only the largest term in each sum (over small fields), we need $m\binom{n}{D-d} \geq \binom{n}{D}$, so we need $m \geq \frac{(n-D+d)\cdots(n-D+1)}{D(D-1)\cdots(D-d+1)} \approx \frac{n^d}{D^d}$. Solving for D, we find $D \approx \frac{n}{m^{1/d}}$. Since $T \approx \binom{n}{D}$ when $D << n$, the complexity in this case is about $\binom{n}{n/m^{1/d}}^{\omega}$. In particular if $m = \epsilon n^2$ where $0 < \epsilon < 1/2$, we need $D \approx \left\lceil \frac{1}{\sqrt{\epsilon}} \right\rceil$, and we expect XL to be polynomial in n.

A separate analysis of XL over small fields was conducted in [8]. As with the results for large fields, the results are largely empirical, but the authors of that paper were able to explain some of the linear dependencies. All empirical results are over $GF(2)$ and the systems are quadratic ($d = 2$).

When $D = 3$ empirical evidence showed that $I = min(T, R) - \alpha$ where $\alpha = 0$, 1, or 2. It is unclear why these small number of linear dependencies arise in some cases, but it may be related to the number of solutions of the system.

When $D = 4$ empirical evidence suggests that $D = min(T, R - \binom{m}{2} - m)$. This implies that there are always $\binom{m}{2} + m$ systematic linear dependencies. Since the system is quadratic, each original equation l_i is a linear combination of monomials of degree at most 2. Also $D - d = 2$, so we are multiplying original equations by all monomials of degree ≤ 2. The linear dependencies are best demonstrated with an example. Suppose $l_i = x_1x_3 + x_4$ and $l_j = x_1x_2 + x_4x_7$.

The following equations are all included in the XL system for $D = 4$:

1. l_i
2. $x_1x_2l_i$

[5]This exponent can be obtained by the Coppersmith-Winograd algorithm, but only for extremely large matrices [3].

[6]Clearly the complexity estimation above does not apply in this case, but the idea is that in any case, the complexity of XL depends in some way on D.

3. $x_4 x_7 l_i$

4. l_j

5. $x_1 x_3 l_j$

6. $x_4 l_j$

We see that (1) + (2) + (3) $= l_i(l_j) = l_j(l_i) =$ (4) + (5) + (6). This relationship exists for every distinct pair of original equations, for a total of $\binom{m}{2}$ linear dependencies. Similarly, we get m more linear dependencies due to the relationship $l_i(l_i) = l_i$ since we are working over $GF(2)$. This explains the empirically observed $\binom{m}{2} + m$ linear dependencies.

When $D = 5$ we get all of the linear dependencies from the $D = 4$ case, but we get even more dependencies. The new dependencies are of the form $l_i(l_j)x_k = l_j(l_i)x_k$ and $l_i(l_i)x_k = l_i x_k$. There are $n\binom{m}{2} + mn$ such dependencies for a total of $[\binom{m}{1} + \binom{m}{2}] \cdot [\binom{n}{0} + \binom{n}{1}]$, which matches the empirical results for $D = 5$ obtained in [8].

Based on these dependency estimates, we can summarize the empirical results obtained in [8]. We expect that for $q = 2$, $d = 2$, and $4 = 2d \leq D \leq 5 = 3d - 1$ there are $[\binom{m}{1} + \binom{m}{2}] \cdot \sum_{i=0}^{D-4} \binom{n}{i}$ linear dependencies. It is possible that more dependencies arise for larger D, but it is not clear what form these dependencies would have if they exist. This estimation of linear dependencies extends naturally to the case when $d \geq 2$.

For $d \leq D < 2d$ we do not expect any systematic dependencies (at least of the form described above) because the dependency equations were derived from products of l_i equations. When $D < 2d$, equations involving products of l_i's are not in the XL system because a product of l_i's has degree $\geq 2d$ but each XL equation has degree $< 2d$. For $2d \leq D \leq 2d+n$ we expect $[\binom{m}{1} + \binom{m}{2}] \cdot \sum_{i=0}^{D-2d} \binom{n}{i}$ linear dependencies of the form described above. Conjecture A.3.1 in [7] summarizes these results:

Conjecture 3.2. *For $D \in \{d...2d-1\}$, $I = min(T, R) - \epsilon$ where ϵ is small, for example, $\epsilon < 5$.*

For $D \in \{2d...3d-1\}$, $I = min\left(T, R - [\binom{m}{1} + \binom{m}{2}] \cdot \sum_{i=0}^{D-2d} \binom{n}{i}\right) - \epsilon$ where ϵ is small, for example, $\epsilon < 5$.

This conjecture was confirmed with empirical evidence in [7]. One of the biggest questions concerning this conjecture is the size of ϵ. It is unclear if these small number of dependencies are random or systematic. There may be more dependencies when $d > 2$, but it is not clear what these dependencies would look like. Regardless of extra dependencies, it appears that the dependencies described for $d = 2$ also apply when $d > 2$. This relatively small number of linear dependencies justifies the assumptioni in Section 3.2.2 that XL will succeed when $R > T$. In practice, it is unlikely that we can choose m so that $R \approx T$; in most cases to get $R > T$, we will have $R >> T$ so the linear dependencies do not affect the analysis.

3.2.4 FXL

The FXL algorithm is designed to capitalize on how rapidly D falls as $m - n$ increases, even by one or two. FXL involves guessing the value of a few x_i variables, and then solving the system in the

the same number of equations but with the smaller number of variables. For example, we would like for $m - n \geq 2$ so that $D \approx \sqrt{n}$. If the original system has $m - n = 1$, we can guess the value of one variable and then solve the resulting system of m equations in $n - 1$ variables.

Given a system with $m - n = 1$ we need $D = n$, so the entire algorithm requires about $7/64\left(\sum_{i=1}^{n}\binom{n}{i}\right)^{\omega} \approx 2^{\omega n}$ operations. If instead we use the FXL algorithm and guess the value of one variable over $GF(q)$, we can solve the resulting system in $n - 1$ variables in about $7/64q\left(\sum_{i=1}^{\sqrt{n}}\binom{n}{i}\right)^{\omega} \approx q\binom{n}{\sqrt{n}}^{\omega}$ operations.

Thus FXL offers large savings when adding a few equations to the original system drastically reduces the complexity (the XL parameter D). However when the field is small, equations of the form $x_i^q = x_i$ have degree $< D$, and so could be included in the XL system. In fact, these n equations are included in the XL system implicitly when we reduce all exponents to the range $1, 2, ...q - 1$. For example, over $GF(2)$ the monomial $x_1^2 x_2 = x_1 x_2$ because $x_1^2 = x_1$. Thus any system that has $m = n$ or $m = n + 1$ requires only $D = \sqrt{n}$ due to these implicit equations. This result is confirmed by empirical evidence from [8]: XL can solve a system with $n = m = 10$ and $D = 4$ (rather than $D = 2^{10}$ as over a large field) and $n = m = 20$ can be solved using $D = 5$. Note this justifies our assumption in Section 3.2.3 that $D << n$, so our analysis is valid over large fields when $m \geq n + 2$ and over small fields when $m \geq n$.

Thus if we guess r variables using FXL, the complexity is multiplied by 2^r but D does not get smaller, so FXL is not as interesting over small fields. In any case, we will see that the XL' algorithm uses the same idea to obtain much better results.

3.2.5 XL'

Rather than guessing variables before we start XL which results in having to repeat the process several times, XL' allows us to perform XL once and do an exhaustive search over a small number of variables at the end. The primary advantage is that XL' allows us to use XL with a (slightly) smaller D.

Given a $R \times T$ XL system, we expect XL to succeed when $I \geq T - min(D, q - 1)$. The idea of basic XL in step 4 is to get an equation in one original variable. With XL', our goal is to get at least r equations in r original variables. When we have at least r equations, we expect this system to have one solution. We then solve this smaller system by exhaustive search (there are q^r possibilities to check, so over $GF(2)$ we can allow r up to about 40 [8]).

Let $\phi = \sum_{i=0}^{D}\binom{r}{i}$ be the number of monomials in r variables of degree $\leq D$ (including a constant, since T also includes the constant). We can reorder the columns of the XL system so that the ϕ linearized variables in r original variables occur last. When $T - I \leq \phi$, we can reduce the system to equations in the r XL' variables. [7] If we further require that $T - I \leq \phi - r$ we can reduce the XL system to a system of at least r equations in the ϕ monomials in r variables (possibly after reordering the rows).

For example, suppose the XL system (mod11) with $n = 4$ and $D = 3$ is

[7]Keep in mind that $T =$ number of monomials $+ 1$. When we have equations in ϕ variables, only $\phi - 1$ of those are free. Thus we actually need the number of free variables to be strictly less than ϕ, because if all of the ϕ variables were free, there would be no equations in only those variables.

$$\begin{bmatrix} 1 & 9 & 4 & 6 & 6 & 6 & 0 & 4 & 6 & 7 & 8 & 2 & 0 & 1 \\ 5 & 3 & 6 & 7 & 1 & 0 & 2 & 4 & 8 & 2 & 6 & 4 & 0 & 9 \\ 0 & 2 & 1 & 9 & 9 & 6 & 3 & 0 & 4 & 10 & 9 & 3 & 4 & 0 \\ 4 & 3 & 4 & 0 & 2 & 4 & 3 & 2 & 0 & 4 & 1 & 9 & 9 & 10 \\ 1 & 2 & 10 & 4 & 8 & 4 & 0 & 2 & 10 & 2 & 10 & 0 & 4 & 5 \\ 8 & 4 & 0 & 2 & 2 & 4 & 7 & 10 & 5 & 2 & 8 & 5 & 5 & 8 \\ 10 & 4 & 7 & 0 & 3 & 9 & 10 & 10 & 8 & 3 & 3 & 10 & 9 & 10 \\ 7 & 3 & 7 & 7 & 0 & 1 & 4 & 2 & 4 & 7 & 1 & 8 & 4 & 3 \\ 9 & 6 & 8 & 6 & 6 & 3 & 2 & 10 & 8 & 10 & 3 & 5 & 1 & 1 \\ 0 & 2 & 1 & 7 & 9 & 10 & 5 & 7 & 8 & 5 & 10 & 4 & 6 & 4 \\ 9 & 5 & 0 & 3 & 7 & 1 & 6 & 10 & 10 & 3 & 5 & 10 & 2 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_{12} \\ y_{13} \\ y_{14} \\ y_{23} \\ y_{24} \\ y_{34} \\ y_{123} \\ y_{124} \\ y_{134} \\ y_{234} \end{bmatrix} = \begin{bmatrix} 10 \\ 3 \\ 1 \\ 3 \\ 7 \\ 1 \\ 1 \\ 7 \\ 8 \\ 9 \\ 1 \end{bmatrix}$$

If this were the final XL system, it is clear that we would not succeed since $R < T$. However, we can succeed with XL' with $r = 3$. We arbitrarily choose x_1, x_2, and x_3 as the variables that will be found by exhaustive search. The next step is to reorder the columns so that the $\phi - 1 = 7$ monomials in these three variables occur last. We also put y_4 immediately before the ϕ variables so that it can be found immediately after exhausting to find y_1, y_2, and y_3.

$$\begin{bmatrix} 6 & 0 & 6 & 7 & 2 & 0 & 1 & 1 & 9 & 4 & 6 & 6 & 4 & 8 \\ 7 & 2 & 8 & 2 & 4 & 0 & 9 & 5 & 3 & 6 & 1 & 0 & 4 & 6 \\ 9 & 3 & 4 & 10 & 3 & 4 & 0 & 0 & 2 & 1 & 9 & 6 & 0 & 9 \\ 0 & 3 & 0 & 4 & 9 & 9 & 10 & 4 & 3 & 4 & 2 & 4 & 2 & 1 \\ 4 & 0 & 10 & 2 & 0 & 4 & 5 & 1 & 2 & 10 & 8 & 4 & 2 & 10 \\ 2 & 7 & 5 & 2 & 5 & 5 & 8 & 8 & 4 & 0 & 2 & 4 & 10 & 8 \\ 0 & 10 & 8 & 3 & 10 & 9 & 10 & 10 & 4 & 7 & 3 & 9 & 10 & 3 \\ 7 & 4 & 4 & 7 & 8 & 4 & 3 & 7 & 3 & 7 & 0 & 1 & 2 & 1 \\ 6 & 2 & 8 & 10 & 5 & 1 & 1 & 9 & 6 & 8 & 6 & 3 & 10 & 3 \\ 7 & 5 & 8 & 5 & 4 & 6 & 4 & 0 & 2 & 1 & 9 & 10 & 7 & 10 \\ 3 & 6 & 10 & 3 & 10 & 2 & 1 & 9 & 5 & 0 & 7 & 1 & 10 & 5 \end{bmatrix} \begin{bmatrix} y_4 \\ y_{14} \\ y_{24} \\ y_{34} \\ y_{124} \\ y_{134} \\ y_{234} \\ y_1 \\ y_2 \\ y_3 \\ y_{12} \\ y_{13} \\ y_{23} \\ y_{123} \end{bmatrix} = \begin{bmatrix} 10 \\ 3 \\ 1 \\ 3 \\ 7 \\ 1 \\ 1 \\ 7 \\ 8 \\ 9 \\ 1 \end{bmatrix}$$

Next we perform Gaussian elimination until we get $\phi - T + I$ equations in ϕ variables.

$$\begin{bmatrix} 6 & 0 & 6 & 7 & 2 & 0 & 1 & 1 & 9 & 4 & 6 & 6 & 4 & 8 \\ 0 & 9 & 10 & 8 & 2 & 0 & 5 & 9 & 2 & 6 & 6 & 7 & 8 & 7 \\ 0 & 0 & 10 & 6 & 3 & 4 & 6 & 1 & 8 & 4 & 9 & 2 & 6 & 2 \\ 0 & 0 & 0 & 4 & 9 & 8 & 8 & 6 & 6 & 4 & 8 & 5 & 6 & 8 \\ 0 & 0 & 0 & 0 & 4 & 9 & 3 & 4 & 5 & 9 & 6 & 7 & 0 & 5 \\ 0 & 0 & 0 & 0 & 0 & 5 & 3 & 4 & 8 & 6 & 1 & 7 & 7 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 8 & 4 & 7 & 6 & 1 & 2 & 9 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 1 & 9 & 7 & 7 & 8 & 5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 7 & 6 & 8 & 3 & 1 & 0 & 10 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 9 & 6 & 3 & 7 & 0 & 6 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 7 & 9 & 0 & 4 & 6 & 4 & 8 \end{bmatrix} \begin{bmatrix} y_{14} \\ y_{24} \\ y_{34} \\ y_{124} \\ y_{134} \\ y_{234} \\ y_4 \\ y_1 \\ y_2 \\ y_3 \\ y_{12} \\ y_{13} \\ y_{23} \\ y_{123} \end{bmatrix} = \begin{bmatrix} 10 \\ 5 \\ 10 \\ 10 \\ 10 \\ 7 \\ 3 \\ 9 \\ 8 \\ 1 \\ 2 \end{bmatrix}$$

Now we are interested in solving the smaller system

$$\begin{bmatrix} 4 & 1 & 9 & 7 & 7 & 8 & 5 \\ 7 & 6 & 8 & 3 & 1 & 0 & 10 \\ 9 & 6 & 3 & 7 & 0 & 6 & 2 \\ 7 & 9 & 0 & 4 & 6 & 4 & 8 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_{12} \\ y_{13} \\ y_{23} \\ y_{123} \end{bmatrix} = \begin{bmatrix} 9 \\ 8 \\ 1 \\ 2 \end{bmatrix}$$

We solve this system by exhausting the values of the r original variables, then computing the corresponding values for each of the ϕ linearized variables. There are two solutions to the smaller system: $\{x_1 = 2, x_2 = 3, x_3 = 6\}$ and $\{x_1 = 3, x_2 = 8, x_3 = 6\}$.

From the reduced XL system, we have $2x_4 + 8x_1 + 4x_2 + 7x_3 + 6x_1x_2 + x_1x_3 + 2x_2x_3 + 9x_1x_2x_3 \equiv 3 \bmod 11$. The first solution gives $x_4 = 10$, which is a solution to the original non-linear system. The second solution gives $x_4 = 6$, which is *not* a solution to the original non-linear system. Thus the only solution is $\{x_1 = 3, x_2 = 8, x_3 = 6, x_4 = 10\}$.

We expect the complexity of XL' to be about the same as XL with the same D; the power of XL' is that we can solve systems with a smaller D than with XL. For example if a certain system require $D = 6$ with XL, but it can be solved with $D = 5$ with XL' by guessing r variables, we expect that $\binom{n}{6}^\omega > \binom{n}{5}^\omega + q^r$.

4 Toyocrypt

4.1 Toyocrypt

Toyocrypt-HR1 (HR1) is a pseudo-random sequence generator based on the idea of a non-linear function applied to the state of a Galois LFSR. We will use the notation introduced in Section 2 for the LFSR. HR1 uses a 128-bit Galois LFSR with a special kind of non-linear function called a bent function [10]. The 128-bit feedback polynomial is also known as the fixed key and is always chosen to be primitive. The HR1 key also has a 128-bit vector $L(0) \neq \vec{0}$, known as the stream key.[8] Although the key size appears to be 256 bits, the requirement that the feedback function be primitive reduces the key size to 248 bits because there are only 2^{120} primitive polynomials of degree 128 in $GF(2)[x]$ [9]. Note that when the feedback function is primitive, the LFSR will achieve its maximum period of $2^{128}-1$ [11]. This means that a given internal state will occur exactly once every $2^{128}-1$ clocks, so for any imaginable application we would not be able to exploit repetition in the pseudo-random key stream (before or after the non-linear function is applied).

Although the specification of HR1 does allow the feedback polynomial (fixed key) to change, the cost of finding a new primitive polynomial is prohibitive in a software implementation and most hardware implementations use a constant feedback polynomial for increased speed. In any event we assume for the purposes of cryptanalysis that the fixed key is known. If it is not published for a specific implementation, it may be obtained by looking at a software implementation or through reverse-engineering a hardware implementation [9].

Knowing the fixed key further reduces the effective key size from 248 bits to 128 bits (the length of the stream key), but this does not in itself lead to an attack on HR1 because 128 bits is still to large too brute force at this time.

4.1.1 Non-linear filter

As shown in the example in Section 2.3, it is not sufficient to use a LFSR as a keystream generator for cryptographic purposes because the key can be easily recovered [18, p. 46]. Rather than using a single state bit as the keystream bit (a linear function), HR1 generates keystream bits using a non-linear function of the LFSR state. HR1 uses a non-linear function

$$f : GF(2)^{127} \mapsto GF(2)$$

The pseudo-random keystream bits are generated by composing the non-linear function f with the state L. To prevent an attack when the initial stream key is not sufficiently pseudo-random, the first keybit is generated after the LFSR has been clocked 256 times [17], that is

$$k_t = f(L(t+256)), \quad t \geq 0 \tag{1}$$

We will always refer to the initial fill of the LFSR when $t = 0$ because we can recover the $t = 0$ state given the state at $t = 256$ as shown in Section 2.3. Note that we may not know when the keystream began, but we may choose an arbitrary point to call $t = 0$, and it will make no difference in the analysis.

HR1 designers chose f to be resistant against all known attacks. In particular, the Toyocrypt non-linear function is balanced, has high degree, and has optimal resistance to correlation attacks. The general form of the Toyocrypt nonlinear function is

$$f(x_0, x_1, ..., x_{127}) = x_{127} + \sum_{i=0}^{62} x_i x_{\alpha_i} + x_{10}x_{23}x_{32}x_{42} +$$
$$x_1x_2x_9x_{12}x_{18}x_{20}x_{23}x_{25}x_{26}x_{28}x_{33}x_{38}x_{41}x_{42}x_{51}x_{53}x_{59} + \prod_{i=0}^{62} x_i$$

[8]Recall that since the leading coefficient must be 1, we drop it from the vector form of the polynomial, resulting in 128 coefficients that can be varied to form a fixed key. $L(0) \neq \vec{0}$ avoids the degenerate case for the LFSR.

where $\{\alpha_0, \alpha_1, ..., \alpha_{62}\} \sim \{63, 64, ..., 125\}$, that is, the α_i's are a permutation of the second set.

As we will see, adding a non-linear function complicates the process of recovering the initial state of the LFSR, but inversion is still possible if the non-linear function is not carefully chosen.

The encryption algorithm (Toyocrypt-HS1) works by computing the xor of the keystream (the sequence of output bits of the non-linear function) and the plaintext.

5 Attack on Toyocrypt

We will explain the attack for a generalization of Toyocrypt which is relevant to many stream ciphers. We will assume that the stream cipher uses an n bit LFSR combined with a general non-linear filter, not necessarily of the form above. The attack works in two cases:

S1 when the non-linear filter has a low algebraic degree d

S2 when the non-linear filter can be approximated with good probability $(1 - \epsilon)$ by a function g that has low algebraic degree d

Toyocrypt falls into **S2** because the Toyocrypt non-linear function has high algebraic degree, but there are only two high degree terms. If we drop the degree 63 term and the degree 17 term, we are left with a degree 4 approximation. This approximation is correct unless exactly one of the high-degree terms is non-zero. Since the degree 17 term is fixed in the Toyocrypt non-linear function, the probability that all 17 variables are set is 2^{-17}. Similarly, the probability that the degree 63 term will affect the output of the non-linear function is 2^{-63}. If both high degree terms are non-zero, they cancel each other and the approximation is still correct. The Toyocrypt non-linear function has the interesting property that the degree 63 term contains all of the variables in the degree 17 term. Any time the degree 63 term is non-zero, the degree 17 term is also non-zero, and the two high degree terms cancel. Thus the approximation is wrong only when all 17 variables in the degree 17 term are set, but the other 46 variables in the degree 63 term are not all set. Thus the degree four approximation is correct with probability $(1 - 2^{-17} + 2^{-63}) \approx (1 - 2^{-17})$. It is also worth considering a degree 2 or degree 17 approximation, but the feasibility of this attack using those approximations is not known.

5.1 Obtaining the Known Keystream

As in Section 2.3, the attack on Toyocrypt is a known-plaintext attack. In this section we explore the plausibility of the assumptions of this attack. The attack will require a large number of known keystream bits. It is sufficient to know plaintext bits since we may simply xor the known plaintext with the ciphertext to obtain the corresponding keystream bit. One reason an attacker may know plaintext bits is because of constant/predictable header data on a message. For the purposes of the attack described below, it is not necessary for the known keystream bits to be consecutive, so we may be able to exploit the fact that the message is written with Latin characters [7]. This is due to the fact that common Latin characters (anything you can type with a standard keyboard) is represented by ASCII 0 to 127, meaning that the most significant bit in a character byte will likely be 0. This allows us to determine every 8^{th} plaintext bit. A final scenario where we might be able to obtain enough information to launch an attack is if the message is encrypted with parity and we know the scheme being used [7]. This will be explained more when we describe how to create equations from the known keystream bits in Section 5.2.

5.2 Attack Using XL

For extreme simplicity, we will work with a $n = 4$ example using the non-linear function

$$f(x_0, x_1, x_2, x_3) = x_3 + x_0x_1 + x_1x_2 + x_0x_1x_2x_3$$

which we will approximate by

$$g(x_0, x_1, x_2, x_3) = x_3 + x_0x_1 + x_1x_2$$

We assume that we somehow determined m keybits and that for simplicity, they are consecutive. To simplify notation we will ignore the Toyocrypt requirement to clock the LFSR before generating keybits. We index the known keybits $k_1, k_2, ...k_m$. We assume that we do not know any of the initial bits of the LFSR. It makes no difference if we start the LFSR from its initial state, or after it has been clocked $2n$ times (per the Toyocrypt algorithm). If we label the initial LFSR bits x_0, x_1, ..., x_{n-1}, we can compute the algebraic composition of the LFSR bits at time t in terms of the unknown bits at time $t = 0$, in the same way as Section 2.3.

For the purposes of this example, suppose we know $k_1 = 0$ and we can determine based on the known fixed key 0011

$$\begin{aligned} L_0(1) &= L_3(0) \\ L_1(1) &= L_0(0) + L_3(0) \\ L_2(1) &= L_1(0) \\ L_3(1) &= L_2(0) \end{aligned}$$

Using the same method as the example in Section 2.3, $k_t = f(L(t))$ can be found by using the algebraic representations of $L_k(t)$. To simplify notation, we will write $L_i(0) = x_i$.

$0 = k_1 = f(L(1)) \approx g(L(1)) = g(L_0(1), L_1(1), L_2(1), L_3(1)) = g(x_3, x_0 + x_3, x_1, x_2) = x_2 + x_3(x_0 + x_3) + (x_0 + x_3)x_1 = x_2 + x_3 + x_0x_3 + x_0x_1 + x_1x_3$

This equation becomes one of the original equations that will form the XL system of m original equations in n original variables. We showed in Section 3.2.3 how to determine m given D, the XL parameter. Since each known keybit leads to one original equation, we need m known keybits to solve the XL system. When we can solve this system using XL (or XL'), we have successfully found the initial fill of the LFSR. This allows us to generate the original keystream, which we can xor with the known ciphertext to recover the plaintext.

One serious problem with this method is that we are using an approximation to the non-linear function. When the original equation is incorrect, it will generate even more incorrect equations in the XL system, so we expect the overdetermined system will be inconsistent. In this case, we select m new keybits (or use a better approximation) and try again.

We can also now see how to use a known linear parity function to compute known keybits. Suppose we know the message is encoded with even parity (before encryption), so that the last bit in every plaintext byte is a parity bit that ensures an even number of ones in the byte. Let c_i be the ciphertext bits and p_i be the plaintext bits. We know the following equations hold:

$$\begin{aligned}
c_0 &= k_0 + p_0 \\
c_1 &= k_1 + p_1 \\
c_2 &= k_2 + p_2 \\
c_3 &= k_3 + p_3 \\
c_4 &= k_4 + p_4 \\
c_5 &= k_5 + p_5 \\
c_6 &= k_6 + p_6 \\
c_7 &= k_7 + p_0 + p_1 + p_2 + p_3 + p_4 + p_5 + p_6
\end{aligned}$$

If we xor these equations together, we find

$$\begin{aligned}
&c_0 + c_1 + c_2 + c_3 + c_4 + c_5 + c_6 + c_7 \\
=&k_0 + k_1 + k_2 + k_3 + k_4 + k_5 + k_6 + k_7 \\
=&g(L(0)) + g(L(1)) + g(L(2)) + g(L(3)) + g(L(4)) + g(L(5)) + g(L(6)) + g(L(7))
\end{aligned}$$

The ciphertext bits are assumed to be known and the sum of $g(L(t))$ has the same degree as each individual $g(L(t))$. Thus the equation is of the same form as described above when we knew the value of a key bit explicitly.

5.3 Analysis of Toyocrypt Attack

The approximation g is used to determine each of the m original equations of the XL system, and we assume it fails with probability ϵ. Since the states of the LFSR that determine the equations are linearly related, we probably cannot treat each equation as an independent event so the probability that the original system is correct is bounded below by $(1-\epsilon)^m$. If we repeat the attack $(1-\epsilon)^{-m}$ times with different equations, we expect to succeed. In this case the complexity of XL is about $(1-\epsilon)^{-m} \cdot \frac{7}{64} \cdot (\sum_{i=0}^{D} \binom{n}{i})^{\log_2 7}$. If we use fewer original equations, our chance of success with each attempt is improved, but we must use a higher D so that each attempt is more expensive. We explore this tradeoff below.

5.3.1 XL Analysis

Per Conjecture 3.2 when $D \in \{d...2d-1\}$ we expect the XL system to have full rank when R exceeds T by a small amount. Thus we need $R > T$ [9], which implies $m > \frac{\sum_{i=0}^{D} \binom{n}{i}}{\sum_{i=0}^{D-d} \binom{n}{i}}$. When $D \in \{2d...3d-1\}$, we need $I = R - [\binom{m}{1} + \binom{m}{2}] \cdot \sum_{i=0}^{D-2d} \binom{n}{i} \geq T-1$. Rearranging, we need m to be a solution

[9]This condition actually gives us two more equations than we need since the number of monomials is $T-1$. Thus we only need $I = T-1$, but if we require $R > T$, there can be two linear dependencies among the R equations and we can still succeed. In practice, we will choose m to be larger by a few than this analysis predicts and this should give us enough linearly independent equations.

to $\left(-\frac{1}{2}\sum_{i=0}^{D-2d}\binom{n}{i}\right)m^2 + \left(\sum_{i=0}^{D-d}\binom{n}{i} - \frac{1}{2}\sum_{i=0}^{D-2d}\binom{n}{i}\right)m + \left(1 - \sum_{i=0}^{D}\binom{n}{i}\right) \geq 0$. The (+) solution of the quadratic corresponds to the smaller value of m because the coefficient of m^2 is negative. If m is smaller than the (+) solution, we do not have enough linearly independent equations for XL to succeed. When m is larger than the (−) solution, linear dependencies accumulate to make $I < T - 1$. Thus we need m between the (+) and (−) solutions for XL to succeed. We will be interested in the smaller of these two values (the (+) solution) since for a fixed D, we want to make m as small as possible. Otherwise we are decreasing our chance of success without any reduction in complexity (in fact the complexity increases when D is fixed and we increase m due to the term $(1-\epsilon)^{-m}$).

The table below uses these formulas to find the smallest m for which XL succeeds on the Toyocrypt system for several values of D (recall $d = 4$, $n = 128$). The Data row in the table corresponds to the number of original equations needed for XL to succeed with a given D. The Memory row is the size of the final XL system represented as a bit matrix. We need about $RT \approx T^2$ bits to store the XL system. The complexity calculation is based on the result for Strassen's algorithm [16] in Section 3.2.2, modified to include the approximation term. Thus the complexity for the Toyocrypt attack is $(1 - 2^{-17})^{-m} \cdot \frac{7}{64} \cdot (\sum_{i=0}^{D}\binom{n}{i})^{\log_2 7}$.

D	4	5	6	7	8	9	10	11
Data	2^{23}	2^{21}	2^{19}	2^{18}	2^{17}	2^{16}	2^{15}	2^{14}
Memory	2^{46}	2^{56}	2^{65}	2^{73}	2^{81}	2^{88}	2^{96}	2^{102}
Complexity	2^{183}	2^{99}	2^{95}	2^{102}	2^{112}	2^{122}	2^{131}	2^{141}

Table 2: Conjectured Complexity of Toyocrypt Attack Using XL

This table is essentially the same as the table in Section 6 of [7], but it is interesting for several reasons. First, this table was produced using a different method than the table in [7]. In that paper, the authors used a fixed ratio and chose m so that $R > 1.1T$. Our approach relies on Conjecture 3.2, and so we obtain a tighter bound. This bound is noticeable in the complexity results for $D = 4, 5, 6$. Our complexity bounds are lower in these three cases. It seems that for small D, the fixed ratio overestimates the number of original equations needed (this is exactly what the authors of [7] were relying on when they chose a fixed ratio). The results in this table indicate the best attack on Toyocrypt using the method described above requires 84 KB of known keystream bits, 3 EB of memory, and has a complexity of 2^{95}, which is better than brute force. Alternatively, we can plot the complexity of the attack for several different values of D.

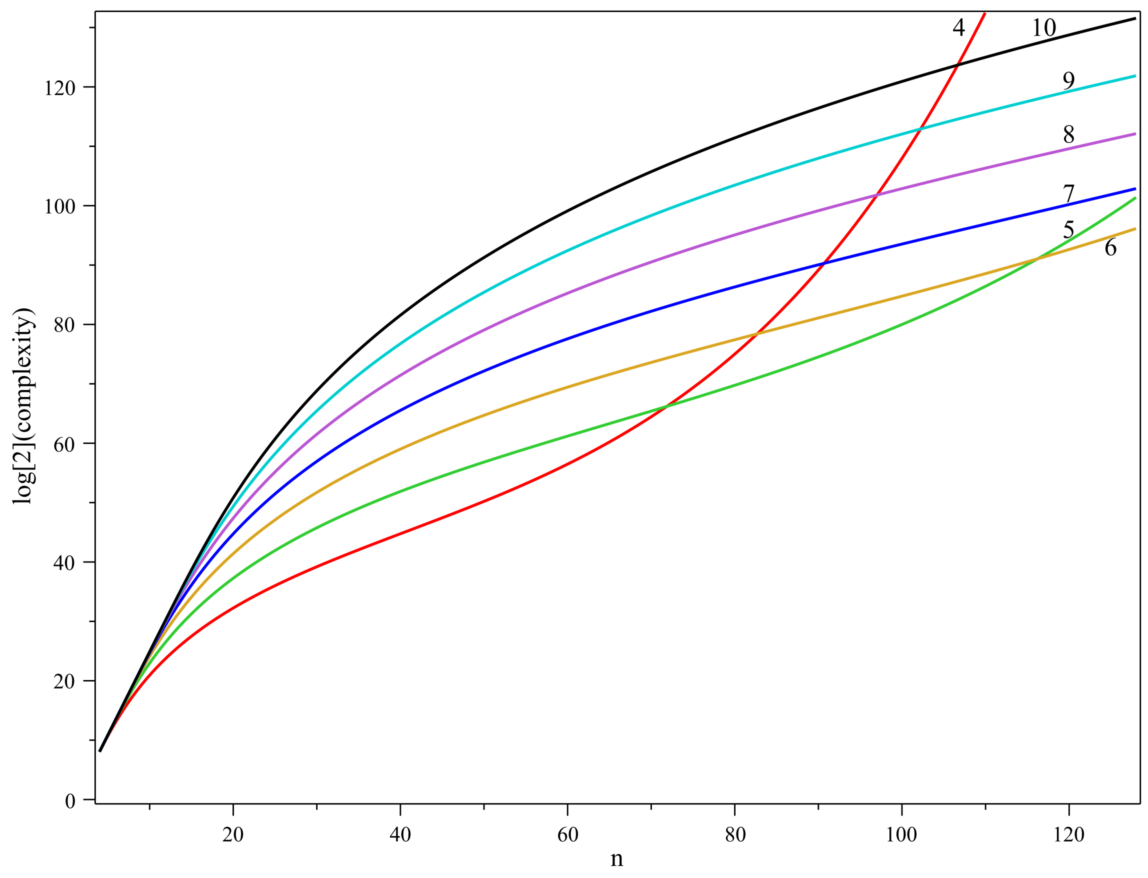

Figure 2: Conjectured Complexity of XL for D=4..10

5.3.2 XL' Analysis

Now we analyze the Toyocrypt attack using XL', which to our knowledge has not been done before. As in Section 3.2.5, let $r \leq 40$ be the number of variables guessed in the XL' attack and $\phi = \sum_{i=0}^{D} \binom{r}{i}$ be the number of monomials of degree $\leq D$ in r variables. Since we now need $T - I \geq \phi - r$, we have for $D \in \{d...2d-1\}$ that $m \geq \dfrac{T - \phi + r}{\sum_{i=0}^{D-d} \binom{n}{i}}$. For $D \in \{2d...3d-1\}$ only the constant term in the quadratic changes, so we need m to be a solution of $(-\frac{1}{2} \sum_{i=0}^{D-2d} \binom{n}{i})m^2 + (\sum_{i=0}^{D-d} \binom{n}{i} - \frac{1}{2} \sum_{i=0}^{D-2d} \binom{n}{i})m + (\sum_{i=0}^{D} \binom{r}{i} - r - \sum_{i=0}^{D} \binom{n}{i}) \geq 0$ (note that the 1 from the previous equation is now part of ϕ.) We assume that $r = 40$ because this will minimize the number of equations needed. If any of the complexities falls below 2^{40} we would need to reconsider this choice, because at that point the exhaustive search would be more costly than the Gaussian reduction. As the table below shows, the complexity is much higher in all cases so we are justified in using $r = 40$. Based on these equations, we can find the smallest value of m such that XL' succeeds for several values of D.

D	4	5	6	7	8	9	10	11
Data	2^{23}	2^{21}	2^{19}	2^{18}	2^{17}	2^{16}	2^{15}	2^{14}
Memory	2^{46}	2^{56}	2^{65}	2^{73}	2^{81}	2^{88}	2^{96}	2^{102}
Complexity	2^{182}	2^{99}	2^{95}	2^{102}	2^{112}	2^{122}	2^{131}	2^{141}

Table 3: Conjectured Complexity of Toyocrypt Attack Using XL'

This table is identical to the XL table (except for the complexity when $D = 4$.) The number of required equations is in fact smaller for XL' than XL for every value of D in the table, but not by any appreciable amount. If we view the memory requirement as T^2 rather than RT, the memory requirement does not change since T depends only on D. Corresponding to the slightly smaller values of m, the complexity is better in each case, but not appreciably.

These results are very disappointing in light of the apparent improvement of XL' over XL in some cases. There is a subtle difference between the previous analysis of XL' and this analysis. Previously, we assumed we had a fixed number of known equations. This meant D was bounded below based on the number of known equations (i.e. we could not reduce D just by adding more original equations). Thus XL' allowed us to decrease D by a small amount, which drastically reduces the complexity of the attack on a particular system. With the Toyocrypt attack, we assume that there are arbitrarily many key bits available. Given D, XL' allows us to use a smaller number of original equations. Unfortunately, this does not decrease the complexity of the attack by nearly as much as decreasing D. One reason XL' performs worse than expected is that although $\phi \approx \binom{40}{D}$ is quite large, it is still small compared to the total number of monomials, $T \approx \binom{128}{D}$. Thus the savings are relatively small for the Toyocrypt attack.

5.3.3 Other Attacks on Toyocrypt

There are other attacks on Toyocrypt. In particular, the attack from [14] is sometimes faster than the attack described above, but it requires 32 consecutive known keybits, whereas our attack has no adjacency requirements. An even better attack given in [6] breaks Toyocrypt in 2^{49} CPU clocks, but we did not investigate this attack.

5.4 When to use XL

Given our analysis, we now consider the cases where XL is useful. As shown in Table 2, if some of the original equations may be incorrect, XL provides a tradeoff between low probability of success and high complexity. In particular, a larger D allows us to use fewer original equations which increases the probability of success for each attack, but increases the cost of each attack. Thus even if we have enough equations to linearize directly without XL, the presence of the approximation error makes XL useful for reducing the overall complexity.

For the Toyocrypt attack, we assume we have access to a large number of keybits. If this is not the case, our choice of D may be bounded below by the number of original equations. Since the number of original equations increases as D decreases, linearization may not succeed (if we do not have sufficient equations to allow $D = d$), but XL can still succeed. If we are in the case of **S1** above, we would have access to a large number of equations that are known to be correct (since we do not use a non-linear approximation). In this case, we can categorically conclude that we should use linearization rather than XL. We will not have to repeat the attack since the equations are correct, and the complexity of XL (when we do not have to repeat the attack) depends on D but not m. Thus we should use enough original equations to allow us to achieve the smallest possible D, which is when $D = d$. This requires about $m = \binom{n+1}{2} + n$ original equations.

6 Toycrypt

Due to the high complexity of the Toyocrypt attack, we created a toy cipher analogous to Toyocrypt called Toycrypt. We implemented Toycrypt and the attack using C and the M4RI library for matrix algebra over $GF(2)$ [1].

6.1 Toycrypt Details

Toycrypt uses an n-bit Galois shift register ($n \geq 10$), which is clocked $2n$ times before keybits are generated, the same as Toyocrypt. The non-linear function has the same form as the Toyocrypt non-linear filter and has degree γ:

$$f(x_0, x_1, ..., x_{n-1}) = x_{n-1} + \sum_{i=0}^{n/2-1} x_i x_{\alpha_{j_i}} + x_{\lfloor 10n/128 \rfloor} x_{\lfloor 23n/128 \rfloor} x_{\lfloor 32n/128 \rfloor} x_{\lfloor 42n/128 \rfloor} + \prod_{i=s}^{s+\gamma-1} x_i$$

where α_{j_i} is the i^{th} α_j that is $< n$ and the ordering is the same as the α_j's defined for Toyocrypt.

In the Toyocrypt filter function, $\gamma = 63$, $s = 0$, and $n = 128$. There is also an additional degree 17 term in Toyocrypt. The spirit of the Toycrypt non-linear filter is that there is a degree γ term that gets dropped in the approximation, and we can adjust γ to get more or fewer approximation errors. The additional parameter s in the Toycrypt function allows us to observe any changes resulting from shifting the highest degree term.

Otherwise, Toycrypt works the same as Toyocrypt, and the attack proceeds in the same way, using the approximation

$$g(x_0, x_1, ..., x_{n-1}) = x_{n-1} + \sum_{i=0}^{n/2-1} x_i x_{\alpha_{j_i}} + x_{\lfloor 10n/128 \rfloor} x_{\lfloor 23n/128 \rfloor} x_{\lfloor 32n/128 \rfloor} x_{\lfloor 42n/128 \rfloor}$$

We were able to break Toycrypt when $10 \leq n \leq 24$. We could have broken Toycrypt for slightly larger n, but due to the amount of time needed, we did not. For n in this range, Figure 2 shows that the minimum complexity is achieved when $D = 4$, which is normal linearization. In order to study XL, we used $D = 5$ instead. Although this does not result in optimal complexity, it is still good enough and it allows us to observe XL in action.

6.2 Performance and Results

The figure below shows the time it took to break n-bit Toycrypt on a logarithmic scale. As the graph indicates, it appears that XL is fully exponential for the cases we observed.

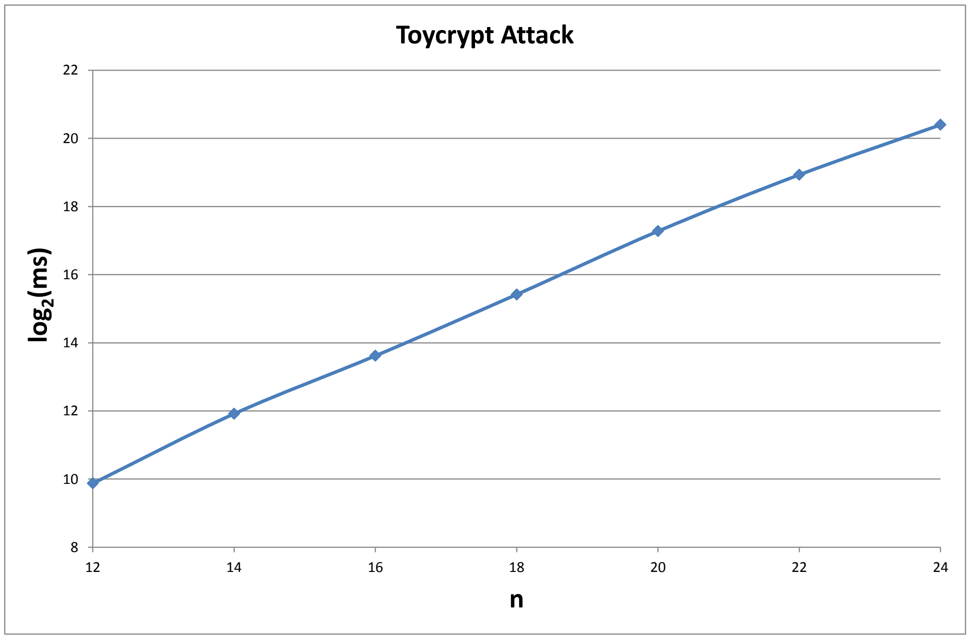

Figure 3: Runtime for Toycrypt Implementation

We observed some interesting and unexpected behavior with our implementation. Although the XL analysis over $GF(2)$ predicts that we need $I = T - 1$, we were able to break Toycrypt when $I < T - 1$ and even when $R < T$. Since the implementation generates the keystream along with the equations, we are able to detect when approximation errors occur. We expect that when approximation errors occur, the XL system will be inconsistent or at the very least the solution will be incorrect. However, we observed that even when approximation errors occurred, we were still able to solve the XL system correctly. This seemed to depend upon which high-degree term was used. We observed different behavior for different values of s.

For all of these examples, we used $D = 5$ as previously noted, $n = 16$, and $\gamma = 8$. We ran 20 trials for different values of m using a single fixed key and different stream keys. We chose our known key bits consecutively for each trial, but each trial used a different set of known key bits. Recall that for $D = 5$, we do not expect any linear dependencies. We first tried $m = 446$ and $s = 0$. This leads to $T = 6885$ and $R = 7582$, so we expect XL to succeed. However, after Gaussian elimination the rank of the XL system (I) was only 6494, which means we do not expect to be able to solve the system. Yet XL succeeded in isolating all of the linear variables, and so the attack worked even though $I < T - 1$. Of the 20 trials, 18 had errors but none of these trials led to an inconsistent system and the correct solution was obtained in every case.

Next we tried decreasing m further to 379. Now $R = 6443 < T$, so again we do not expect to succeed. After Gaussian elimination we found that $I = 5623$, so the linearized XL system was even more underdetermined. Again, all 20 trials resulted in correct answers, even though 14 trials had approximation errors.

Next we tried increasing m to 491. This gives $R = 8374 > T$, so we expect XL to succeed. For this value of m, we found that systems without approximation errors had full rank ($I = 6884$) and were consistent. However, systems that had incorrect original equations had rank $I = 6885 = T$, and were inconsistent. This is exactly the behavior we predicted, but we had to have a very large m to achieve it.

For the next set of trials, we changed to $s = 6$ and repeated some of the experiments above. When $m = 446$, we found that unlike for $s = 0$, every system with errors was inconsistent (and had rank $I = 6495$) while every system without approximation errors had rank $I = 6494$ and found the correct solution. When $m = 379$, again our results were different from the $s = 0$ trials. Every system with approximation errors had rank $I = 5264$ and were inconsistent, while systems without errors had rank $I = 5263$ and found the correct solution.

It is strange that for a fixed m, every consistent system had the same rank. It is also surprising that there were so many linear dependencies where we predicted none. As to how we could find a solution when $I < T - 1$, the results seem to indicate that the XL system has unexpected block structure. We were able to confirm this for an $n = 6$ XL system. Even though there were several free variables after Gaussian reduction, most of the variables (and all of the variables corresponding to linear monomials) did not depend on these free variables. While this explains why we were able to solve the system in these cases, it is not clear why this unexpected block structure arises.

There are still more questions about these results. It is unclear how systems with incorrect original equations are able to lead to the correct solution, since we expected such systems to be incorrect or inconsistent. We also note that while the solution depends only on the values of the linear variables, all of the non-linear terms that could be determined (i.e. that did not depend on free variables) were also correct when systems with errors led to correct final solutions.

We obtained similar results for several different values of n, m, s, and γ. Perhaps one of the most surprising results was how the consistency of the XL system depended on s, which does not play a role in the XL equations at all, since that term is dropped in the approximation.

Much more analysis is needed to determine the exact cause of this behavior. Of particular interest is the nature of the block structure in the XL system. All of the empirical results from this section need to be more closely analyzed; this would be an excellent topic for future work.

7 Conclusion

In this paper we first studied the linearization and relinearization techniques of Kipnis and Shamir and determined some linear dependencies among the relinearization equations. These dependencies reduce the efficiency of relinearization, but the technique is useful for understanding the basic approach taken in more sophisticated algorithms.

Next we studied XL, along with two variations. XL seems to be fully exponential unless the number of original equations is $m > \epsilon n^2$. We analyzed XL over both large and small fields and noticed different behavior due to equations of the form $x_i^q = x_i$. Although XL' appears to offer an improvement over XL, it was not useful in reducing the complexity of the Toy(o)crypt attack.

We introduced Toyocrypt, a stream cipher that was resistant to all known attacks at the time of its design [7]. We described an attack that reduced the problem of recovering the stream key to solving a system of non-linear equations. The attack by Courtois exploits the small number of high-degree terms in the non-linear filter. The best attack we described required 2^{95} CPU clocks and required 2^{65} bits of memory. We offered additional analysis using XL' and found no improvement in either time or space complexity.

Finally we described our implementation, which is the first that we are aware of. Our trials confirmed that the attack is fully exponential but led to several interesting and unexplained anomalies. It appears that the XL system contains a previously undocumented block structure, which allows us to succeed even when $R < T$. Even more interesting is that we can succeed in cases where approximation errors occur. This seems to depend on the specific non-linear function being used. Due to the large complexity of the attack on the full Toyocrypt system, we were unable to test the actual Toyocrypt non-linear function. The implementation results leave many open questions that would be an excellent topic for future work.

7.1 Acknowledgements

I want to thank Dr. Joshua Holden for his insight, support, and time throughout this project. I would also like to thank the Japan's IPA Crypto Research Group for providing me with the Toyocrypt-HR1 and Toyocrypt-HS1 specifications, which were crucial for the implementation.

References

[1] Martin Albrecht and Gregory Bard. *The M4RI Library – Version 20100817*. The M4RI Team, 2010.

[2] Gregory V. Bard. *Algebraic Cryptanalysis*. Springer, 1st edition, 2009.

[3] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 1–6, New York, NY, USA, 1987. ACM.

[4] Nicolas Courtois, Louis Goubin, Willi Meier, and Jean-Daniel Tacier. Solving Underdefined Systems of Multivariate Quadratic Equations. In David Naccache and Pascal Paillier, editors, *Public Key Cryptography*, volume 2274 of *Lecture Notes in Computer Science*, pages 211–227. Springer Berlin / Heidelberg, 2002.

[5] Nicolas Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir. Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations. In *Proceedings of the 19th International Conference on the Theory and Applications of Cryptographic Techniques*, EUROCRYPT'00, pages 392–407. Springer-Verlag, 2000.

[6] Nicolas Courtois and Willi Meier. Fast Algebraic Attacks on Stream Ciphers with Linear Feedback. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 176–194. Springer Berlin / Heidelberg, 2003.

[7] Nicolas T. Courtois. Higher Order Correlation Attacks, XL Algorithm and Cryptanalysis of Toyocrypt. In *Proceedings of the 5th International Conference on Information Security and Cryptology*, ICISC'02, pages 182–199. Springer-Verlag, 2003.

[8] Nicolas T. Courtois and Jacques Patarin. About the XL Algorithm over GF(2). In *Proceedings of the 2003 RSA Conference on the Cryptographers' Track*, CT-RSA'03, pages 141–157. Springer-Verlag, 2003.

[9] Ed Dawson, Andrew Clark, Helen Gustafson, Bill Millan, and Leonie Simpson. Evaluation of Toyocrypt-HS1. Technical report, Queensland University of Technology, January 2001.

[10] Hans Dobbertin and Gregor Leander. A Survey of Some Recent Results on Bent Functions. In Tor Helleseth, Dilip Sarwate, Hong-Yeop Song, and Kyeongcheol Yang, editors, *Sequences and Their Applications - SETA 2004*, volume 3486 of *Lecture Notes in Computer Science*, pages 129–147. Springer Berlin / Heidelberg, 2005.

[11] S.W. Golomb. *Shift Register Sequences*. Aegean Park Press, 1982.

[12] Aviad Kipnis and Adi Shamir. Cryptanalysis of the HFE Public Key Cryptosystem by Relinearization. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '99, pages 19–30. Springer-Verlag, 1999.

[13] R. Lidl and H. Niederreiter. *Introduction to Finite Fields and Their Applications*. Cambridge University Press, 1986.

[14] M. Mihaljevic and H. Imai. Cryptanalys of Toyocrypt-HS1 Stream Cipher. In *IEICE Transactions on Fundamentals*, volume E85-A, pages 66–73. Jan. 2003.

[15] Claude E. Shannon. Communication Theory of Secrecy Systems. *Bell Systems Technical Journal*, 28:656–715, 1949.

[16] Volker Strassen. Gaussian Elimination is Not Optimal. *Numerische Mathematik*, 13:354–356, 1969. 10.1007/BF02165411.

[17] Koichi Sugimoto. *Cryptographic Techniques Specifications Toyocrypt-HR1*, October 2000.

[18] Lawrence C. Washington and Wade Trappe. *Introduction to Cryptography: With Coding Theory*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2002.

A Degree Six Relinearization

The following randomly chosen homogeneous system in five variables can be solved using degree four relinearization:

$$\begin{bmatrix} 2 & 3 & 10 & 1 & 4 & 3 & 0 & 9 & 1 & 10 & 6 & 9 & 8 & 6 & 3 \\ 6 & 8 & 8 & 10 & 9 & 0 & 3 & 9 & 1 & 0 & 3 & 7 & 9 & 9 & 8 \\ 0 & 5 & 2 & 7 & 3 & 7 & 7 & 0 & 2 & 6 & 10 & 10 & 0 & 6 & 7 \\ 6 & 1 & 6 & 5 & 6 & 10 & 9 & 0 & 10 & 6 & 2 & 3 & 9 & 2 & 9 \\ 3 & 4 & 3 & 6 & 2 & 10 & 10 & 4 & 1 & 1 & 1 & 0 & 0 & 9 & 3 \\ 8 & 8 & 7 & 0 & 2 & 2 & 5 & 3 & 7 & 10 & 6 & 6 & 5 & 4 & 2 \\ 1 & 9 & 1 & 7 & 1 & 5 & 2 & 8 & 7 & 3 & 8 & 9 & 9 & 1 & 7 \end{bmatrix} \begin{bmatrix} x_1^2 \\ x_1x_2 \\ x_1x_3 \\ x_1x_4 \\ x_1x_5 \\ x_2^2 \\ x_2x_3 \\ x_2x_4 \\ x_2x_5 \\ x_3^2 \\ x_3x_4 \\ x_3x_5 \\ x_4^2 \\ x_4x_5 \\ x_5^2 \end{bmatrix} = \begin{bmatrix} 9 \\ 8 \\ 0 \\ 1 \\ 7 \\ 4 \\ 8 \end{bmatrix}$$

One solution to this system is

$$\begin{bmatrix} x_1^2 = 5 \\ x_1x_2 = 9 \\ x_1x_3 = 5 \\ x_1x_4 = 6 \\ x_1x_5 = 10 \\ x_2^2 = 3 \\ x_2x_3 = 9 \\ x_2x_4 = 2 \\ x_2x_5 = 7 \\ x_3^2 = 5 \\ x_3x_4 = 6 \\ x_3x_5 = 10 \\ x_4^2 = 5 \\ x_4x_5 = 1 \\ x_5^2 = 9 \end{bmatrix}$$

which gives $x_1 \equiv \pm 4 \bmod 11$, $x_2 \equiv \pm 5 \bmod 11$, $x_3 \equiv \pm 4 \bmod 11$, $x_4 \equiv \pm 4 \bmod 11$, and $x_5 \equiv \pm 3 \bmod 11$. The solutions to the non-linear system are $\{x_1 = 4,\ x_2 = 5,\ x_3 = 4,\ x_4 = 7,\ x_5 = 8\}$ and $\{x_1 = 7,\ x_2 = 6,\ x_3 = 7,\ x_4 = 4,\ x_5 = 3\}$.

The system requires seven equations to solve as predicted in Table 1. The system above can also be solved with degree six relinearization, so at least in this case, degree six relinearization is at least as powerful as degree four relinearization.

If we remove the last equation from the system, we are unable to solve it with degree four relinearization. However, we can still solve the smaller system with degree six relinearization. This shows that degree six relinearization is more powerful than degree four relinearization in some cases.

B Random Linear Dependencies

We can gain some insight into how often "random" linear dependencies occur by using the matrix from Appendix A and changing the right hand side to

$$\begin{bmatrix} 1 \\ 10 \\ 5 \\ 7 \\ 4 \\ 1 \\ 1 \end{bmatrix}$$

(which corresponds to the same solution above, but with $x_2 = 1$).

Since the system has the same matrix, we expect to have the same number of systematic dependencies (even systematic dependencies that have not been classified). However, neither degree four nor degree six relinearization can solve the system with seven equations! This means there are "random" dependencies with the new right hand side vector that did not occur with the system in Appendix A.