Rochester Institute of Technology
RIT Digital Institutional Repository

Theses

2007

Performance evaluation of eXtended sparse linearization in GF(2) and GF(28)

Tigin Kaptanoglu

Follow this and additional works at: https://repository.rit.edu/theses

Recommended Citation
Kaptanoglu, Tigin, "Performance evaluation of eXtended sparse linearization in GF(2) and GF(28)" (2007). Thesis. Rochester Institute of Technology. Accessed from

This Thesis is brought to you for free and open access by the RIT Libraries. For more information, please contact repository@rit.edu.

M.S. Thesis Report

Performance Evaluation of eXtended Sparse Linearization in GF(2) and GF(2^8)

Tiğin Kaptanoğlu
txk4406@cs.rit.edu
January 19, 2007

Department of Computer Science
Rochester Institute of Technology
Rochester, NY 14623

Committee:

Chair	:	Prof. Stanisław Radziszowski
Reader	:	Prof. Alan Kaminsky
Observer	:	Prof. Anurag Agarwal

Contents

1 Motivation 5

2 Overview 6
- 2.1 Background 6
- 2.2 History of Block Ciphers and Attacks 7

3 The Cipher Rijndael 8
- 3.1 AddRoundKey 8
- 3.2 SubBytes 8
- 3.3 ShiftRows 9
- 3.4 MixColumns 10
- 3.5 Key Scheduling 10

4 Mathematical Preliminaries 10
- 4.1 Finite Fields 10
  - 4.1.1 GF(2) 10
  - 4.1.2 GF(2^8) 11
- 4.2 Linear Elimination 11
  - 4.2.1 Gaussian Elimination 12
  - 4.2.2 Gauss-Jordan Elimination 12
- 4.3 Multivariate Quadratic Problem 14
- 4.4 Linearization 15

5 Derivation of the Equations 16
- 5.1 The Ferguson-Schroeppel-Whiting Equations 16
- 5.2 The Cid-Murphy-Robshaw Equations 17
- 5.3 The Biryukov-De Canniere Comparison of Equations 18

6 Field Transformation 19
- 6.1 Univariate Linear Field Transformation 20
- 6.2 Multivariate Quadratic Field Transformation 20
- 6.3 Fitting the Bit-Equations Together 26

7 XL 26
- 7.1 The Algorithm 26
- 7.2 Analysis of the Algorithm 28
  - 7.2.1 Moh's Analysis of XL on TTM 28
  - 7.2.2 Diem's Analysis of XL 29

8 XSL **29**
8.1 The Algorithm 29
8.2 Analysis of the Algorithm 31
8.2.1 Analysis of XSL by C. Cid and G. Leurent 31
8.2.2 Song-Seberry Paper 31
8.2.3 Essay by B. Schneier 32
8.3 The XSL variant 32

9 Similar Research on XSL **33**
9.1 Evaluating Algebraic Attacks on AES 33
9.2 XL and XSL attacks on Baby Rijndael 33

10 Implementation of XSL **34**
10.1 What We Did 34
10.2 What We Left Out 36

11 Design of Software **37**

12 Equations **40**
12.1 Number of Equations 40
12.2 Number of Variables 40
12.3 Sparsity Ratio 40
12.4 Indexing of Variables 41

13 Main Modules **42**
13.1 Equation File Format 42
13.2 Random Equation Generator 43
13.3 Main Program 43
13.4 Equation System 44
13.5 Variables 47
13.6 Brute Force Search 47
13.7 Field Transformer 48
13.8 Exceptions 48
13.9 Tools 49
13.9.1 Add and Mult 49
13.9.2 Equation Writer 49

14 Experimental Results **50**
14.1 Charts and Figures 50
14.1.1 XSL 50
14.1.2 XSL Variant 52
14.1.3 Comparison of XSL and XSL-variant 53

14.2 Interpretation of Results 54
14.2.1 Effect of Parameters on XSL's Performance 54
14.2.2 XSL versus Variant 60

15 Room For Improvement **61**

16 Conclusion **62**

A Running the Code **67**

B XSL on a Byte-System **68**

C Transformation: Byte to Bit **70**

Abstract

XSL (eXtended Sparse Linearization) is a recent algebraic attack aimed at the Advanced Encryption Standard. In order to shed some light into the behavior of the algorithm, which is largely unknown, we have studied XSL on equation systems with variables interpreted either as bits or bytes. The algorithm solves byte-systems much faster than it does bit-systems, which promts us to suggest that if a more compact representation of equation systems can be found, such as one where the variables are 8-byte blocks, or even a more generalized form of 8^n-byte blocks, it may be possible to increase the speed of XSL dramatically.

1 Motivation

The pattern of advances in cryptography resembles a prey-predator coevolutionary cycle; a cipher is developed, it takes a while for the attacks to evolve, take shape and damage the cipher, which forces the birth of a new cipher, which in turn causes the attacks to adapt and improve, and so ad infinitum. Currently, Rijndael, chosen in 2001 as the official Advanced Encryption Standard after a process that spanned two rounds and four years, is the big cipher, the last link in the chain of evolution, the new predator that is inapproachable and resistant to all known forms of attack This alone is enough to seduce a mathematician into doing some research on Rijndael. Some experts considering it as the risky choice among the final round contenders for the Advanced Encryption Standard process, coupled with the fact that there are some attacks out there aimed at Rijndael whose behaviors are not very clear, has lured us into analyzing one such attack and trying to identify how it behaves under certain circumstances.

The eXtended Sparse Linearization (XSL) attack is relatively new, and seems to be the only predator who at least claims to be able to bite Rijndael, if not totally hunt it down. Developed in 2002 by N. Courtois and J. Pieprzyk, it is inspired by Patarin's cryptanalysis [20] of the Matsumoto-Imai public key scheme [15] with the linearization technique and more recently by A. Kipnis and A. Shamir's cryptanalysis of HFE with relinearization [12], and claims that Rijndael (and Serpent) can be cryptanalyzed in a similar fashion. However, some parts in the algorithm are vague, and since no full-size implementation exists, not much is known about the behavior of the algorithm.

Picking up from this state of affairs, we have tried to analyze the XSL algorithm; to be more specific, we have implemented the algorithm and conducted experiments on XSL by giving it an equation system in bytes, then the equivalent system as expressed in bits, and compared the performance of the algorithm for both cases in an attempt to see which expression of the same system it favors. This involved the derivation of formulae for performing the transformation of one system to the other. In addition to the bit-byte comparison, it was also possible to observe the impact of parameters such as the number of equations, the number of variables and sparsity on the speed, memory requirements and the actual functionality of the algorithm. Finally, we have also measured XSL's performance against an XSL variant which was designed with the sole purpose of changing the priority of the algorithm from speed and memory economy to being able to solve a greater number of

equation systems.

Our experiments demonstrated that the main limiting factor was the excessive usage of memory by the XSL algorithm due to its extension process. It was further observed that the algorithm behaves much faster when presented with a more compact expression of an equation system (bytes as opposed to bits). This has prompted us to suggest that a general conversion from bits to 8^n bits could be formulated and the algorithm could be applied to the most compact representation of the equation system, which could be expected to solve MQ systems even faster. We have left the application of this suggestion to later studies.

2 Overview

2.1 Background

Ever since its introduction to the cryptography scene as the official standard, the security of Rijndael has been open to debate, with the cipher consistently getting the better of its attackers and currently looking unbreakable. However, while it has been proven to be safe and secure against traditional forms of cryptanalysis (such as differential and linear cryptanalysis), its highly algebraic structure has been an attractive prospect for cryptologists to try to make use of this structure and develop methods to recover the secret key efficiently. One new genre of cryptographic attacks that aim to exploit this structure are called by a variety of names such as "algebraic attacks", "linearization attacks" and "MQ (multivariate quadratic) attacks". This form of cryptanalysis suggests that the encryption process of Rijndael, which is a bijective function that maps the plaintext to a ciphertext for a given key, can be expressed as a system of multivariate quadratic equations; thus, finding a practical way of solving these equations would yield a weakness in the cipher. However, solving such equation systems is an already-known NP-complete problem, called MQ. It was known by the designers of the cipher that the encryption process could be expressed as an MQ problem and, obviously, that expression alone is not sufficient to suggest a weakness in the cipher. What may do so, however, is an observation that the MQ systems generated by the encryption of Rijndael are in fact sparse and overdefined. This is an indication that the instances of the MQ problem that are derived out of Rijndael reside in a certain area within the problem space such that the observed characteristics may be exploited to devise a general, practical solution to this certain subset of the problem. These claims have so far remained

merely as theoretical suggestions, as of yet no full-scale implementation of these attacks is known to exist and their actual complexities and behaviors are unknown.

2.2 History of Block Ciphers and Attacks

It is possible to trace the block cipher standards back to Lucifer [9], a block cipher that was designed in the early 1970s by Horst Feistel of IBM. Lucifer never became an official encryption standard but can be said to have directly inspired the Data Enrcyption Standard (DES) [11]. It encrypted 128-bit data blocks with 128-bit keys and had operations much like those used by DES. It also defined the term "Feistel cipher", which briefly meant a block cipher that uses P-Boxes and S-Boxes to achieve the diffusion of bits.

In 1974, the National Institute of Standards and Technology (NIST or, as it was called then National Bureau of Standards (NBS)) announced a contest to select an official encryption standard to be used by the U.S. government. The winner of the competition was a cipher developed by a team from IBM. It was a Feistel cipher, a direct heir of Lucifer in fact. It encrypted data in blocks of 64 bits with 56-bit keys.

From 1976 to 1998 DES served as the official standard, during which period controversy always followed it; some suggested that the designers as well as the NSA already knew how to crack it as early as the 1970s when it was first designed. However, it wasn't until 1990 when a security opening was seriously suggested publicly when a paper on differential cryptanalysis was published by cryptologists Eli Biham and Adi Shamir [1]. However, this attack required 2^{47} known plaintexts in order to break DES and it was practically unusable. This raised questions as to whether this attack was already known to the developers of DES when it was first designed.

With the amount of research growing in parallel to public interest in cryptography and especially block ciphers, attacks directed at DES started to come thick and fast in the 1990s, with gradually more voices being raised against the use of DES for securing sensitive data. In 1994, Mitsuru Matsui published his work on linear cryptanalysis [14]; it needed 2^{43} known plaintexts in order to break DES. Although it was deemed impractical, it was an improvement in the cryptanalysis of DES and eventually a step closer to its dethronement.

Later in the 1990s the speed at which new attacks were produced in-

creased dramatically, and so did the security implications they imposed on DES. By the time when DES was broken in 1998 with a brute-force search, NIST had already announced a competition to select its successor.

After a process that saw two rounds and almost four years of debate, the number of contestants were reduced to 15, then 5, and finally Rijndael, developed by Joan Daemen and Vincent Rijmen, was announced as the Advanced Encryption Standard. It was, obviously, resistant to both differential and linear cryptanalysis, as well as to brute-force search because of its large key size (128 to 256 bits).

In the years that followed the standardization process, the cryptography world did not waste any time in attacking the new cipher with full intent. So far, Rijndael has stood firm against all known forms of attack. It has been suggested, and widely rejected but not disproven, that it may be possible to exploit its highly algebraic structure. These suggestions, formulated as algorithms and called algebraic attacks, form the basis of our work.

3 The Cipher Rijndael

In order to analyze the attack, it is first necessary to take a look at the prey.

Rijndael [7] is a block cipher with a block size of 128 bits. The recommended number of rounds depends on the level of security desired: 10 to 14 rounds of encryption may be used, with keys of length $128 + 32 * n$, n being the extra number of rounds beyond 10. One round of encryption is made up of 4 components, which are explained in the subsections that follow.

3.1 AddRoundKey

The bits of the input byte are XORed with the bits of the round key. The derivation of the round key via the key scheduling algorithm will be discussed further in this section.

3.2 SubBytes

SubBytes is a bijective function that maps each input byte to a unique output byte. The S-Box can be represented either as a look-up table with 256 entries, or as a combination of two functions. The first of these functions

is the calculation of the inverse of the input byte in GF(2^8). The result is processed through the linear function to produce the outcome of this component. The linear function is the multiplication of the byte (expressed as a bit vector) with a fixed 8-by-8 matrix and the addition of a fixed byte to the outcome.

Example with one byte of output:

Assume the input byte in hexadecimal representation is 53 (01010011 in binary and x^6+x^4+x+1 in polynomial representation). The multiplicative inverse of the input polynomial is calculated $\pmod{x^8+x^4+x^3+x+1}$, which is the irreducible polynomial used by Rijndael, in GF(2^8). The inverse is $x^7+x^6+x^3+x$ since:

$$(x^6+x^4+x+1)*(x^7+x^6+x^3+x) \equiv 1 \pmod{x^8+x^4+x^3+x+1}$$

The inverse of 53 (01010011) is CA (11001010). The next step is to put this inverse byte through the following calculation:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

The output is ED (11101101), which is the value that 53 is mapped onto by the S-Box.

3.3 ShiftRows

The state, which is a 16-byte data block, is expressed as a 4-by-4 matrix and the rows are shifted cyclically: the nth row is shifted n positions to the right.

Example:

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix} \rightarrow \begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{13} & a_{10} & a_{11} & a_{12} \\ a_{22} & a_{23} & a_{20} & a_{21} \\ a_{31} & a_{32} & a_{33} & a_{30} \end{pmatrix}$$

3.4 MixColumns

Columns of the 4-by-4 byte-matrix (the state) are multiplied $\pmod{x^4+1}$ by the polynomial $3*x^3+x^2+x+2$ over GF(2^8) and the polynomial products form the new state matrix. The ShiftRows and MixColumns phases provide linear diffusion.

$$\begin{pmatrix} FA & 55 & 02 & 27 \\ CF & A0 & 9B & 31 \\ 49 & 84 & 91 & D7 \\ A3 & 28 & 64 & 99 \end{pmatrix} \rightarrow \begin{pmatrix} 4F & FD & 47 & 53 \\ 07 & B1 & E3 & BE \\ 59 & 9E & 0C & 13 \\ CE & 8B & C4 & A6 \end{pmatrix}$$

3.5 Key Scheduling

The key scheduling algorithm derives the first round key from the original key, and the consecutive round keys from the previous round key. The original k-bit key is expanded into $(n+1)*k$ bits (n is the number of rounds), where every block of k bits is the round key for the corresponding round. The expansion process consists of a byte substitution by the use of the S-Box and a bit-by-bit XOR with the round constant.

4 Mathematical Preliminaries

Before going on to describe the attacks, it is essential to describe the finite fields we will be operating in.

4.1 Finite Fields

4.1.1 GF(2)

The arithmetic in Galois Field(2) is trivial. Addition of two bits is a simple XOR:

$$\begin{array}{ccccc} 0 & + & 0 & = & 0 \\ 0 & + & 1 & = & 1 \\ 1 & + & 0 & = & 1 \\ 1 & + & 1 & = & 0 \end{array}$$

Multiplication of two bits is a simple XAND:

$$\begin{array}{ccccc} 0 & * & 0 & = & 0 \\ 0 & * & 1 & = & 0 \\ 1 & * & 0 & = & 0 \\ 1 & * & 1 & = & 1 \end{array}$$

A bit's additive inverse is itself. The same is true for a multiplicative inverse.

4.1.2 GF(2^8)

Galois Field (256) arithmetic is more complicated than GF(2) arithmetic. Addition of two bytes is defined as a bit-by-bit XOR operation. Multiplication of two bytes is defined as follows: The two bytes are expressed as polynomials. For byte A, a binary representation of

$$a_7a_6a_5a_4a_3a_2a_1a_0$$

is equivalent to a polynomial representation of

$$a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0x^0$$

The polynomial representations of the two bytes are then multiplied (mod $x^8 + x^4 + x^3 + x + 1$), which is an irreducible polynomial of degree 8 in GF(2^8).

The additive inverse of a byte, from the definition of XOR, is the byte itself. In other words, $A + A = 0$. The multiplicative inverse A^{-1} of byte A is defined by

$$A * A^{-1} \equiv 1 \pmod{x^8 + x^4 + x^3 + x + 1}$$

Example of Multiplication:
Let us consider multiplying bytes 45 and $C1$. The polynomial representations are

$$\begin{aligned} 45 &= x^5 + x^2 + 1 \\ C1 &= x^7 + x^6 + 1 \end{aligned}$$

These polynomials are multiplied and the residue is calculated:

$$x^12 + x^11 + x^9 + x^8 + x^7 + x^6 + x^5 + x^2 + 1 \equiv x^7 + x^4 + x + 1 \pmod{x^8 + x^4 + x^3 + x + 1}$$

Thus, the result of the multiplication is found to be 93, which is represented by the polynomial $x^7 + x^4 + x + 1$.

4.2 Linear Elimination

Linear elimination is a general title for a set of techniques that may be used to evaluate linear equation systems. The system is expressed as a matrix, whose elements consist of the coefficients of the system, and several computations are applied to the matrix to yield the solution.

4.2.1 Gaussian Elimination

Gaussian elimination, found by German mathematician Carl Friedrich Gauss, is a linear elimination technique for solving linear equation systems. Rows of the coefficient matrix for the equation system are used for eliminating other rows, transforming the original matrix to a linearly equivalent state such that at least one row of the final state represents a univariate equation, provided that there are a sufficient number of linearly independent equations in the original equation system to yield such a univariate equation, and that equation can be evaluated to reveal the value of one variable. That value can then be substituted into all other equations that contain a non-zero coefficient for it, thus reducing the other rows and producing at least one more univariate equation at each backward iteration. This elimination and evaluation process is repeated until all the variables have been evaluated and the solution of the equation system is found.

An example in GF(2^8):

$$\left(\begin{array}{ccc|c} 0A & 15 & 30 & 3F \\ 05 & 0C & 3A & 09 \\ 21 & BF & 68 & 48 \end{array}\right) \rightarrow \left(\begin{array}{ccc|c} 0A & 15 & 30 & 3F \\ 00 & 8B & 22 & 9B \\ 00 & 83 & 90 & 0C \end{array}\right) \rightarrow \left(\begin{array}{ccc|c} 0A & 15 & 30 & 3F \\ 00 & 8B & 22 & 9B \\ 00 & 00 & \mathbf{50} & \mathbf{7C} \end{array}\right)$$

$$\rightarrow \left(\begin{array}{ccc|c} 0A & 15 & 30 & 3F \\ 00 & 8B & 22 & 9B \\ 00 & 00 & \mathbf{01} & \mathbf{15} \end{array}\right) \rightarrow \left(\begin{array}{ccc|c} 0A & 15 & 00 & E2 \\ 00 & \mathbf{8B} & 00 & \mathbf{DE} \\ 00 & 00 & \mathbf{01} & \mathbf{15} \end{array}\right)$$

$$\rightarrow \left(\begin{array}{ccc|c} 0A & 15 & 00 & E2 \\ 00 & \mathbf{01} & 00 & \mathbf{52} \\ 00 & 00 & \mathbf{01} & \mathbf{15} \end{array}\right) \rightarrow \left(\begin{array}{ccc|c} \mathbf{0A} & 00 & 00 & \mathbf{B4} \\ 00 & \mathbf{01} & 00 & \mathbf{52} \\ 00 & 00 & \mathbf{01} & \mathbf{15} \end{array}\right)$$

$$\rightarrow \left(\begin{array}{ccc|c} \mathbf{01} & 00 & 00 & \mathbf{12} \\ 00 & \mathbf{10} & 00 & \mathbf{52} \\ 00 & 00 & \mathbf{01} & \mathbf{15} \end{array}\right)$$

4.2.2 Gauss-Jordan Elimination

Gauss-Jordan Elimination is a variation of the Gaussian elimination. It is used to the same purpose. The difference between the two techniques is that Gauss-Jordan does the forward and backward iterations simultaneously, while Gaussian elimination first completes the forward iterations and then moves onto the backwards iterations. With the Gauss-Jordan technique, the

pivot element is used for eliminating all rows in the system, not only those that are below it.

An example in GF(2^8):

$$\left(\begin{array}{ccc|c} 0A & 15 & 30 & 3F \\ 05 & 0C & 3A & 09 \\ 21 & BF & 68 & 48 \end{array}\right) \rightarrow \left(\begin{array}{ccc|c} 0A & 15 & 30 & 3F \\ 00 & 8B & 22 & 9B \\ 00 & 83 & 90 & 0C \end{array}\right) \rightarrow \left(\begin{array}{ccc|c} 0A & 00 & 49 & 34 \\ 00 & 8B & 22 & 9B \\ 00 & 00 & \mathbf{50} & \mathbf{7C} \end{array}\right)$$

$$\rightarrow \left(\begin{array}{ccc|c} \mathbf{0A} & 00 & 00 & \mathbf{B4} \\ 00 & \mathbf{8B} & 00 & \mathbf{27} \\ 00 & 00 & \mathbf{50} & \mathbf{7C} \end{array}\right) \rightarrow \left(\begin{array}{ccc|c} \mathbf{01} & 00 & 00 & \mathbf{12} \\ 00 & \mathbf{01} & 00 & \mathbf{52} \\ 00 & 00 & \mathbf{01} & \mathbf{15} \end{array}\right)$$

We have decided to use the Gauss-Jordan technique for our implementation of XSL. The reason for this was that, unlikely as it is, Gaussian elimination might have produced only one univariate equation on the extended system which is actually multivariate on the original system.

4.3 Multivariate Quadratic Problem

MQ is a known NP-complete problem. The problem is to solve a finite set of equations of degree 2, containing multiple variables. It has several applications in both public and private key cryptography today.

Christopher Wolf has made an extensive study on applications of the MQ problem and its applications in public key cryptography [23]. Understanding the problem as related to public key cryptography is important, since that is where linearization-based attacks originated from, several years before the first linearization attack was proposed against Rijndael. Wolf's study illustrates the first instances of MQ-attacks as well as the NP-completeness of the problem.

The attack that partly inspired the XSL algorithm is the breaking of the Hidden Field Equations (HFE) algorithm in 1999 by Avaid Kipnis and Adi Shamir [19]. Although it was admitted that sufficiently large keys would make it impractical, the attack reduced the complexity of breaking HFE to polynomial, which was the optimistic claim for both XL and XSL when they were first designed. Breaking the cipher was shown to be equivalent to solving an MQ problem, but a representation was found for the MQ instances derived from the cipher such that n equations with n variables in one finite field could be represented as 1 equation with 1 variable in another finite field. The transformation made it possible to treat the system as a univariate equation, hence polynomial complexity.

4.4 Linearization

Linearization is a term used for techniques proposed for solving multivariate non-linear equation systems. It is a simple method which has inspired numerous cryptographic attacks on a wide range of ciphers in the short space of time since its conception. The gist of the idea, developed by their cryptanalysis of HFE as described in the previous section, is to find a representation of an MQ instance such that it can be treated as a pseudo-univariate or -linear equation system. Once it has been expressed as a linear system, known linear techniques such as Gaussian, Gauss-Jordan or any other elimination technique, can be applied to solve the system. The trick is to make the re-expression phase practical.

The first attempt at applying this technique came with XL [5] which is described in detail in Section 7. The algorithm aimed to form a pseudo-linear equation system with a sufficient number of linearly independent equations so that it could be solved with linear equation-solving techniques. The idea was to rename all monomials, linear and quadratic alike, with new variable names and to treat them all as linear variables. The problem was that, as the equations initially appeared, this method would create an equation system with far too many variable names compared to the number of equations. In fact, in an initially 2-equation and 2-variable MQ system, this idea would yield a 2-equation and 5-or-less-variable linear system. Assuming that 3 different monomials actually exist in the system, it is still not solvable by linearization alone. The obvious solution, then, was to generate more equations. This was done by multiplying every equation in the system with all possible monomials of degree 2. This would create an extended system of 8 equations and 8 variables, which is now solvable by linearizing and applying Gaussian elimination. Although a valid method when applied to MQ systems of certain structure, the downside of this method is that the extension process creates such a large system that the attack easily becomes impractical. The XL attack has been studied, analyzed and is today considered to be an impractical method to attack Rijndael.

XSL follows XL by two years. The new idea that it brings to the XL algorithm is to multiply the equations in the system not by every monomial, but a "carefully selected" set of monomials. This ambiguous phrase is open to interpretation and has made several variants possible. In this study, we have followed the suggestion in the initial XSL paper that only monomials that appear in other equations would be used for multiplication with a given equation.

5 Derivation of the Equations

5.1 The Ferguson-Schroeppel-Whiting Equations

The first significant attempt to express Rijndael's encryption process was published in [10] in 2001 by Niels Ferguson, Richard Schroeppel and Doug Whiting. Their MQ expression of the cipher is considered today to be impractical for all purposes, but it is a pioneering work which eventually inspired the XSL attack directly.

The analysis of the encryption process of Rijndael was used to show how to derive the equations; the equations were derived up to 5 rounds, and it was stated that following the same procedure, similar equations can be produced up to any number of rounds. The following derivation and all equations involved are taken from [10]:

In order to express one round of the cipher as a system of equations, they start from the S-Box. The S-Box transformation of Rijndael yields the following equations by definition:

$$S(x) \quad = \quad \Sigma_{d=0}^{7} w_d x^{-2^d}$$

where x is the input byte, $S(x)$ is the S-box transformation, and w_d are constants. Defining the state of the data block at any point as a, and the byte at position (i, j) at round r as $a_{i,j}^{(r)}$, the S-Box transformation at round r for state $a^{(r)}$ can be written as

$$S(a_{i,j}^{(r)}) \quad = \quad \Sigma_{d_r=0}^{7} w_{d_r} (a_{i,j}^{(r)})^{-2^d}$$

The ShiftRow stage is defined as the cyclic shifting of rows by the row number, so the combination of ShiftRow and SubBytes can be expressed as

$$t_{i,j}^{(r)} \quad = s_{i,i+j}^{(r)} \quad = \quad \Sigma_{d_r=0}^{7} w_{d_r} (a_{i,i+j}^{(r)})^{-2^{d_r}}$$

The MixColumn phase can be viewed as a simple matrix multiplication, so the combination of all 3 phases gives us:

$$\begin{aligned} m_{i,j}^{(r)} &= \Sigma_{e_r=0}^{3} v_{i,e_r} t_{e_r,j}^{(r)} \\ &= \Sigma_{e_r=0}^{3} v_{i,e_r} \Sigma_{d_r=0}^{7} w_{d_r} (a_{i,i+j}^{(r)})^{-2^d} \\ &= \Sigma_{e_r=0}^{3} \Sigma_{d_r=0}^{7} w_{i,e_r,d_r} (a_{e_r,e_r+j}^{(r)})^{-2^{d_r}} \end{aligned}$$

Combining this with the AddRoundKey is trivial:

$$\begin{aligned} a_{i,j}^{(r)} &= k_{i,j}^{(r)} + \Sigma_{e_r=0}^{3}\Sigma_{d_r=0}^{7} w_{i,e_r,d_r}(a_{e_r,e_{r+j}}^{(r)})^{-2^{d_r}} \\ &= k_{i,j}^{(r)} + \Sigma_{d,e}\frac{w_{i,e_r,d_r}}{(a_{e_r,e_{r+j}}^{(r)})^{2^{d_r}}} \end{aligned}$$

Repeating this process and combining the derived equations at each step, the 5-round equation is formulated as

$$a_{i,j}^{(6)} = K + \Sigma_{e_5,d_5}\frac{C}{K^* + \Sigma_{e_4,d_4}\frac{C}{K^* + \Sigma_{e_3,d_3}\frac{C}{K^* + \Sigma_{e_2,d_2}\frac{C}{K^* + \Sigma_{e_1,d_1}\frac{C}{K^* + p_*^*}}}}}$$

where K and K^* are round keys and C are constants. Extensions can be made to arbitrary n round equations following the same pattern of computations.

Although the Ferguson-Schroeppel-Whiting derivation is acknowledged in almost every research paper in the area afterwards, the fact that there are around 2^{50} terms in equation system defining the 10-round Rijndael-128 makes it intractable. In the following years, several more concise expressions of the cipher have been found, making this expression more or less obselete.

5.2 The Cid-Murphy-Robshaw Equations

The derivation of equations by Sean Murphy and Matthew Robshaw [18] in 2002, followed up by their paper along with Carlos Cid in 2004 [4], is the most recent work in the area and acts more or less as an unofficial standard for research on algebraic attacks on Rijndael. They set out to resolve the conflict caused by Rijndael operating in two differend finite fields (GF(2) and GF(2^8)) and to make it easier to cryptanalyze the cipher by defining a more general cipher almost identical to Rijndael. The cipher they develop is called the Big Encryption Standard (BES). They simply iterate through the encryption process and express each step as a system of equations. In describing the derivation, we list, step by step, the procedure as presented in their paper.

First, the round function of BES is described, which is virtually identical to that of Rijndael:

$$b \rightarrow M_b * b^{-1} + (k_B)_i$$

where b is the input byte, M_b is a matrix to represent the linear diffusion phase (ShiftRows and MixColumns) and $(k_B)_i$ is the round key.

Using this, the full encryption process for the 10-round BES cipher is simply described as:

$$\begin{array}{rcll} w_0 & = & p + k_0 & \\ x_i & = & w_i^{-1} & 0 \le i \le 9 \\ w_i & = & M_B * x_{i-1} + k_i & 0 \le i \le 9 \\ c & = & M_B^* * x_9 + k_{10} & \end{array}$$

where p is the plaintext, c is the ciphertext, k_i is the round key for the ith round, w_i is the state before the ith round and x_i is the state after the ith round ($0 \le i \le 9$). M_B^* is a modified linear diffusion matrix for the last round, as the last round differs from the other rounds by not using the MixColumns phase.

When the equation system above is rearranged, the entire encryption process can be expressed as a set of multivariate quadratic equations. The $(8 * j + m)$th bit of the bit vectors k_i, w_i and x_i are expressed as $k_{i,(j,m)}$, $w_{i,(j,m)}$ and $x_{i,(j,m)}$ respectively ($0 \le j \le 15$ and $0 \le m \le 7$) and the constant diffusion matrices M_B and M_B^* are represented by α and β :

$$\begin{array}{rll} 0 = & w_{0,(j,m)} + p_{j,m} + k_{0,(j,m)} & \\ 0 = & x_{i,(j,m)} * w_{i,(j,m)} + 1 & 0 \le i \le 9 \\ 0 = & w_{i,(j,m)} + \Sigma_{(j',m')} \alpha_{(j,m),(j',m')} x_{i-1,(m',j')} & 0 \le i \le 9 \\ 0 = & c_{(j,m)} + k_{10,(j,m)} + \Sigma_{(j',m')} \beta_{(j,m),(j',m')} x_{9,(m',j')} & \end{array}$$

Written out, this clear and simple equation set is observed to be an MQ system of 5248 equations and 7808 terms. 3840 of these equations are extremely sparse quadratic equations, while the remaining 1408 are linear. The terms in the system consist of 2560 state variables and 1408 key variables.

5.3 The Biryukov-De Canniere Comparison of Equations

In 2002, Alex Biryukov and Christophe De Canniere included Rijndael along with 5 others in their comparison of MQ representations of block ciphers [2]. Their main objective was to compare these representations for each cipher rather than focus on one cipher alone. They have emphasized the need to

express the systems as simply as possible. This meant minimizing the number of equations and variables.

In constructing the MQ system that fully defines the Rijndael encryption process, they have cited the Murphy-Robshaw derivation rather than making original contribution. The comparison of the 128-bit Rijndael MQ with the other 128-bit ciphers is interesting:

	Khazad	Mistyl	Kasumi	Camellia	Rijndael	Serpent
Variables	9600	6832	7952	8880	9800	11960
Total monomials	17264	10688	12216	15104	16096	29640
Equations	7664	3856	4264	6224	6296	17680

Table 1: MQ sizes of the ciphers

Judging only by the numbers, whereas it would be impossible to make a strong statement about the vulnerability of any cipher against algebraic attacks, it is apparent that Rijndael can be represented as a much smaller, sparser and more overdefined system than its AES-rival, Serpent. These adjectives are indeed the characteristics that were emphasized in the XSL cipher and are shown by our experiments that make the attack more practical.

6 Field Transformation

In order to compare the performance of the XSL algorithm for two different representations of the same equation system, it is necessary that the transformation from a byte-system to a bit-system be defined. The transformation formulae are derived from the definition of multiplication operation as defined in GF(2^8). There are three types of terms that need to be transformed:

- Transforming *the constants* is trivial. The 8 bits of the constant term in the original equation are written as the constant terms of the transformed equation system in bits.
- *The linear terms* take a bit more calculation than the constants. One linear term is spread into 8 equations and each term defines 8 new variables in the transformed system, each new variable being one bit in the original byte. The coefficients of the new terms are expressed as a certain linear combination of the bits of the original byte-coefficient. The formulae are provided in the next section.

- Transforming *the quadratic terms* is done by repeating the process of transforming the linear terms; each new term is quadratic in the new system, and the coefficients are again certain linear combinations of the original byte-coefficient. The formulae for quadratic term transformation follow the linear term transformation term formulae.

6.1 Univariate Linear Field Transformation

The simple, univariate, linear byte-equation

$$z = a * x$$

where a is the coefficient and x is the variable, translates to the following 8 bit-equations:

$$\begin{aligned}
z_7 &= x_7 * a_7 + \textstyle\sum_{i=5}^{7} x_i * a_{12-i} + \sum_{i=4}^{7} x_i * a_{11-i} + \sum_{i=0}^{7} x_i * a_{7-i} \\
z_6 &= \textstyle\sum_{i=6}^{7} x_i * a_{13-i} + \sum_{i=4}^{7} x_i * a_{11-i} + \sum_{i=3}^{7} x_i * a_{10-i} + \sum_{i=0}^{6} x_i * a_{6-i} \\
z_5 &= \textstyle\sum_{i=5}^{7} x_i * a_{12-i} + \sum_{i=3}^{7} x_i * a_{10-i} + \sum_{i=2}^{7} x_i * a_{9-i} + \sum_{i=0}^{5} x_i * a_{5-i} \\
z_4 &= x_7 * a_7 + \textstyle\sum_{i=4}^{7} x_i * a_{11-i} + \sum_{i=2}^{7} x_i * a_{9-i} + \sum_{i=1}^{7} x_i * a_{8-i} \\
&\quad + \textstyle\sum_{i=0}^{4} x_i * a_{4-i} \\
z_3 &= x_7 * a_7 + \textstyle\sum_{i=6}^{7} x_i * a_{13-i} + \sum_{i=5}^{7} x_i * a_{12-i} + \sum_{i=4}^{7} x_i * a_{11-i} \\
&\quad + \textstyle\sum_{i=3}^{7} x_i * a_{10-i} + \sum_{i=1}^{7} x_i * a_{8-i} + \sum_{i=0}^{3} x_i * a_{3-i} \\
z_2 &= \textstyle\sum_{i=6}^{7} x_i * a_{13-i} + \sum_{i=3}^{7} x_i * a_{10-i} + \sum_{i=2}^{7} x_i * a_{9-i} + \sum_{i=0}^{2} x_i * a_{2-i} \\
z_1 &= x_7 * y_7 + \textstyle\sum_{i=5}^{7} x_i * a_{12-i} + \sum_{i=2}^{7} x_i * a_{9-i} + \sum_{i=1}^{7} x_i * a_{8-i} \\
&\quad + \textstyle\sum_{i=0}^{1} x_i * a_{1-i} \\
z_0 &= \textstyle\sum_{i=6}^{7} x_i * a_{13-i} + \sum_{i=5}^{7} x_i * a_{12-i} + \sum_{i=1}^{7} x_i * a_{8-i} + x_0 * a_0
\end{aligned}$$

6.2 Multivariate Quadratic Field Transformation

The simple, multivariate, quadratic byte-equation

$$z = a * x * y$$

where a is the coefficient and x and y are the variables, translates to the following 8 bit-equations:

$$
\begin{aligned}
z_7 \;=\;& (a_7)(x_0y_0)+(a_6)(x_1y_0)+(a_5)(x_2y_0)+(a_4)(x_3y_0)+(a_7+a_3)(x_4y_0)\\
&+(a_7+a_6+a_2)(x_5y_0)+(a_6+a_5+a_1)(x_6y_0)+(a_7+a_5+a_4+a_0)(x_7y_0)\\
&+(a_6)(x_0y_1)+(a_5)(x_1y_1)+(a_4)(x_2y_1)+(a_7+a_3)(x_3y_1)+(a_7+a_6+a_2)(x_4y_1)\\
&+(a_6+a_5+a_1)(x_5y_1)+(a_7+a_5+a_4+a_0)(x_6y_1)+(a_6+a_4+a_3)(x_7y_1)\\
&+(a_5)(x_0y_2)+(a_4)(x_1y_2)+(a_7+a_3)(x_2y_2)+(a_7+a_6+a_2)(x_3y_2)\\
&+(a_6+a_5+a_1)(x_4y_2)+(a_7+a_5+a_4+a_0)(x_5y_2)+(a_6+a_4+a_3)(x_6y_2)\\
&+(a_5+a_3+a_2)(x_7y_2)+(a_4)(x_0y_3)+(a_7+a_3)(x_1y_3)+(a_7+a_6+a_2)(x_2y_3)\\
&+(a_6+a_5+a_1)(x_3y_3)+(a_7+a_5+a_4+a_0)(x_4y_3)+(a_6+a_4+a_3)(x_5y_3)\\
&+(a_5+a_3+a_2)(x_6y_3)+(a_7+a_4+a_2+a_1)(x_7y_3)+(a_7+a_3)(x_0y_4)\\
&+(a_7+a_6+a_2)(x_1y_4)+(a_6+a_5+a_1)(x_2y_4)+(a_7+a_5+a_4+a_0)(x_3y_4)\\
&+(a_6+a_4+a_3)(x_4y_4)+(a_5+a_3+a_2)(x_5y_4)+(a_7+a_4+a_2+a_1)(x_6y_4)\\
&+(a_6+a_3+a_1+a_0)(x_7y_4)+(a_7+a_6+a_2)(x_0y_5)\\
&+(a_6+a_5+a_1)(x_1y_5)+(a_7+a_5+a_4+a_0)(x_2y_5)+(a_6+a_4+a_3)(x_3y_5)\\
&+(a_5+a_3+a_2)(x_4y_5)+(a_7+a_4+a_2+a_1)(x_5y_5)+(a_6+a_3+a_1+a_0)(x_6y_5)\\
&+(a_7+a_5+a_2+a_0)(x_7y_5)+(a_6+a_5+a_1)(x_0y_6)+(a_7+a_5+a_4+a_0)(x_1y_6)\\
&+(a_6+a_4+a_3)(x_2y_6)+(a_5+a_3+a_2)(x_3y_6)+(a_7+a_4+a_2+a_1)(x_4y_6)\\
&+(a_6+a_3+a_1+a_0)(x_5y_6)+(a_7+a_5+a_2+a_0)(x_6y_6)+(a_7+a_6+a_4+a_1)(x_7y_6)\\
&+(a_7+a_5+a_4+a_0)(x_0y_7)+(a_6+a_4+a_3)(x_1y_7)+(a_5+a_3+a_2)(x_2y_7)\\
&+(a_7+a_4+a_2+a_1)(x_3y_7)+(a_6+a_3+a_1+a_0)(x_4y_7)+(a_7+a_5+a_2+a_0)(x_5y_7)\\
&+(a_7+a_6+a_4+a_1)(x_6y_7)+(a_6+a_5+a_3+a_0)(x_7y_7)\\
z_6 \;=\;& (a_6)(x_0y_0)+(a_5)(x_1y_0)+(a_4)(x_2y_0)+(a_7+a_3)(x_3y_0)\\
&+(a_7+a_6+a_2)(x_4y_0)+(a_6+a_5+a_1)(x_5y_0)+(a_7+a_5+a_4+a_0)(x_6y_0)\\
&+(a_6+a_4+a_3)(x_7y_0)+(a_5)(x_0y_1)+(a_4)(x_1y_1)+(a_7+a_3)(x_2y_1)\\
&+(a_7+a_6+a_2)(x_3y_1)+(a_6+a_5+a_1)(x_4y_1)+(a_7+a_5+a_4+a_0)(x_5y_1)\\
&+(a_6+a_4+a_3)(x_6y_1)+(a_5+a_3+a_2)(x_7y_1)+(a_4)(x_0y_2)\\
&+(a_7+a_3)(x_1y_2)+(a_7+a_6+a_2)(x_2y_2)+(a_6+a_5+a_1)(x_3y_2)\\
&+(a_7+a_5+a_4+a_0)(x_4y_2)+(a_6+a_4+a_3)(x_5y_2)+(a_5+a_3+a_2)(x_6y_2)\\
&+(a_7+a_4+a_2+a_1)(x_7y_2)+(a_7+a_3)(x_0y_3)+(a_7+a_6+a_2)(x_1y_3)\\
&+(a_6+a_5+a_1)(x_2y_3)+(a_7+a_5+a_4+a_0)(x_3y_3)+(a_6+a_4+a_3)(x_4y_3)\\
&+(a_5+a_3+a_2)(x_5y_3)+(a_7+a_4+a_2+a_1)(x_6y_3)+(a_6+a_3+a_1+a_0)(x_7y_3)\\
&+(a_7+a_6+a_2)(x_0y_4)+(a_6+a_5+a_1)(x_1y_4)+(a_7+a_5+a_4+a_0)(x_2y_4)\\
&+(a_6+a_4+a_3)(x_3y_4)+(a_5+a_3+a_2)(x_4y_4)+(a_7+a_4+a_2+a_1)(x_5y_4)\\
&+(a_6+a_3+a_1+a_0)(x_6y_4)+(a_7+a_5+a_2+a_0)(x_7y_4)+(a_6+a_5+a_1)(x_0y_5)\\
&+(a_7+a_5+a_4+a_0)(x_1y_5)+(a_6+a_4+a_3)(x_2y_5)+(a_5+a_3+a_2)(x_3y_5)\\
&+(a_7+a_4+a_2+a_1)(x_4y_5)+(a_6+a_3+a_1+a_0)(x_5y_5)+(a_7+a_5+a_2+a_0)(x_6y_5)\\
&+(a_7+a_6+a_4+a_1)(x_7y_5)+(a_7+a_5+a_4+a_0)(x_0y_6)+(a_6+a_4+a_3)(x_1y_6)\\
&+(a_5+a_3+a_2)(x_2y_6)+(a_7+a_4+a_2+a_1)(x_3y_6)+(a_6+a_3+a_1+a_0)(x_4y_6)\\
&+(a_7+a_5+a_2+a_0)(x_5y_6)+(a_7+a_6+a_4+a_1)(x_6y_6)+(a_6+a_5+a_3+a_0)(x_7y_6)\\
&+(a_6+a_4+a_3)(x_0y_7)+(a_5+a_3+a_2)(x_1y_7)+(a_7+a_4+a_2+a_1)(x_2y_7)\\
&+(a_6+a_3+a_1+a_0)(x_3y_7)+(a_7+a_5+a_2+a_0)(x_4y_7)+(a_7+a_6+a_4+a_1)(x_5y_7)\\
&+(a_6+a_5+a_3+a_0)(x_6y_7)+(a_5+a_4+a_2)(x_7y_7)
\end{aligned}
$$

$$
\begin{aligned}
z_5 \;=\;& (a_5)(x_0y_0) + (a_4)(x_1y_0) + (a_7 + a_3)(x_2y_0) + (a_7 + a_6 + a_2)(x_3y_0) \\
&+(a_6 + a_5 + a_1)(x_4y_0) + (a_7 + a_5 + a_4 + a_0)(x_5y_0) + (a_6 + a_4 + a_3)(x_6y_0) \\
&+(a_5 + a_3 + a_2)(x_7y_0) + (a_4)(x_0y_1) + (a_7 + a_3)(x_1y_1) + (a_7 + a_6 + a_2)(x_2y_1) \\
&+(a_6 + a_5 + a_1)(x_3y_1) + (a_7 + a_5 + a_4 + a_0)(x_4y_1) + (a_6 + a_4 + a_3)(x_5y_1) \\
&+(a_5 + a_3 + a_2)(x_6y_1) + (a_7 + a_4 + a_2 + a_1)(x_7y_1) + (a_7 + a_3)(x_0y_2) \\
&+(a_7 + a_6 + a_2)(x_1y_2) + (a_6 + a_5 + a_1)(x_2y_2) + (a_7 + a_5 + a_4 + a_0)(x_3y_2) \\
&+(a_6 + a_4 + a_3)(x_4y_2) + (a_5 + a_3 + a_2)(x_5y_2) + (a_7 + a_4 + a_2 + a_1)(x_6y_2) \\
&+(a_6 + a_3 + a_1 + a_0)(x_7y_2) + (a_7 + a_6 + a_2)(x_0y_3) + (a_6 + a_5 + a_1)(x_1y_3) \\
&+(a_7 + a_5 + a_4 + a_0)(x_2y_3) + (a_6 + a_4 + a_3)(x_3y_3) + (a_5 + a_3 + a_2)(x_4y_3) \\
&+(a_7 + a_4 + a_2 + a_1)(x_5y_3) + (a_6 + a_3 + a_1 + a_0)(x_6y_3) + (a_7 + a_5 + a_2 + a_0)(x_7y_3) \\
&+(a_6 + a_5 + a_1)(x_0y_4) + (a_7 + a_5 + a_4 + a_0)(x_1y_4) + (a_6 + a_4 + a_3)(x_2y_4) \\
&+(a_5 + a_3 + a_2)(x_3y_4) + (a_7 + a_4 + a_2 + a_1)(x_4y_4) + (a_6 + a_3 + a_1 + a_0)(x_5y_4) \\
&+(a_7 + a_5 + a_2 + a_0)(x_6y_4) + (a_7 + a_6 + a_4 + a_1)(x_7y_4) + (a_7 + a_5 + a_4 + a_0)(x_0y_5) \\
&+(a_6 + a_4 + a_3)(x_1y_5) + (a_5 + a_3 + a_2)(x_2y_5) + (a_7 + a_4 + a_2 + a_1)(x_3y_5) \\
&+(a_6 + a_3 + a_1 + a_0)(x_4y_5) + (a_7 + a_5 + a_2 + a_0)(x_5y_5) + (a_7 + a_6 + a_4 + a_1)(x_6y_5) \\
&+(a_6 + a_5 + a_3 + a_0)(x_7y_5) + (a_6 + a_4 + a_3)(x_0y_6) + (a_5 + a_3 + a_2)(x_1y_6) \\
&+(a_7 + a_4 + a_2 + a_1)(x_2y_6) + (a_6 + a_3 + a_1 + a_0)(x_3y_6) + (a_7 + a_5 + a_2 + a_0)(x_4y_6) \\
&+(a_7 + a_6 + a_4 + a_1)(x_5y_6) + (a_6 + a_5 + a_3 + a_0)(x_6y_6) + (a_5 + a_4 + a_2)(x_7y_7) \\
&+(a_5 + a_3 + a_2)(x_0y_7) + (a_7 + a_4 + a_2 + a_1)(x_1y_7) + (a_6 + a_3 + a_1 + a_0)(x_2y_7) \\
&+(a_7 + a_5 + a_2 + a_0)(x_3y_7) + (a_7 + a_6 + a_4 + a_1)(x_4y_7) + (a_6 + a_5 + a_3 + a_0)(x_5y_7) \\
&+(a_5 + a_4 + a_2)(x_6y_7) + (a_7 + a_4 + a_3 + a_1)(x_7y_7) \\
z_4 \;=\;& (a_4)(x_0y_0) + (a_7 + a_3)(x_1y_0) + (a_7 + a_6 + a_2)(x_2y_0) + (a_6 + a_5 + a_1)(x_3y_0) \\
&+(a_7 + a_5 + a_4 + a_0)(x_4y_0) + (a_6 + a_4 + a_3)(x_5y_0) + (a_5 + a_3 + a_2)(x_6y_0) \\
&+(a_7 + a_4 + a_2 + a_1)(x_7y_0) + (a_7 + a_3)(x_0y_1) + (a_7 + a_6 + a_2)(x_1y_1) \\
&+(a_6 + a_5 + a_1)(x_2y_1) + (a_7 + a_5 + a_4 + a_0)(x_3y_1) + (a_6 + a_4 + a_3)(x_4y_1) \\
&+(a_5 + a_3 + a_2)(x_5y_1) + (a_7 + a_4 + a_2 + a_1)(x_6y_1) + (a_6 + a_3 + a_1 + a_0)(x_7y_1) \\
&+(a_7 + a_6 + a_2)(x_0y_2) + (a_6 + a_5 + a_1)(x_1y_2) + (a_7 + a_5 + a_4 + a_0)(x_2y_2) \\
&+(a_6 + a_4 + a_3)(x_3y_2) + (a_5 + a_3 + a_2)(x_4y_2) + (a_7 + a_4 + a_2 + a_1)(x_5y_2) \\
&+(a_6 + a_3 + a_1 + a_0)(x_6y_2) + (a_7 + a_5 + a_2 + a_0)(x_7y_2) + (a_6 + a_5 + a_1)(x_0y_3) \\
&+(a_7 + a_5 + a_4 + a_0)(x_1y_3) + (a_6 + a_4 + a_3)(x_2y_3) + (a_5 + a_3 + a_2)(x_3y_3) \\
&+(a_7 + a_4 + a_2 + a_1)(x_4y_3) + (a_6 + a_3 + a_1 + a_0)(x_5y_3) + (a_7 + a_5 + a_2 + a_0)(x_6y_3) \\
&+(a_7 + a_6 + a_4 + a_1)(x_7y_3) + (a_7 + a_5 + a_4 + a_0)(x_0y_4) + (a_6 + a_4 + a_3)(x_1y_4) \\
&+(a_5 + a_3 + a_2)(x_2y_4) + (a_7 + a_4 + a_2 + a_1)(x_3y_4) + (a_6 + a_3 + a_1 + a_0)(x_4y_4) \\
&+(a_7 + a_5 + a_2 + a_0)(x_5y_4) + (a_7 + a_6 + a_4 + a_1)(x_6y_4) + (a_6 + a_5 + a_3 + a_0)(x_7y_4) \\
&+(a_6 + a_4 + a_3)(x_0y_5) + (a_5 + a_3 + a_2)(x_1y_5) + (a_7 + a_4 + a_2 + a_1)(x_2y_5) \\
&+(a_6 + a_3 + a_1 + a_0)(x_3y_5) + (a_7 + a_5 + a_2 + a_0)(x_4y_5) + (a_7 + a_6 + a_4 + a_1)(x_5y_5) \\
&+(a_6 + a_5 + a_3 + a_0)(x_6y_5) + (a_5 + a_4 + a_2)(x_7y_5) + (a_5 + a_3 + a_2)(x_0y_6) \\
&+(a_7 + a_4 + a_2 + a_1)(x_1y_6) + (a_6 + a_3 + a_1 + a_0)(x_2y_6) + (a_7 + a_5 + a_2 + a_0)(x_3y_6) \\
&+(a_7 + a_6 + a_4 + a_1)(x_4y_6) + (a_6 + a_5 + a_3 + a_0)(x_5y_6) + (a_5 + a_4 + a_2)(x_6y_6) \\
&+(a_7 + a_4 + a_3 + a_1)(x_7y_6) + (a_7 + a_4 + a_2 + a_1)(x_0y_7) + (a_6 + a_3 + a_1 + a_0)(x_1y_7) \\
&+(a_7 + a_5 + a_2 + a_0)(x_2y_7) + (a_7 + a_6 + a_4 + a_1)(x_3y_7) + (a_6 + a_5 + a_3 + a_0)(x_4y_7) \\
&+(a_5 + a_4 + a_2)(x_5y_7) + (a_7 + a_4 + a_3 + a_1)(x_6y_7) + (a_7 + a_6 + a_3 + a_2 + a_0)(x_7y_7)
\end{aligned}
$$

$$
\begin{aligned}
z_3 \;=\;& (a_3)(x_0y_0) + (a_7 + a_2)(x_1y_0) + (a_6 + a_1)(x_2y_0) + (a_7 + a_5 + a_0)(x_3y_0) \\
&+(a_7 + a_6 + a_4)(x_4y_0) + (a_7 + a_6 + a_5 + a_3)(x_5y_0) \\
&+(a_7 + a_6 + a_5 + a_4 + a_2)(x_6y_0) + (a_7 + a_6 + a_5 + a_4 + a_3 + a_1)(x_7y_0) \\
&+(a_7 + a_2)(x_0y_1) + (a_6 + a_1)(x_1y_1) + (a_7 + a_5 + a_0)(x_2y_1) \\
&+(a_7 + a_6 + a_4)(x_3y_1) + (a_7 + a_6 + a_5 + a_3)(x_4y_1) \\
&+(a_7 + a_6 + a_5 + a_4 + a_2)(x_5y_1) + (a_7 + a_6 + a_5 + a_4 + a_3 + a_1)(x_6y_1) \\
&+(a_7 + a_6 + a_5 + a_4 + a_3 + a_2 + a_0)(x_7y_1) + (a_6 + a_1)(x_0y_2) \\
&+(a_7 + a_5 + a_0)(x_1y_2) + (a_7 + a_6 + a_4)(x_2y_2) + (a_7 + a_6 + a_5 + a_3)(x_3y_2) \\
&+(a_7 + a_6 + a_5 + a_4 + a_2)(x_4y_2) + (a_7 + a_6 + a_5 + a_4 + a_3 + a_1)(x_5y_2) \\
&+(a_7 + a_6 + a_5 + a_4 + a_3 + a_2 + a_0)(x_6y_2) \\
&+(a_7 + a_6 + a_5 + a_4 + a_3 + a_2 + a_1)(x_7y_2) + (a_7 + a_5 + a_0)(x_0y_3) \\
&+(a_7 + a_6 + a_4)(x_1y_3) + (a_7 + a_6 + a_5 + a_3)(x_2y_3) \\
&+(a_7 + a_6 + a_5 + a_4 + a_2)(x_3y_3) + (a_7 + a_6 + a_5 + a_4 + a_3 + a_1)(x_4y_3) \\
&+(a_7 + a_6 + a_5 + a_4 + a_3 + a_2 + a_0)(x_5y_3) \\
&+(a_7 + a_6 + a_5 + a_4 + a_3 + a_2 + a_1)(x_6y_3) \\
&+(a_7 + a_6 + a_5 + a_4 + a_3 + a_2 + a_1 + a_0)(x_7y_3) + (a_7 + a_6 + a_4)(x_0y_4) \\
&+(a_7 + a_6 + a_5 + a_3)(x_1y_4) + (a_7 + a_6 + a_5 + a_4 + a_2)(x_2y_4) \\
&+(a_7 + a_6 + a_5 + a_4 + a_3 + a_1)(x_3y_4) \\
&+(a_7 + a_6 + a_5 + a_4 + a_3 + a_2 + a_0)(x_4y_4) \\
&+(a_7 + a_6 + a_5 + a_4 + a_3 + a_2 + a_1)(x_5y_4) \\
&+(a_7 + a_6 + a_5 + a_4 + a_3 + a_2 + a_1 + a_0)(x_6y_4) \\
&+(a_6 + a_5 + a_4 + a_3 + a_2 + a_1 + a_0)(x_7y_4) + (a_7 + a_6 + a_5 + a_3)(x_0y_5) \\
&+(a_7 + a_6 + a_5 + a_4 + a_2)(x_1y_5) + (a_7 + a_6 + a_5 + a_4 + a_3 + a_1)(x_2y_5) \\
&+(a_7 + a_6 + a_5 + a_4 + a_3 + a_2 + a_0)(x_3y_5) \\
&+(a_7 + a_6 + a_5 + a_4 + a_3 + a_2 + a_1)(x_4y_5) \\
&+(a_7 + a_6 + a_5 + a_4 + a_3 + a_2 + a_1 + a_0)(x_5y_5) \\
&+(a_6 + a_5 + a_4 + a_3 + a_2 + a_1 + a_0)(x_6y_5) \\
&+(a_5 + a_4 + a_3 + a_2 + a_1 + a_0)(x_7y_5) + (a_7 + a_6 + a_5 + a_4 + a_2)(x_0y_6) \\
&+(a_7 + a_6 + a_5 + a_4 + a_3 + a_1)(x_1y_6) \\
&+(a_7 + a_6 + a_5 + a_4 + a_3 + a_2 + a_0)(x_2y_6) \\
&+(a_7 + a_6 + a_5 + a_4 + a_3 + a_2 + a_1)(x_3y_6) \\
&+(a_7 + a_6 + a_5 + a_4 + a_3 + a_2 + a_1 + a_0)(x_4y_6) \\
&+(a_6 + a_5 + a_4 + a_3 + a_2 + a_1 + a_0)(x_5y_6) \\
&+(a_5 + a_4 + a_3 + a_2 + a_1 + a_0)(x_6y_6) + (a_4 + a_3 + a_2 + a_1 + a_0)(x_7y_6) \\
&+(a_7 + a_6 + a_5 + a_4 + a_3 + a_1)(x_0y_7) \\
&+(a_7 + a_6 + a_5 + a_4 + a_3 + a_2 + a_0)(x_1y_7) \\
&+(a_7 + a_6 + a_5 + a_4 + a_3 + a_2 + a_1)(x_2y_7) \\
&+(a_7 + a_6 + a_5 + a_4 + a_3 + a_2 + a_1 + a_0)(x_3y_7) \\
&+(a_6 + a_5 + a_4 + a_3 + a_2 + a_1 + a_0)(x_4y_7) \\
&+(a_5 + a_4 + a_3 + a_2 + a_1 + a_0)(x_5y_7) \\
&+(a_4 + a_3 + a_2 + a_1 + a_0)(x_6y_7) + (a_3 + a_2 + a_1 + a_0)(x_7y_7)
\end{aligned}
$$

$$
\begin{aligned}
z_2 \;=\;& (a_2)(x_0y_0) + (a_1)(x_1y_0) + (a_7 + a_0)(x_2y_0) + (a_7 + a_6)(x_3y_0) + (a_6 + a_5)(x_4y_0)\\
&+(a_5 + a_4)(x_5y_0) + (a_7 + a_4 + a_3)(x_6y_0) + (a_6 + a_3 + a_2)(x_7y_0)\\
&+(a_1)(x_0y_1) + (a_7 + a_0)(x_1y_1) + (a_7 + a_6)(x_2y_1) + (a_6 + a_5)(x_3y_1)\\
&+(a_5 + a_4)(x_4y_1) + (a_7 + a_4 + a_3)(x_5y_1) + (a_6 + a_3 + a_2)(x_6y_1)\\
&+(a_7 + a_5 + a_2 + a_1)(x_7y_1) + (a_7 + a_0)(x_0y_2) + (a_7 + a_6)(x_1y_2)\\
&+(a_6 + a_5)(x_2y_2) + (a_5 + a_4)(x_3y_2) + (a_7 + a_4 + a_3)(x_4y_2)\\
&+(a_6 + a_3 + a_2)(x_5y_2) + (a_7 + a_5 + a_2 + a_1)(x_6y_2)\\
&+(a_7 + a_6 + a_4 + a_1 + a_0)(x_7y_2) + (a_7 + a_6)(x_0y_3) + (a_6 + a_5)(x_1y_3)\\
&+(a_5 + a_4)(x_2y_3) + (a_7 + a_4 + a_3)(x_3y_3) + (a_6 + a_3 + a_2)(x_4y_3)\\
&+(a_7 + a_5 + a_2 + a_1)(x_5y_3) + (a_7 + a_6 + a_4 + a_1 + a_0)(x_6y_3)\\
&+(a_7 + a_6 + a_5 + a_3 + a_0)(x_7y_3) + (a_6 + a_5)(x_0y_4) + (a_5 + a_4)(x_1y_4)\\
&+(a_7 + a_4 + a_3)(x_2y_4) + (a_6 + a_3 + a_2)(x_3y_4) + (a_7 + a_5 + a_2 + a_1)(x_4y_4)\\
&+(a_7 + a_6 + a_4 + a_1 + a_0)(x_5y_4) + (a_7 + a_6 + a_5 + a_3 + a_0)(x_6y_4)\\
&+(a_6 + a_5 + a_4 + a_2)(x_7y_4) + (a_5 + a_4)(x_0y_5) + (a_7 + a_4 + a_3)(x_1y_5)\\
&+(a_6 + a_3 + a_2)(x_2y_5) + (a_7 + a_5 + a_2 + a_1)(x_3y_5)\\
&+(a_7 + a_6 + a_4 + a_1 + a_0)(x_4y_5) + (a_7 + a_6 + a_5 + a_3 + a_0)(x_5y_5)\\
&+(a_6 + a_5 + a_4 + a_2)(x_6y_5) + (a_7 + a_5 + a_4 + a_3 + a_1)(x_7y_5)\\
&+(a_7 + a_4 + a_3)(x_0y_6) + (a_6 + a_3 + a_2)(x_1y_6) + (a_7 + a_5 + a_2 + a_1)(x_2y_6)\\
&+(a_7 + a_6 + a_4 + a_1 + a_0)(x_3y_6) + (a_7 + a_6 + a_5 + a_3 + a_0)(x_4y_6)\\
&+(a_6 + a_5 + a_4 + a_2)(x_5y_6) + (a_7 + a_5 + a_4 + a_3 + a_1)(x_6y_6)\\
&+(a_7 + a_6 + a_4 + a_3 + a_2 + a_0)(x_7y_6) + (a_6 + a_3 + a_2)(x_0y_7)\\
&+(a_7 + a_5 + a_2 + a_1)(x_1y_7) + (a_7 + a_6 + a_4 + a_1 + a_0)(x_2y_7)\\
&+(a_7 + a_6 + a_5 + a_3 + a_0)(x_3y_7) + (a_6 + a_5 + a_4 + a_2)(x_4y_7)\\
&+(a_7 + a_5 + a_4 + a_3 + a_1)(x_5y_7) + (a_7 + a_6 + a_4 + a_3 + a_2 + a_0)(x_6y_7)\\
&+(a_7 + a_6 + a_5 + a_3 + a_2 + a_1)(x_7y_7)\\
z_1 \;=\;& (a_1)(x_0y_0) + (a_7 + a_0)(x_1y_0) + (a_7 + a_6)(x_2y_0) + (a_6 + a_5)(x_3y_0)\\
&+(a_5 + a_4)(x_4y_0) + (a_7 + a_4 + a_3)(x_5y_0) + (a_6 + a_3 + a_2)(x_6y_0)\\
&+(a_7 + a_5 + a_2 + a_1)(x_7y_0) + (a_7 + a_0)(x_0y_1) + (a_7 + a_6)(x_1y_1)\\
&+(a_6 + a_5)(x_2y_1) + (a_5 + a_4)(x_3y_1) + (a_7 + a_4 + a_3)(x_4y_1)\\
&+(a_6 + a_3 + a_2)(x_5y_1) + (a_7 + a_5 + a_2 + a_1)(x_6y_1)\\
&+(a_7 + a_6 + a_4 + a_1 + a_0)(x_7y_1) + (a_7 + a_6)(x_0y_2) + (a_6 + a_5)(x_1y_2)\\
&+(a_5 + a_4)(x_2y_2) + (a_7 + a_4 + a_3)(x_3y_2) + (a_6 + a_3 + a_2)(x_4y_2)\\
&+(a_7 + a_5 + a_2 + a_1)(x_5y_2) + (a_7 + a_6 + a_4 + a_1 + a_0)(x_6y_2)\\
&+(a_7 + a_6 + a_5 + a_3 + a_0)(x_7y_2) + (a_6 + a_5)(x_0y_3) + (a_5 + a_4)(x_1y_3)\\
&+(a_7 + a_4 + a_3)(x_2y_3) + (a_6 + a_3 + a_2)(x_3y_3) + (a_7 + a_5 + a_2 + a_1)(x_4y_3)\\
&+(a_7 + a_6 + a_4 + a_1 + a_0)(x_5y_3) + (a_7 + a_6 + a_5 + a_3 + a_0)(x_6y_3)\\
&+(a_6 + a_5 + a_4 + a_2)(x_7y_3) + (a_5 + a_4)(x_0y_4) + (a_7 + a_4 + a_3)(x_1y_4)\\
&+(a_6 + a_3 + a_2)(x_2y_4) + (a_7 + a_5 + a_2 + a_1)(x_3y_4)\\
&+(a_7 + a_6 + a_4 + a_1 + a_0)(x_4y_4) + (a_7 + a_6 + a_5 + a_3 + a_0)(x_5y_4)\\
&+(a_6 + a_5 + a_4 + a_2)(x_6y_4) + (a_7 + a_5 + a_4 + a_3 + a_1)(x_7y_4)\\
&+(a_7 + a_4 + a_3)(x_0y_5) + (a_6 + a_3 + a_2)(x_1y_5) + (a_7 + a_5 + a_2 + a_1)(x_2y_5)
\end{aligned}
$$

$$
\begin{aligned}
& +(a_7+a_6+a_4+a_1+a_0)(x_3y_5)+(a_7+a_6+a_5+a_3+a_0)(x_4y_5) \\
& +(a_6+a_5+a_4+a_2)(x_5y_5)+(a_7+a_5+a_4+a_3+a_1)(x_6y_5) \\
& +(a_7+a_6+a_4+a_3+a_2+a_0)(x_7y_5)+(a_6+a_3+a_2)(x_0y_6) \\
& +(a_7+a_5+a_2+a_1)(x_1y_6)+(a_7+a_6+a_4+a_1+a_0)(x_2y_6) \\
& +(a_7+a_6+a_5+a_3+a_0)(x_3y_6)+(a_6+a_5+a_4+a_2)(x_4y_6) \\
& +(a_7+a_5+a_4+a_3+a_1)(x_5y_6)+(a_7+a_6+a_4+a_3+a_2+a_0)(x_6y_6) \\
& +(a_7+a_6+a_5+a_3+a_2+a_1)(x_7y_6)+(a_7+a_5+a_2+a_1)(x_0y_7) \\
& +(a_7+a_6+a_4+a_1+a_0)(x_1y_7)+(a_7+a_6+a_5+a_3+a_0)(x_2y_7) \\
& +(a_6+a_5+a_4+a_2)(x_3y_7)+(a_7+a_5+a_4+a_3+a_1)(x_4y_7) \\
& +(a_7+a_6+a_4+a_3+a_2+a_0)(x_5y_7)+(a_7+a_6+a_5+a_3+a_2+a_1)(x_6y_7) \\
& +(a_6+a_5+a_4+a_2+a_1+a_0)(x_7y_7) \\
z_0 = & (a_0)(x_0y_0)+(a_7)(x_1y_0)+(a_6)(x_2y_0)+(a_5)(x_3y_0)+(a_4)(x_4y_0) \\
& +(a_7+a_3)(x_5y_0)+(a_7+a_6+a_2)(x_6y_0)+(a_6+a_5+a_1)(x_7y_0) \\
& +(a_7)(x_0y_1)+(a_6)(x_1y_1)+(a_5)(x_2y_1)+(a_4)(x_3y_1)+(a_7+a_3)(x_4y_1) \\
& +(a_7+a_6+a_2)(x_5y_1)+(a_6+a_5+a_1)(x_6y_1)+(a_5+a_5+a_4+a_0)(x_7y_1) \\
& +(a_6)(x_0y_2)+(a_5)(x_1y_2)+(a_4)(x_2y_2)+(a_7+a_3)(x_3y_2) \\
& +(a_7+a_6+a_2)(x_4y_2)+(a_6+a_5+a_1)(x_5y_2)+(a_5+a_5+a_4+a_0)(x_6y_2) \\
& +(a_6+a_4+a_3)(x_7y_2)+(a_5)(x_0y_3)+(a_4)(x_1y_3)+(a_7+a_3)(x_2y_3) \\
& +(a_7+a_6+a_2)(x_3y_3)+(a_6+a_5+a_1)(x_4y_3)+(a_5+a_5+a_4+a_0)(x_5y_3) \\
& +(a_6+a_4+a_3)(x_6y_3)+(a_5+a_3+a_2)(x_7y_3)+(a_4)(x_0y_4) \\
& +(a_7+a_3)(x_1y_4)+(a_7+a_6+a_2)(x_2y_4)+(a_6+a_5+a_1)(x_3y_4) \\
& +(a_5+a_5+a_4+a_0)(x_4y_4)+(a_6+a_4+a_3)(x_5y_4)+(a_5+a_3+a_2)(x_6y_4) \\
& +(a_7+a_4+a_2+a_1)(x_7y_4)+(a_7+a_3)(x_0y_5)+(a_7+a_6+a_2)(x_1y_5) \\
& +(a_6+a_5+a_1)(x_2y_5)+(a_5+a_5+a_4+a_0)(x_3y_5)+(a_6+a_4+a_3)(x_4y_5) \\
& +(a_5+a_3+a_2)(x_5y_5)+(a_7+a_4+a_2+a_1)(x_6y_5)+(a_6+a_3+a_1+a_0)(x_7y_5) \\
& +(a_7+a_6+a_2)(x_0y_6)+(a_6+a_5+a_1)(x_1y_6)+(a_5+a_5+a_4+a_0)(x_2y_6) \\
& +(a_6+a_4+a_3)(x_3y_6)+(a_5+a_3+a_2)(x_4y_6)+(a_7+a_4+a_2+a_1)(x_5y_6) \\
& +(a_6+a_3+a_1+a_0)(x_6y_6)+(a_7+a_5+a_2+a_0)(x_7y_6) \\
& +(a_6+a_5+a_1)(x_0y_7)+(a_5+a_5+a_4+a_0)(x_1y_7)+(a_6+a_4+a_3)(x_2y_7) \\
& +(a_5+a_3+a_2)(x_3y_7)+(a_7+a_4+a_2+a_1)(x_4y_7) \\
& +(a_6+a_3+a_1+a_0)(x_5y_7)+(a_7+a_5+a_2+a_0)(x_6y_7) \\
& +(a_7+a_6+a_4+a_1)(x_7y_7)
\end{aligned}
$$

6.3 Fitting the Bit-Equations Together

After deriving the formulae for transforming individual terms in an equation, the last step is to fit them together to form an equation system which is equivalent to the original byte-system. The indexing must be consistent in order to allow the solution to be both possible (term-indexing is consistent throughout the bit-equations) and intelligible (term-indexing is arranged such that, when the solution bit-vector is written out, every 8-bit block represents an actual byte of the solution byte-vector of the original system, rather than a random distribution of solution bits). We have used exponential indexing, details of which are explained in Section 12.4.

Our indexing system has made this transition rather easily computable:

- The bits of the *constant term* of the nth equation diffuse to the transformed bit-system as the constant terms of the $(8n + i)$th equations $(0 \leq i \leq 7)$.
- For *linear terms* , the index of the byte-variable needs to be known in addition to the equation number. The bits of a single linear term x_k of equation n diffuse to all equations $8n + i$, where they occupy indices 3^{8k+j} $(0 \leq i, j \leq 7)$.
- *Quadratic terms* diffuse to a larger area in the bit-system, occupy a larger number of indices and require many more XOR operations. For a quadratic term, the indices of both variables need to be known in addition to the equation number. For the quadratic term consisting of variables x_k and x_m in equation n, the bits diffuse to all equations $8n+i$ $(0 \leq i \leq 7)$. They occupy all 64 indices $3^{8k+j} + 3^{8m+l}$ $(0 \leq j, l \leq 7)$.

7 XL

7.1 The Algorithm

The eXtended Linearization attack was presented in [5] by Nicolas Courtois, Alexander Klimov, Jacques Patarin and Adi Shamir at the EuroCrypt 2000 conference. Being the first algebraic attack to be suggested against Rijndael (even before it became the official standard), it aroused great interest from the cryptology community.

The idea was based on Jacques Patarin's Linearization method [20]. Showing that breaking Rijndael was equivalent to solving a multivariate quadratic

equation system, they set out to develop a method to solve these MQ systems efficiently. Furthermore, they have observed that the equation systems that can be used to fully express the encryption process of the cipher were overdefined (more equations than unknowns), an observation which they used for the basis of their suggestion that this characteristic may make it possible to develop a method to solve these systems faster than a brute-force search.

In order to linearize a quadratic equation system, all one needs to do is to assign a new variable name to each different monomial in the system and apply linear techniques to the relinearized system. This, however, presents a problem. Assigning new variable names to each different monomial is likely to produce a linearized system with more variables than equations; in short, not enough linearly independent equations to apply linear techniques.

Let's assume the following MQ system over GF(2), which, with 2 variables and 2 equations, is as small as MQ systems get:

$$\begin{aligned} 0 &= x^2 + xy \\ 1 &= xy + y^2 \end{aligned}$$

This equation system has 2 variables and sufficient independent equations as to yield a unique solution ($x = 0$ and $y = 1$). However, finding this solution requires a brute-force search, hence the need to Linearize (the "L" of "XL"):

$$\begin{aligned} 0 &= a + b \\ 1 &= b + c \end{aligned}$$

Having relinearized the system, now we have a linear equation system. The problem is that we only have 2 equations for 3 variables and cannot apply linear techniques to solve it, as it has more than one unique solution. The reason for this is that we now consider a, b and c as independent variables, whereas in the original system b was dependent on both a and c.

The solution to overcome this obstacle is to generate new linearly independent equations while preserving the correctness of the system and the uniqueness of the solution set. To achieve this, we need to find a way to eXtend the system (the "X" of "XL"). Courtois et al. have suggested a general method called D-linearization: an integer D is picked ($D > 2$), and all monomials of degree $D - 2$ are listed. Then, each equation in the system multiplied by each one of these monomials, each multiplication producing a new equation which preserves the solution(s) of the system.

Let's assume that, for the equation system above, we have picked $D = 4$. We list all monomials of degree 2:

$$x^2, xy, y^2$$

Then we multiply both equations by each of these monomials, and while keeping the original equations, we generate 6 new equations:

$$\begin{aligned}
0 &= x^2 + xy \\
1 &= xy + y^2 \\
0 &= x^4 + x^3y \\
0 &= x^2 + x^3y + x^2y^2 \\
0 &= x^3y + x^2y^2 \\
0 &= x^2y^2 + xy^3 \\
0 &= x^2y^2 + xy^3 \\
1 &= xy^3 + y^4
\end{aligned}$$

If we linearize this system now, we obtain

$$\begin{aligned}
0 &= a + b \\
1 &= b + c \\
0 &= d + e \\
0 &= a + e + f \\
0 &= e + f \\
0 &= f + g \\
0 &= f + g \\
1 &= g + h
\end{aligned}$$

In its extended and linearized form, the equation system now has 8 *linear* equations with 8 *independent* variables; it is now solvable by linear techniques. However, there is one more problem to overcome. If we use a linear technique to evaluate all variables in the system, then we would be doing a lot of extra work to solve for 8 variables whereas we only need to solve for 2. To avoid this extra work, we only solve the system for one variable, then substitute its value into the original equation system and work with a diminished system. We repeat the process until all variables in the original system are evaluated.

7.2 Analysis of the Algorithm

7.2.1 Moh's Analysis of XL on TTM

XL is an algorithm for solving MQ equations systems and can be applied to any problem which can be expressed as such a system. In 2001, Tzuong-Tsieng Moh made an analysis of XL applied to the Tame Transformation

Method (TTM) [16], a public key cryptosystem developed by himself and described in [17]. XL is applied to TTM under three different conditions, with three different values of D at each case. The three conditions are defined by setting the n and m variables; m defining the finite field the cipher is working in, and n defining the dimension of the affine space. The complexities are calculated by estimating the actual number of operations involved in multiplication and linear elimination:

n	**m**	**D**	**complexity**	**D**	**complexity**	**D**	**complexity**
64	100	8	2^{100}	13	2^{142}	14	2^{149}
44	80	7	2^{80}	9	2^{96}	10	2^{103}
40	72	6	2^{69}	8	2^{85}	10	2^{99}

Computations for the chosen values of D indicate that XL is as slow as or slower than a brute force search on TTM.

7.2.2 Diem's Analysis of XL

In 2004, Claus Diem has provided a more general analysis of the XL algorithm when applied to any MQ system over any finite field [8]. The analysis has produced complexity bounds as to the running time of the algorithm and the bounds contrast strongly with the initial claims of the developers of XL. The outcome of Diem's work was that, for a generic overdefined MQ system with a unique solution over a finite field, XL was not subexponential, and that the assumptions made by the developers of XL were too optimistic. The analysis is based on Moh's work which was briefly discussed in the previous section.

8 XSL

8.1 The Algorithm

The eXtended Sparse Linearization method [5], introduced in 2002 by Nicolas Courtois and Josef Pieprzyk, has its roots in the XL algorithm which was described in the previous section. They have tried to address some shortcomings of XL and to make it a more efficient method that is at least as good at solving MQ systems as XL is.

An additional observation that was made on the MQ systems of Rijndael, that they were sparse as well as overdefined, helped them to improve on XL, which only used the overdefined characteristic of those systems. Multiplying equations with every possible monomial of a given degree produced so many

new equations with so many new variables that the equation system became intractable very quickly. Observing that the equation systems were actually sparse, Courtois and Pieprzyk suggested that equations are multiplied only by *"carefully selected monomials".* The idea is to generate a smaller number of new equations and new variables and still make the extended and linearized system solvable by linear methods. The phrase "carefully selected monomials", however, is ambiguous, and therefore open to interpretation, which is why several variants of XSL have sprouted up since the original paper. We have opted to follow the suggestion in the original paper that equations are only multiplied by monomials that already appear in other equations. This decision implicitly restricts the XL method such that the value D always has to be 4, since the monomials that exist in other equations are quadratic. Given that the equation system is sparse to start with, this restriction also assures that the extended system is much smaller, both in width and in length, than the one produced by XL, which makes it much easier to solve.

Let us consider the equation system in the previous section:

$$\begin{aligned} 0 &= x^2 + xy \\ 1 &= xy + y^2 \end{aligned}$$

Whereas for XL we would have to multiply both equations by all of x^2, xy and y^2, for XSL we only need to multiply the first equation by xy and y^2, and the second one by x^2 and xy:

$$\begin{aligned} 0 &= x^2 + xy \\ 1 &= xy + y^2 \\ 0 &= x^3y + x^2y^2 \\ 0 &= x^2y^2 + xy^3 \\ 0 &= x^2 + x^3y + x^2y^2 \\ 0 &= xy + x^2y^2 + xy^3 \end{aligned}$$

When relinearized, this system is expressed as:

$$\begin{aligned} 0 &= a + b \\ 1 &= b + c \\ 0 &= d + e \\ 0 &= e + f \\ 0 &= a + d + e \\ 0 &= b + e + f \end{aligned}$$

The resulting system has 6 linearly independent equations and 6 independent variables, thus is solvable by linear methods. The following steps are the same as they were in XL; one variable is evaluated and substituted into the original equation system, and the whole process is repeated until all variables in the system have been evaluated.

Appendix B contains a working example of XSL on a slightly larger system. Also, the program `EquationWriter` can be run for observing the algorithm in action on systems of any desired sizes; it simply runs through the first round of extending, linearizing, solving and also converting and prints the output in an intelligible format.

8.2 Analysis of the Algorithm

8.2.1 Analysis of XSL by C. Cid and G. Leurent

At the Asiacrypt conference in 2005, Carlos Cid and Gaetan Leurent presented their analysis of the XSL algorithm, which is the most recent comprehensive study of XSL to date [3]. The conclusion they arrived at was that XSL *"in its current form... does not provide an efficient method for solving the AES system of equations."* The fact that they write the word "algorithm" in quotation marks when mentioning the "XSL algorithm" summarizes their views on XSL; it is not guaranteed to terminate for every MQ system, and indeed they reach the conclusion that XLS is not an efficient attack against Rijndael *"in its current form."* Although they do not say it explicitly, that phrase implies that they cannot say with similar certainty that there is no way to modify XSL to produce an efficient method to solve MQ systems.

8.2.2 Song-Seberry Paper

Dated 2003, Beomsik Song and Jennifer Seberry of the University of Wollongong conducted a study and compiled a report on the security of Rijndael based on their observations. They picked up where previous studies, especially Murphy and Robshaw [18], left off and made further observations on the implications of what has been suggested of the algebraic structure of the cipher. They state that although the cipher is built from simple components, the overall structure of the cipher appears to be more complex than initially anticipated. Although they declare Rijndael's well-designed combination of linear and non-linear layers with the key schedule strong enough to resist potential algebraic attacks, they also acknowledge XSL as a *"research study [that] has been making considerable progress in the cryptanalysis of AES-like*

block ciphers."

8.2.3 Essay by B. Schneier

Bruce Schneier is a cryptography expert, the designer of several ciphers among which is Twofish (a rival of Rijndael in the AES contest) and publishes Crypto-Gram, a monthly combination of cryptography-related essays. In the heyday of XSL, he has published an essay [21], dated September 15, 2002, on the perceived implications of the XSL attack for which it was claimed that it could break Rijndael, at least theoretically. This being an informal discussion rather than a scientific study, Schneier shies away from making any conclusive statements, and summarizes his views as *"AES may have been broken. Serpent, too. Or maybe not. In either case, there's no need to panic. Yet. But there might be soon. Maybe."*

More than 3 years after the essay, Schneier has expressed similar views at a lecture at the Rochester Institute of Technology. His answer has confirmed the vague idea in almost everyone's mind that although Rijndael looks firm and secure against the attacks that currently exist, it may indeed be possible to exploit its algebraic structure. *"Rijndael was the risky choice,"* recalls Schneier, regarding this algebraic structure and reiterates that more research is required on algebraic attacks to make a firm statement.

8.3 The XSL variant

In an effort to demonstrate the trade-off between the solving probability and the speed/memory requirements, we have also used an XSL variant in our experiments. The variant was aimed to make more MQ systems solvable at the expense of speed due to the increase in equation size during the extension phase. The difference in speed of the two algorithms was observed to get more insignificant as the system size grew, and the variant did not prove to be a stronger solver than XSL.

Whereas XSL multiplies equations only with quadratic terms that appear in other equations, the XSL variant uses linear terms in addition to quadratic terms for multiplication. The initial projection was that this would create few new monomials and a greater number of linearly independent equations, thus making a portion of the MQ systems solvable. The obvious downside is that this algorithm now generates extra-large extended systems, which is not desired when the physical constraints are already so severe.

9 Similar Research on XSL

9.1 Evaluating Algebraic Attacks on AES

In his work [22], Ralf-Philipp Weinmann focuses on an evaluation of the XL and XSL algorithms based on equation systems derived from a very much reduced version of Rijndael, called Mini-Rijndael. The block size of Mini-Rijndael is 16 bits and uses 1 round of encryption for the experiments with XSL. This reduction of size required the building blocks of the finite field to be re-defined; a new irreducible polynomial had to be chosen for multiplication, a new S-box had to be designed, etc. While this new design essentially forced the experiments to work with a cipher that is actually different from Rijndael, it can more or less be viewed as having a similar structure to Rijndael while making the equation systems small enough to conduct experiments with. We have decided to use random equation systems so that we would have more control over the structure of the systems and would be able to operate over a wider range of systems, in addition to the assumption that using any cipher other than Rijndael, albeit designed as a model of Rijndael, would be as good as a random equation system as far as acquiring real Rijndael results was concerned; Weinmann's decision to define a baby cipher in the likeness of Rijndael, however, enables us to see how the XSL algorithm work if Rijndael had indeed been a small cipher, and therefore is a valid model for observing the algorithm. Indeed, it was the existence of his work that, to a certain extent, made us prefer to use random and more general equation systems rather than using a Mini-Rijndael and producing a carbon copy of his analysis.

The equation systems generated are over GF(2) and it is concluded that XSL on these equations is *"unlikely to succeed."* The problem arose when trying to generate sufficient linearly independent equations; for a 3-bit S-Box and one round of encryption, the extended system had 4566 equations containing 4627 terms, and was therefore 60 equations short of what is required in order to have a solution. For two rounds of encryption, the gap between the number of equations and terms grew further and became 400.

9.2 XL and XSL attacks on Baby Rijndael

Presented in [13], Elizabeth Kleiman undertakes a similar task of constructing a small Rijndael-mode, called the "baby Rijndael", and tests the XSL algorithm on this cipher. The Rijndael model uses a block size of 16 bits and for the experiments, 1 and 4 rounds of encryption are used by default. The

S-Box and other components of the cipher need to be re-defined to accommodate the reduced block size.

Differently from Weinmann's work, Kleiman experiments with XL in addition to XSL. Equations are generated following the standard Murphy-Robshaw method and they are processed with both methods. For 1 round, the equation system that has been generated and experimented with contains 150 equations and 24 bit-variables (corresponds approximately to 19 equations with 3 byte-variables in our experiments). 4 rounds of encryption generates a system with 600 equations and 96 bit-variables (equivalent to a 75-equation system with 12 byte-variables).

The shortcoming of this work is that, even though the derivation of equations is valid, the equation system is only extended once and not solved. The conclusions, therefore, are reached by speculating on the implications of the sizes of the systems that are generated by the extension methods of both algorithms. The conclusion is that, based on the extended system sizes, XL seems to work on one round of Baby Rijndael and both XL and XSL work (with the hopeful assumption that a sufficient number of those equations are linearly independent, an assumption which is not verified) on four rounds of Baby Rijndael.

10 Implementation of XSL

10.1 What We Did

There were some initial decisions to be made as to how to proceed with the XSL algorithm. The reason for the birth of XSL was to cryptanalyze Rijndael and Serpent, and any other block ciphers satisfying certain criteria as described in [6]. Our starting point for this work was Rijndael. Using equation systems from the full-sized cipher, as is confirmed by everyone else who has studied the attack, is impossible, due to the sheer size of the equation system. One way to go was to still use Rijndael, but make only one round of encryption to produce the equation systems. Even this was too big to handle. At this point, we faced a two-way choice: either to reduce the cipher further (i.e. reduce the block size), or to seek a more general analysis of the cipher by using random equation systems.

There were a couple of problems about working with a reduced version of Rijndael. First, it was already done [13] [22]. Second, as far as we were

concerned, as long as we did not use the real Rijndael, the results we would gather for any other cipher, even if it was a model of Rijndael, would be no better than using random equation systems. If we were to analyze XSL's behavior on Rijndael, then we had to use the real Rijndael, which was out of the question. Reducing only the number of rounds would have been acceptable because we would still be using the same S-box, same finite field and same arithmetic as the real Rijndael, but even that fell outside the computing resources available to us. Once the cipher was reduced further and the block size was changed, the very essence of the cipher would have to change since we would now be working in a new finite field, would need to design a new, reduced S-box and define our own irreducible polynomial for doing arithmetic in this field. Going down this path would have enabled us to answer the question "How would XSL work if Rijndael was a smaller cipher?"; however, the question "How does XSL work on Rijndael?" would remain unanswered.

Using random equation systems had its own advantages and disadvantages. The main disadvantage is rather obvious: if we use random systems, we lose contact with our starting point, Rijndael. Although this sounds like a major deficiency, we believed that it would be overshadowed by the benefits which otherwise would not have been possible. To start with, we would have a greater freedom to change certain parameters in equation systems, giving us the power to observe the behavior of the algorithm in many different situations. For instance, we have had the opportunity to observe the impact of the number of variables to the speed of the algorithm, the effect of the number of variables on its speed as well as functionality, and the result of modifying the sparsity of the system on the behavior of the algorithm. Had we used equation systems derived from a certain cipher, we would only have equation systems of a certain type, a certain combination of the parameters we have just mentioned. With a more general design, we have had the opportunity to observe some claims made by the designers of the algorithm, such as the algorithm preferring overdefined (this was tested by the "number of equations" and "number of variables" parameters) and sparse (this was tested by the "sparsity" parameter) equation systems. While doing that, we have designed our system such that it was possible to use the algorithm with manually written equation systems as well. The implication of this is that, if our program was fed with a real Rijndael-driven equation system, then the behavior of the algorithm could still be observed. Needless to say, one would need a considerable amount of memory and processing power to be able to do this, but it still gives our design the power to perform any kind of test with it.

Another major obstacle to overcome was, since our primary objective was to compare the performance of the algorithm in two different finite fields, to derive the formulae with which to transform an equation system in bytes to a system in bits. Using random equations for both fields would not do; the algorithm had to work on two different representation of the same system. Therefore, once a byte-system was generated (either randomly or by reading from a file) and XSL was performed on it, it had to be transformed into its bit-system representation. For deriving the formulae, the definition of multiplication in Rijndael's finite field was used. Dealing with constants was trivial as there was no multiplication, but for linear and especially quadratic terms, there was a considerable amount of shuffling of bits; deriving these formulae and verifying their correctness was a rather precarious and time-consuming process, as every single coefficient in the bit-system had to be formulated, then verified by a rather large number of examples, as well as verification by doing the opposite transformation for assured correctness.

Given these design decisions and implementation details, what we have accomplished can be concisely stated as the comparison of the performance of the XSL attack on bytes and on bits, comparison of XSL against a brute-force search, for a wide range of equation systems with numerous combinations of several parameters.

10.2 What We Left Out

It should be emphasized that we did not set out to break the Rijndael cipher as that is beyond the scope of this project. Our aim was to test the performance of the XSL algorithm, which was in fact developed to break Rijndael and Serpent, in two different finite fields and hope to shed some light as to its behavior, and to carry out experiments to get results as realistic as possible to see which finite field it "prefers".

As is the case when dealing with ciphers and algorithms of this size, we have come across some limitations such as limited memory and processing power. Due especially to memory constraints, we have decided not to use any equation systems which are indeed derived from Rijndael or Serpent; instead, we have used random equation systems. As long as we had several knobs to modify the combination of parameters and used the same system for bytes and bits, we deemed this to be sufficient to get reasonable results as to the behavior of the algorithm. In fact, deriving and using actual equations from the pre-mentioned ciphers would be an independent project of considerable size, and could be considered as a viable prospect for any interested scholars.

11 Design of Software

At the center of everything is the class `MainTester`. This class includes the main program which creates a tester object to carry out the required tasks. For each requested task (XSL on bytes, XSL on bits or Brute-Force Search), the tester delegates the necessary actions to the `EquationSystem` object that it has a reference to. The `EquationSystem` is simply a collection of `Type` objects (the variables) which are stored in `HashMaps` (the equations), which are in turn stored in an `ArrayList` (the equation system). For XSL on bytes, the byte-system is created and the `MainTester` object calls the multiply, linearize, solve and substitute methods on the `EquationSystem` until the system is solved. The time for the whole operation is recorded and printed out. For a brute-force search, a `BruteForce` object is created by the `MainTester` and all work regarding the creation of an initial solution vector and iteratively advancing it until a valid solution is found is delegated to the `BruteForce` object. Again, the time for the whole operation is recorded and printed out. For XSL on bits, the system first needs to be transformed. A `FieldTransformer` object is given a reference to the byte-system; it transforms the system to its equivalent bit-system and returns the bit-system to the `MainTester`. The tester than performs the same solving pattern that is performed on the byte-system. The whole operation of solving the system is recorded and printed out. The transformation process is not included in the measured time since it is not a part of the solving process. Figure 1 includes a class diagram and Figure 2 shows an interaction diagram.

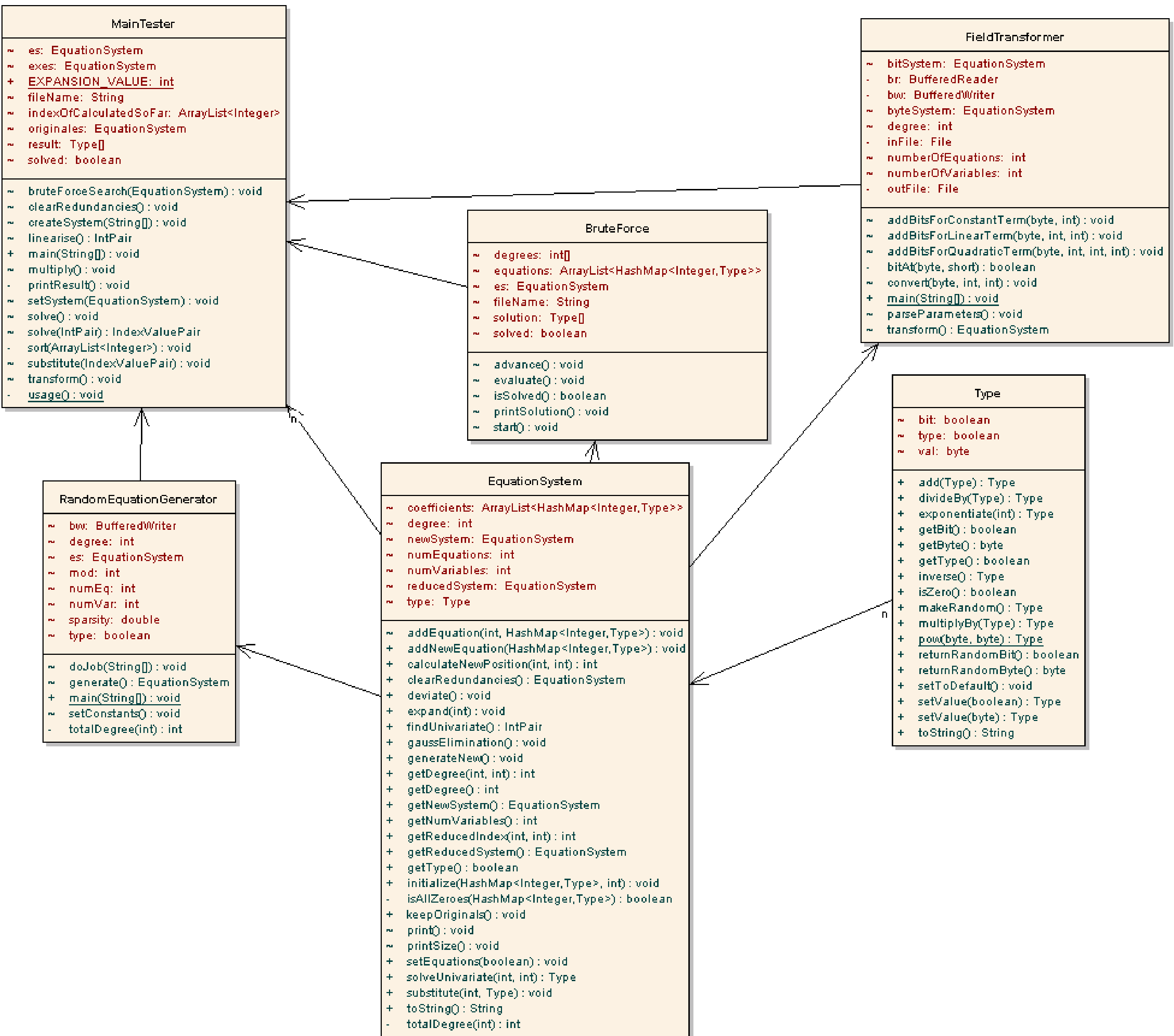

Figure 1: Class Diagram

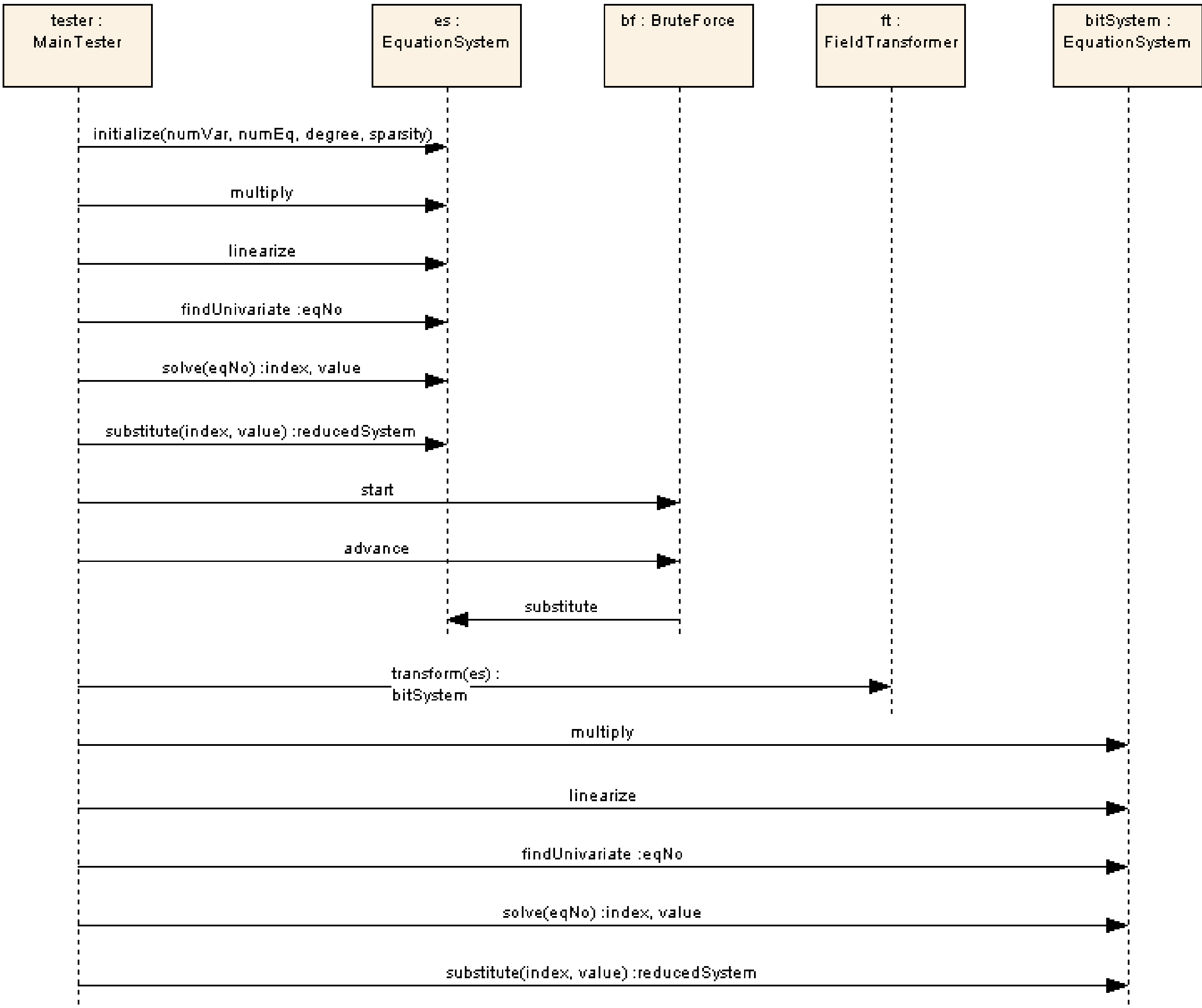

Figure 2: Interaction Diagram

12 Equations

The equation systems used in this project were generated randomly and depend on several parameters.

12.1 Number of Equations

A significant parameter for the XSL algorithm is the number of equations in the system. The algorithm is specifically aimed at overdefined systems; in fact, while a general solution would require as many linearly independent equations as the number of variables, this does not always suffice for XSL.

12.2 Number of Variables

The number of variables determines the actual size of the system and has an especially big impact on the speed and memory requirements of the algorithm considering the number of variables in the extended equation system. Considering that the equation system is always quadratic to begin with, then the possible number of terms in a non-sparse equation with n is $\frac{(n+1)(n+2)}{2}$. When extended by multiplication, the degree becomes 4 and the number of terms is now $\frac{(n+1)(n+2)(n+3)(n+4)}{24}$. This is a huge growth (for $n = 5$, it grows from 21 to 126), therefore the number of terms is the factor that affects the speed of the algorithm most, as well as being the number one factor in creating memory constraints.

12.3 Sparsity Ratio

The XSL algorithm was specifically aimed at systems that were sparse as well as overdefined so we have decided to keep sparsity as a parameter and experiment on how big its impact is on the behavior of the algorithm. Obviously, the sparser the system is, the smaller the extended system is, so the faster the algorithm becomes. However, especially for systems that are not quite overdefined, sparsity can be a problem. Since there are not many terms that appear in the equation and the number of equations is not very big, the extended and relinearized system, formed by adding new equations to the system which are generated by multiplying existing equations with all other existing terms, may not contain enough linearly independent equations for the algorithm to work. The sparsity of a system has a visible negative impact on the probability of solvability of the system and a positive effect on the speed of the algorithm provided that it is solvable.

12.4 Indexing of Variables

The indices of variables in equations were arranged in a particular way in order to provide ease of calculation. When the index of a term in an n-degree equation is written $\pmod n$, each digit of this $\pmod n$ expression gives the exponent of the corresponding variable in the term. For instance, in a quartic equation with 3 variables, the 22nd term in an equation would be 112 in $\pmod 4$, therefore would contain the coefficient for the term $x_2x_1x_0^2$. For an equation system of n degrees containing m variables, this representation will allow terms of up to $n*m$ degrees; this is not a problem, since at the time of creation, such terms of higher degrees are given 0 as a default coefficient and consequently not stored in the system. If such terms were allowed to exist in the system, the algorithm would still work, but the degree of the equation would no longer be n.

This indexing system provides ease of calculation for

- extending the system. For multiplying two terms, the index of the product should be calculated. This is easily done by writing them $\pmod n$, adding the corresponding exponents and calculating the new value. For instance, in a quadratic equation with 3 variables, the terms x_2x_0 and x_1^2 are found at indices 10 and 6 (101 and 020 as written $\pmod 3$) respectively. To get the index of the product, all one has to do is add the indices and find 121. Remember, however, that the new equation system is a product of a quadratic system with quadratic terms; therefore, the new degree of the system is now 4 instead of 2, and the index 121 of the product term is $\pmod 5$, not $\pmod 3$. Therefore, the index of the product in the `HashMap` that represents the equation is 36. The coefficients of the terms x_2x_0 and x_1^2 are multiplied and the result is stored at index 36 in the new equation, which represents the term $x_2x_1^2x_0$.

- re-indexing the existing terms in the system. At the extension stage, in addition to producing additional equations, the original equation systems are to be preserved as well. However, since the degree of the equation system changes during the extension, the terms in the original equation need to be re-indexed. All that needs to be done for this is to write the original index $\pmod n$ (n being the degree of the original system), then treat it as a number $\pmod N$ (N being the degree of the extended system) and re-calculate the index of the term, then move its coefficient to that index. For instance, for extending the term x_2x_0 in a quadratic equation system with 3 variables, the index is written

(mod 3), which is 101. Then this number is considered (mod 5), and the new index is calculated as 26.

- identifying univariate equations. After the extension comes the linear elimination, after which the algorithm needs to identify a univariate equation in the system and solve it. For doing this, one only needs to make sure that a pseudo-univariate equation (one that looks univariate in the relinearized system, but may contain more than one term of the original expression) contains only one original variable. This is simple; write the index (mod n) and check the number of non-zero digits. If there is only one non-zero digit, then the equation is univariate and can be solved.

13 Main Modules

The main test program, `MainTester`, contains a high-level description of the XSL algorithm. It conceives an equation system in GF(2^8) by either random generation or reading from a file. All of the following steps are optional and include solving the system with XSL or brute-force search, then transforming the system into GF(2) and solving it with XSL.

13.1 Equation File Format

The first line of the equation file contains information about the nature and size of the equation system. The first parameter is the type of the system (0 for bits, 1 for bytes). The second one is the number of equations in the file. The third one is the number of variables in the system. The fourth and last parameter is the degree of the system. All equations used in this project are of degree 2, but the option to use a different degree was provided for flexibility.

All following lines (except for those containing "//" anywhere in them, which are interpreted by the file parser as comments) are equations. An equation starts with the constant, then moves on to the coefficients for variables. The variables are arranged in increasing order of variable name and degree.

Example:
In an equation system with two variables, the line

17 124 80 52 10 0 -71 0 0

is interpreted as

$$17 = 124x_0 + 80x_0^2 + 52x_1 + 10x_0x_1 + 0x_0^2x_1 + 185x_1^2 + 0x_0x_1^2 + 0x_0^2x_1^2$$

It is clear that the original file contains 0's for cubic or quartic terms which will not be used. This representation would be very inefficient if left like this; therefore, once the equation system is created, all zero-elements are deleted from the equation system. Not only does this get rid of higher degree terms which will not be used, but also it may clear entries for variables which don't exist in the system due to its sparsity. Saving this space is essential in the functioning of the algorithm on a machine with memory restrictions.

13.2 Random Equation Generator

The `RandomEquationGenerator` is fed with a few arguments from the command line: the name of the file to print the system after generation, the type of the variables (0 for bits, 1 for bytes), the number of equations, the number of variables, the degree of the equation system, the probability of a coefficient being zero (the sparsity ratio) and the actual values of the variables in the system. Random coefficients are generated for linear and quadratic terms in the equation system. Based on the values of the variables, the constant terms are calculated. The equations are then written to file and the program terminates.

13.3 Main Program

The main program starts by reading a multivariate quadratic equation system (in bytes) either from a file whose name is passed as a command line argument or from a random equation generator. Once the equation system is created, XSL is performed on it by calling multiply, linearize, solve and substitute methods of the equation system until all variables have been evaluated. The execution time is recorded. Then brute-force search is applied and the time is recorded again. Finally, the original equation system is converted to its bit-representation using the Field Transformation Formulae derived in Section 6. XSL is performed on the new system, all variables are evaluated and the execution time is recorded. The execution times for all three methods are printed on standard output and the program terminates.

All three steps in the main program are optional. The fifth and last command line argument is used to see if any of these steps are going to be performed; if it contains

- 'B', then the byte-system will be solved by XSL;

- 'X', then the byte-system will be solved with by a brute-force search;
- 'b', then the transformation will take place and the bit-system will be solved by XSL.

13.4 Equation System

The `EquationSystem` class contains implementations of all algorithms that are used by XSL to solve the MQ system. The system itself is stored as an `ArrayList` of `HashMaps`, each `HashMap` representing one equation. Each `HashMap` entry represents one term in the equation; the key is the index of the term in the equation, and the value is the coefficient of the corresponding term. This design, as opposed to our original design of an `ArrayList` of `ArrayLists`, has allowed us to discard all zeroes from the system and only keep track of non-zero coefficients through their indices, saving valuable space to allow experiments with much larger equation systems. This has come in handy especially when dealing with sparse systems, which are what the XSL algorithm is mainly designed to operate with. Some functionalities that are implemented in this class are:

- *Extension by multiplication:*
  One of the major sections of the XSL algorithm is to extend an equation system. This is done by multiplying each equation by every quadratic term that appears in every other equation. There are two steps in performing the extension: keeping the original equations and generating the new ones. Keeping the original equations consists of rearranging the indices of terms as they appear in the `HashMap`. This is necessary because the degree of the equation system increases and each term should now be represented with a different index. Generating new equation system involves a bit more work. All equations are iterated through; then for each equation, every other equation is taken, and every quadratic term in that other equation is used for multiplying with the first equation at hand and thus generating a new equation. The multiplication consists of multiplying the coefficient values as well as determining the new index of each variable based on the indices of the original multiplicants. For instance, let's consider the multiplication of terms $40xy$ and $67y^2$ in a 2-variable quadratic system. The product should be $150xy^3$. The original terms are represented as $(4, 40)$ and $(6, 67)$ in the original system. The indices 4 and 6 are represented as 11 and 20 in $\pmod 3$ (remember that this is a quadratic equation, so the exponents of terms are written $\pmod 3$); therefore, the index of the product should be the addition of the original indices, which is

31 as written in   (mod 5). This makes the new index 16. The value at index 16 is calculated by multiplying the original coefficients and is 150. Therefore, the new entry in the new `HashMap` is $(16, 150)$.

- *Gauss-Jordan Elimination:*
  Linear elimination is provided in the form of Gauss-Jordan elimination, although it is possible to easily convert it to Gaussian elimination by commenting in some sections of the code. The elimination takes place by iterating through every possible index, finding an equation which contains a term with the index that can be used as a pivot element (i.e. the equation contains a term with the given index, and contains no other terms with a lower index except for the constant) and using it to eliminate all other equations which also contain a term with the same index. The elimination continues until all terms are iterated through. The XSL algorithm expects this stage to produce at least one univariate equation; the equation systems for which this does not happen are said to be unsolvable by XSL. Remember that the algorithm is aimed at specific types of quadratic equation systems and is not a general MQ solving algorithm.

- *Identifying and solving univariate equations:*
  The term "univariate" is used in a different sense than it would mean in a usual linear equation system. Since the system is relinearized, a univariate equation in the modified system may be multivariate in the original system. For instance, the equation

  $$5 = 6x_0x_1$$

  would appear to be univariate in the relinearized system since it would appear as

  $$5 = 6a$$

  but should be discarded by the algorithm as multivariate, since it contains both x_0 and x_1 when expressed in the original equation. This is where the indices of terms in the `HashMap` come in handy, and they are used in order to identify whether a univariate equation in the modified system is also univariate in the original system. Once identified, they can be solved by iterating through every possible byte value until a solution is found.

- *Substitution:*
  Once a variable is evaluated, it has to be substituted in the equation system in order to reduce the system and take one more step towards acquiring the final solution. In addition to knowing the value of the variable, also the index needs to be known so that it can be substituted at the right place. There are three possibilities for the substitution: if a term is univariate on the evaluated variable, then the resulting term is a constant; if it does not contain the evaluated variable, then no action is taken except for recalculating the new index; if it is multivariate and contains the evaluated variable, then the required multiplications are made based on the degree of the variable in the term, and a new index is calculated for the variable. For instance, let's assume that we have a 3-variable quadratic equation

  $$55 = 70x_0 + 6x_1 + 213x_1^2x_2$$

  and variable x_1 is evaluated as 21. The reduced equation should look like

  $$73 = 70x_0 + 67x_2$$

  To achieve this reduced form, first the terms that contain x_1 should be identified. They are those terms that have a non-zero second digit in the $\pmod 3$ representation of the indices:

  $$((000, 55), (001, 70), (010, 6), (120, 213))$$

  The value of these second digits indicate the exponent of x_1 in the equation. The necessary multiplications are then made and the new indices are calculated by removing the second digits from the indices:

  $$((00, 55), (01, 70), (00, 126), (10, 67))$$

  The terms with the same indices are then combined by simple addition as defined for the finite field we are working in, and the final representation of the reduced system is

  $$((0, 73), (1, 70), (3, 67))$$

- *Clearing redundancies:*
  Coefficients that are zeroes are not stored in the system in order to save space. All existing terms are iterated through, and all zero coefficients are removed. If an equation is formed entirely of zeroes, then that equation is removed from the system as well. This can happen either at the creation of the equation system, or after substitution in a univariate equation.

13.5 Variables

Variables in an equation system are instances of the class `Type`. The value contained in the object can either be a byte or a boolean. The class contains the main algebraic operations over both GF(2) and GF(2^8). Some functionalities that are implemented in this class include:

- *Addition:*
  Bit-by-bit addition is performed on the variables.
- *Multiplication:*
  Multiplication as defined in Section 4.1.
- *Multiplicative inverse:*
  The inverse of a variable is used for linear elimination.
- *Division:*
  Defined as multiplication with a variable's multiplicative inverse, division is also used for linear elimination.
- *Exponentiation:*
  Consists of successive multiplications to find powers of variables.

13.6 Brute Force Search

Class `BruteForce` includes a brute-force search for a solution vector. The algorithm simply iterates through the solution space and checks every possible vector of values for being a solution to the system. As soon as a solution is encountered, or if the solution space is exhausted without finding a solution, the algorithm terminates. This means that a solution is not checked for uniqueness. The behavior of the brute-force search is predictable and the speed of the search increases exponentially with the number of variables.

13.7 Field Transformer

Class `FieldTransformer` contains the lengthy field transformation formulae. An instance of this class contains a byte-system and derives an equivalent bit-system from it. The process consists of taking one variable at a time, deciding whether it is a constant, linear or quadratic term, and doing the necessary operations as defined by the formulae and setting the necessary bits in the bit-system. If the term is of a higher degree, then an error message is printed and the program quits. At the end of the process, the converted system in bits is returned.

13.8 Exceptions

Several `Exception` classes were also included in order to identify the exact causes of problems encountered during the execution of the program:

- `NotQuadraticEquationException` is an indication that the program has encountered an equation of degree 3 or higher and will quit. This may happen if an equation is entered manually or if the `RandomEquationGenerator` is malfunctioning. It does not happen after the extension because, although the equations at that stage are quartic, that is necessary for the algorithm to be functioning correctly. Therefore, this check is only performed at initialization.

- `WrongFirstLineInFileException` only occurs when reading equations from a file. The file parser expects the first line of the equation system to be in a certain format and to contain parameters such as the type of the system (byte or bits), the number of equations and variables, and the degree of the equation system (must be 2 for this implementation, but still included as a parameter in order to make it more flexible and reusable by other programs). If the number of parameters in the first line is not equal to the expected value of 4, or if one of the parameters is not a valid integer, or the equation type is specified as another value than 0 (bits) or 1 (bytes), then this exception is thrown and the program quits.

- `WrongNumberOfEquationsException` is thrown as a result of a simple check between the number of equations specified in the first line of the system and the actual number of equation lines contained in the file. If the two values do not match, then an instance of this exception is thrown and the program terminates.

- `WrongNumberOfVariablesInEquationException` is thrown as a result of a simple check between the number of terms specified in each line of equations and the expected number of terms (which is $(d+1)^n$ where n is the number of variables and d is the degree of the system). If the two values do not match, then the equation line is considered invalid, this exception is thrown and the program terminates.

- `WrongLineInFileException` only occurs when reading from a file and it handles all errors in all lines in the file except for the first line, which are handled by the `WrongFirstLineInFileException`. This exception is thrown for any other error encountered in a line in the file.

- `WrongVariableValueException` is thrown when a value in an equation line in the file does not fall into the desired ranges of the specified equation type ($[-128, 127]$ for bytes and $[0, 1]$ for bits). This is considered illegal, causes this exception to be thrown and the program to terminate.

13.9 Tools

13.9.1 Add and Mult

`Add` and `Mult` are small, algebraic tools that may be used to add and multiply a variable number of bytes in GF(2^8). They are helpful in manually checking the correctness of a solution and for manually creating an equation system. They can take an arbitrary number of arguments from the command line, perform the necessary operation and print out the result.

13.9.2 Equation Writer

Class `EquationWriter` was simply included in order to demonstrate each component of the XSL algorithm clearly. It creates a very small equation system (2 variables, 3 equations and 40% sparsity) and prints it out, not just the coefficients, but the whole equation system in an intelligible format, using x and y for variable names. The system is then extended, and the new, extended system is printed out in the same way. The system is linearized and the new system, with the new variable names from a to t (there are 24 possible non-constant terms in each extended equation) used for each different monomial in the extended system. Then Gauss-Jordan elimination is performed, and the eliminated system is printed again. Finally, the original system is converted into its equivalent bit-system and that is also printed, with the nth bits of the variables x and y being represented as x_n and y_n

respectively. In short, this little tool provides easily understandable examples of the following procedures of XSL:

- Extension
- Relinearization
- Linear Elimination
- Field Transformation

14 Experimental Results

We have run experiments on random equation systems based on parameters such as number of equations, number of variables and sparsity ratio. All equation systems were quadratic. The experiments were run on a machine with a 2 GHz Intel Pentium processor and 1 GB of RAM.

As for the main objective of our work, which is to compare XSL's performance for byte-systems and their equivalents in bit-systems, the results left no room for doubt: XSL prefers byte-systems. In fact, after a 2-variable 3-equation quadratic byte-system is transformed to GF(2), it takes a few hours just to perform the first step of the linear elimination, whereas it takes less than 0.001 seconds for the original system in GF(2^8). The transformation creates a system of 24 equations with 153 terms; that is before the multiplication and linearization. After the extension, there are 4845 variables in the system, and although the variables are all bits and the algebra is very simple, the number of operations is so huge that it already becomes impractical. Bit-systems become unmanageable much sooner than byte-systems and are extremely harder to solve. Therefore, what appear in the following subsections are experiment results that reflect XSL's performance for byte-systems.

14.1 Charts and Figures

14.1.1 XSL

Below are the results of experiments conducted on the XSL algorithm for multivariate quadratic equation systems in GF(2^8). On the left side of each row are the sparsity ratios, on top of the columns are the size of the equation systems in terms of the number of equations, and the values in the tables express the average execution times of the algorithm in seconds. Each table is for a fixed number of variables. Each value in the table is an average value

for several executions of the algorithm; from 1000 runs for smaller systems to 10 runs for larger ones. As the system gets larger, experiments yield less and less extreme values and seem to converge on average values with much lower deviation than small systems, so 10 runs were deemed sufficient. It is to be considered that all variables are bytes, so a 4-variable system means solving a system for 32 bits. A blank entry in the table means that the system for the specified parameters was unsolvable for XSL because the relinearization after extension did not produce sufficient linearly independent equations.

	5	**10**	**15**	**20**	**30**	**40**	**50**
0.0	0.005	0.007	0.010	0.016	0.034	0.063	0.098
0.2	0.004	0.006	0.008	0.012	0.023	0.042	0.068
0.4	0.003	0.005	0.006	0.0009	0.016	0.028	0.045
0.6		0.004	0.005	0.006	0.009	0.016	0.022
0.8		0.003	0.004	0.005	0.005	0.006	0.008

Table 2: XSL results with 1 variable

	5	**10**	**15**	**20**	**30**	**40**	**50**
0.0	0.042	0.176	0.468	1.035	3.537	6.746	16.484
0.2	0.034	0.134	0.314	0.607	2.815	3.762	7.125
0.4	0.021	0.0545	0.109	0.154	0.338	1.104	0.991
0.6		0.012	0.017	0.029	0.044	0.075	0.130
0.8		0.008	0.009	0.013	0.020	0.029	0.056

Table 3: XSL results with 2 variables

	5	**10**	**15**	**20**	**30**	**40**	**50**
0.0	1.078	3.406	9.245	16.047	81.578	158.563	309.468
0.2	0.922	2.953	7.102	12.894	70.516	87.985	193.969
0.4	0.828	1.812	5.391	8.462	32.313	40.937	92.453
0.6	0.719	0.850	1.227	2.219	4.881	8.450	18.587
0.8		0.019	0.023	0.031	0.059	0.103	0.134

Table 4: XSL results with 3 variables

	5	10	15	20	30	40	50
0.0		78.672	184.672	459.640	1369.515	3015.234	7291.500
0.2		69.094	178.656	413.843	1113.344	2305.859	3636.515
0.4		53.579	115.547	382.422	1083.953	1589.672	2200.688
0.6		23.984	54.172	120.843	707.125	917.375	1279.329

Table 5: XSL results with 4 variables

14.1.2 XSL Variant

Contrary to our intentions, even less equation systems were solvable in the variant than XSL, while those that were solvable were expectedly slower to solve. Therefore, it is impossible to list this as an improvement over XSL.

Below are the results of the experiments on the XSL variant described in Section 8.3. The formatting of the tables are same as the previous section.

	5	10	15	20	30	40	50
0.0	0.0089	0.0339	0.0790	0.1971	0.7744	1.1961	1.895
0.2	0.0074	0.0249	0.0564	0.1646	0.5216	0.8607	1.341
0.4		0.0174	0.0360	0.0739	0.1875	0.4950	0.578
0.6		0.0099	0.0186	0.0349	0.1013	0.1883	0.4451
0.8				0.0115	0.0231	0.0395	0.0954

Table 6: XSL-variant results with 1 variable

	5	10	15	20	30	40	50
0.0	0.1064	0.5950	1.3024	4.0018	12.1766	26.4626	92.4859
0.2	0.0775	0.4571	0.9032	2.8692	6.4546	7.2892	38.3812
0.4	0.0455	0.1908	0.5445	1.1207	1.3735	2.3625	2.8235
0.6		0.0592	0.1030	0.1688	0.2844	0.4985	1.2547
0.8			0.0311	0.0607	0.2079	0.4219	0.9143

Table 7: XSL-variant results with 2 variables

	5	10	15	20	30	40	50
0.0	2.3599	11.0589	37.4750	64.9952	178.7157	482.1859	1013.2891
0.2	1.8573	7.1120	26.9514	41.4408	126.3859	277.0579	594.3859
0.4	1.2703	3.5620	14.6126	26.8235	51.2281	148.7342	189.6328
0.6	0.6849	1.4734	2.4891	2.9405	13.9453	24.1124	28.3343
0.8		0.0714	0.0766	0.1251	0.5593	0.6797	1.0719

Table 8: XSL-variant results with 3 variables

	5	10	15	20	30	40	50
0.0		986.2442	523.3434	880.7343			
0.2		220.6304	400.3073	745.7602	1822.0208		
0.4		140.5107	256.3078	505.5987	1376.1460		
0.6		91.8565	163.6042	309.1509	710.8437		
0.8			0.1770	0.7814	0.9741		

Table 9: XSL-variant results with 4 variables

14.1.3 Comparison of XSL and XSL-variant

This section includes a comparison of the two algorithms in terms of speed. The values in the tables indicate the speed ratio of the two algorithms; in other words, they show how much faster XSL is than its variant. The chart format is the same as the previous sections.

	5	10	15	20	30	40	50
0.0	1.957	4.940	7.902	12.009	23.062	18.991	19.337
0.2	1.685	5.115	6.942	13.339	22.262	20.487	19.722
0.4		3.480	5.771	8.433	11.428	10.960	12.835
0.6		2.530	3.842	5.883	10.810	11.821	19.437

Table 10: Comparison for 1 variable

	5	**10**	**15**	**20**	**30**	**40**	**50**
0.0	2.512	3.388	2.785	3.866	3.443	3.923	5.611
0.2	2.307	3.373	2.877	4.727	2.293	1.937	5.837
0.4	2.157	3.500	4.982	7.282	4.062	2.939	2.850
0.6		4.400	6.104	5.841	6.452	6.649	9.674

Table 11: Comparison for 2 variables

	5	**10**	**15**	**20**	**30**	**40**	**50**
0.0	2.189	3.247	4.053	4.053	2.191	3.041	3.274
0.2	2.014	2.408	3.795	3.214	1.792	3.159	3.064
0.4	1.534	1.966	2.711	3.170	1.585	3.633	2.051
0.6	0.953	1.733	2.029	1.325	2.857	2.853	1.524

Table 12: Comparison for 3 variables

14.2 Interpretation of Results

It is possible to interpret the results in numerous ways. It is possible to keep any two of the three parameters and analyze the effects of the third parameter on the behavior of the algorithm, or to fix one parameter and analyze the combined effect of the other two; it is also noteworthy to compare XSL with the XSL variant.

14.2.1 Effect of Parameters on XSL's Performance

- **Number of Equations and Number of Variables**
  The system size affects the speed of the algorithm dramatically for obvious reasons. The effect is amplified by the extension which depends directly on the number of equations and variables. Looking at Figures 3-6, one can observe that the effect that the equation system size, both length- and width-wise, is independent from the sparsity ratio.

- **Sparsity Ratio and Number of Variables**
  Figures 7-10 demonstrates the huge impact of the rise of the number of variables affects the performance of the algorithm in different sparsity ratio values. As mentioned before, the number of variables has the greatest effect on the speed of the algorithm.

- **Sparsity Ratio and Number of Equations**
  Figures 11-13 demonstrate how different sparsity ratios cause the equation system to behave for different equation sizes. The first observation is that the speed of the algorithm is affected much more by the size for

	5	**10**	**15**	**20**	**30**	**40**	**50**
0.0		12.536	2.834	1.916			
0.2		3.193	2.240	1.802	1.637		
0.4		2.622	2.218	1.322	1.270		
0.6		3.830	3.020	2.558	1.005		

Table 13: Comparison for 4 variables

non-sparse systems. For high sparsity ratios like 80%, the algorithm is much faster, does not slow down as much for larger systems, but also becomes wildly unpredictable. Especially for larger systems with a greater number of variables, the algorithm can behave erratically, with the maximum value for one execution time measuring up to 1000 times more time as the minimum value.

In addition, Figures 14-17 display the performance of the XSL variant.

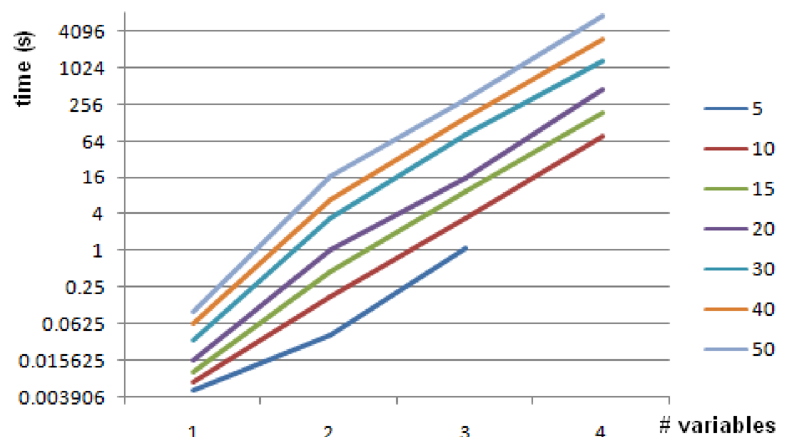

Figure 3: Sparsity = 0

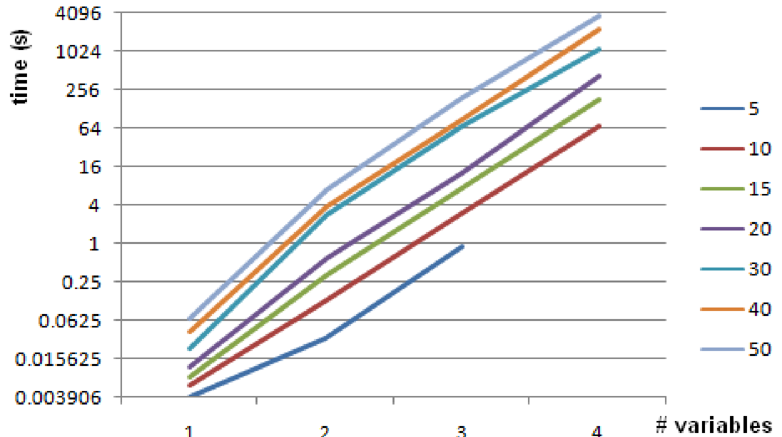

Figure 4: Sparsity = 0.2

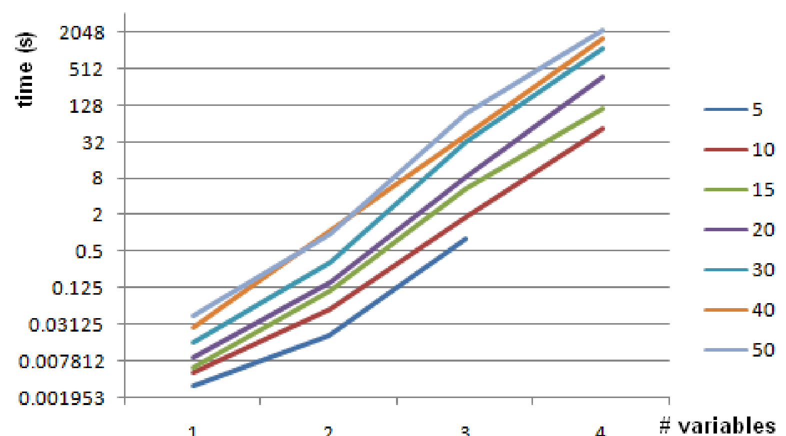

Figure 5: Sparsity = 0.4

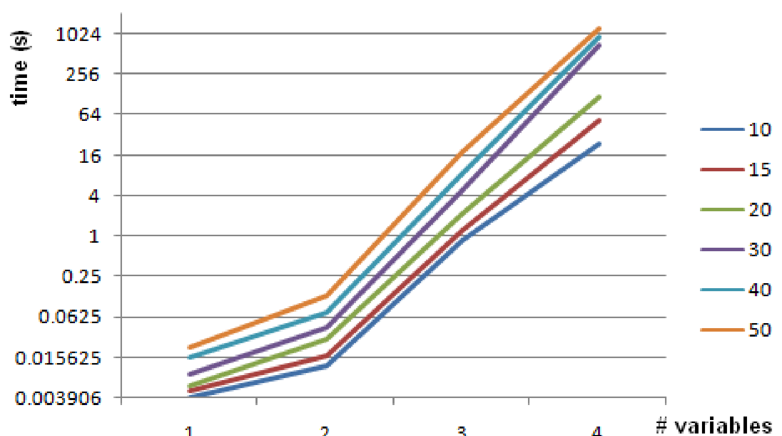

Figure 6: Sparsity = 0.6

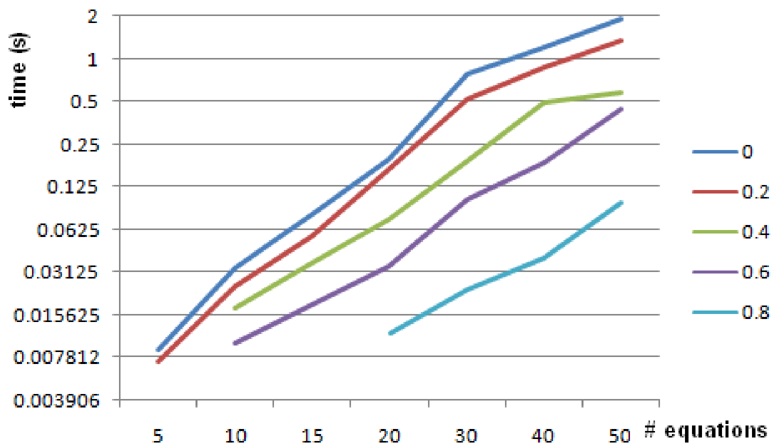

Figure 7: 5 equations

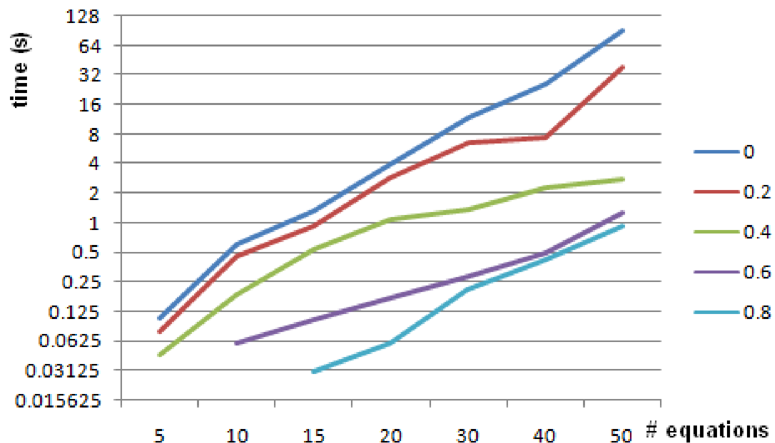

Figure 8: 10 equations

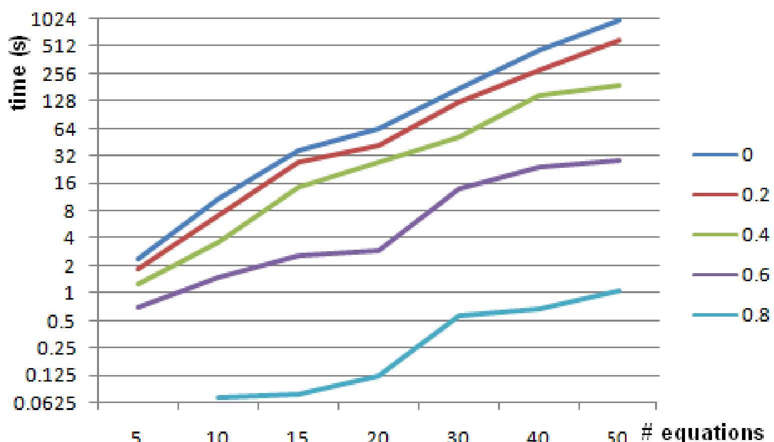

Figure 9: 15 equations

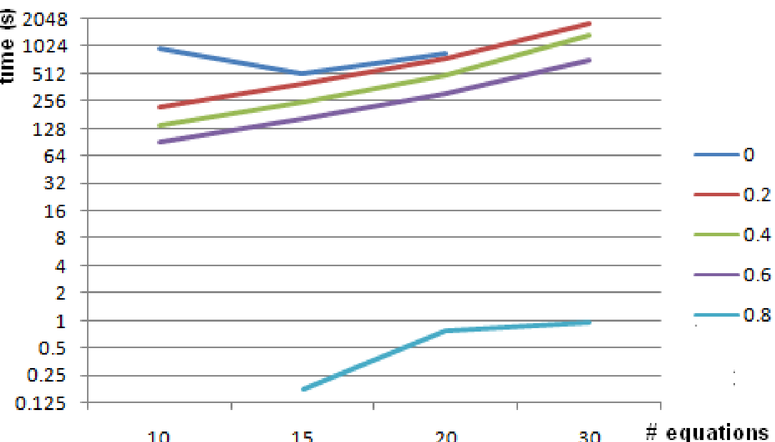

Figure 10: 20 equations

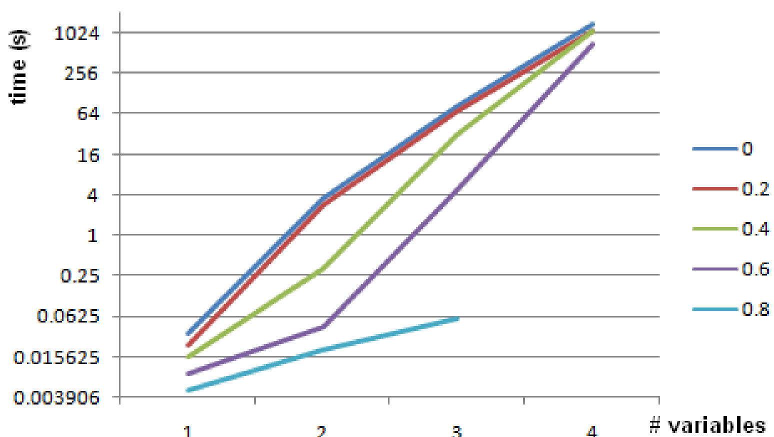

Figure 11: 30 equations

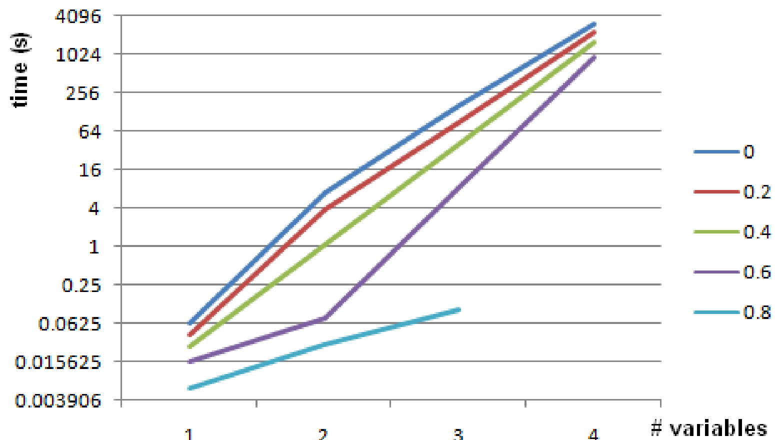

Figure 12: 40 equations

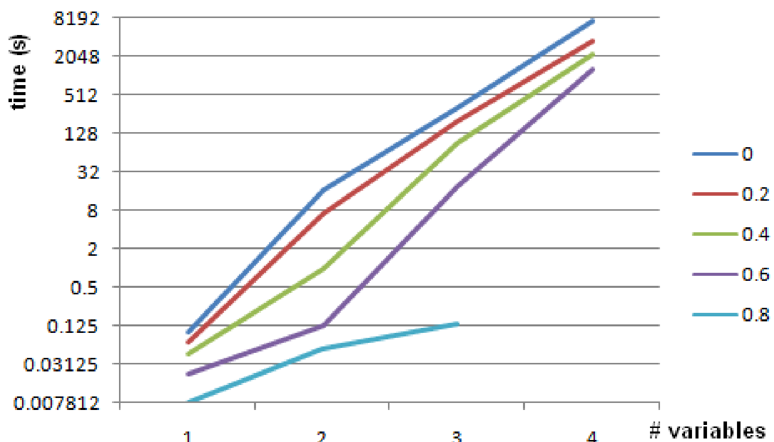

Figure 12: 50 equations

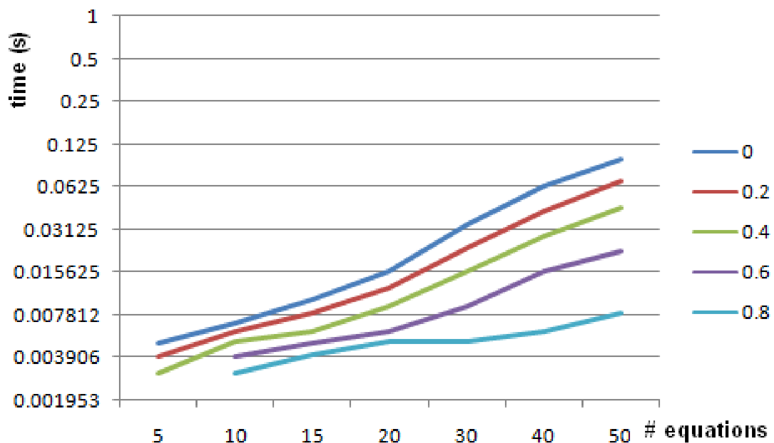

Figure 14: 1 variable

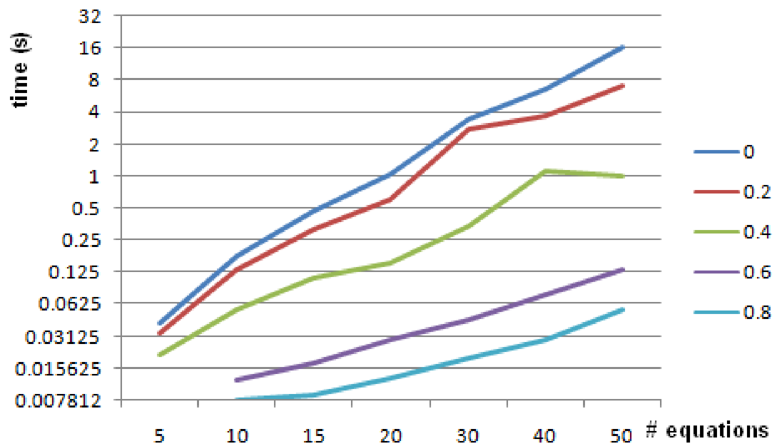

Figure 15: 2 variables

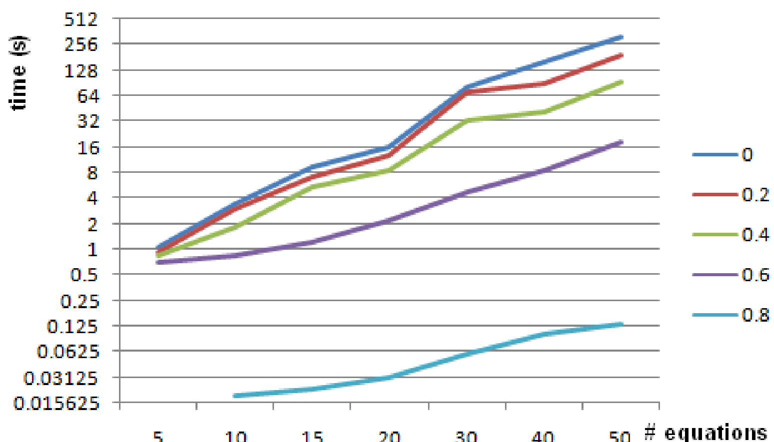

Figure 16: 3 variables

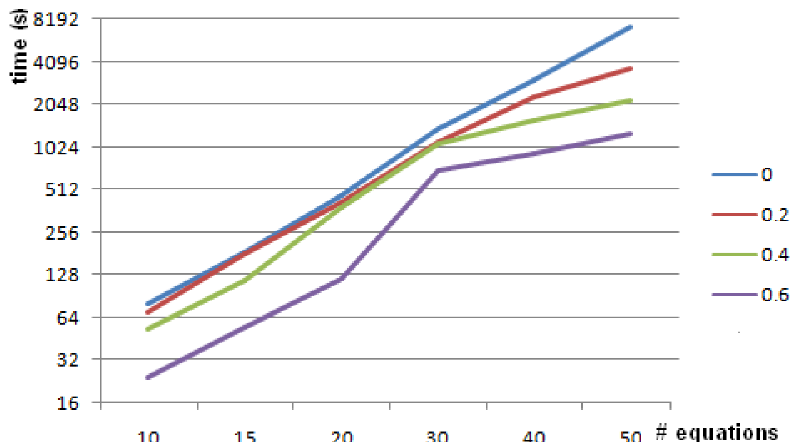

Figure 17: 4 variables

14.2.2 XSL versus Variant

When comparing XSL with the variant, one significant trend is to be observed in the results. Although for small systems the variant runs much slower than XSL, it is to be noticed that the gap shrinks dramatically as the equation system gets larger. For instance, comparing non-sparse equation systems with 20 equations, a 1-variable system takes 12 times longer to solve with the variant than XSL; on the other hand, a 4-variable system only takes 1.9 times longer.

This convergence is to be expected due to the nature of the algorithms and the fact that the number of linear terms increases much more slowly than the number of quadratic terms as the number of variables in the system increases. Since the difference between the two algorithms is that the variant additionally uses linear terms in extension, the ratio of the extra extension to the core extension decreases as the number of variables in the equation system increases. This is especially true for larger systems with more equations; for very small systems, this can work inversely, due to the small number of equations with which to extend the system.

Figure 18 indicates the gap between the two algorithms as the number of variables numbers begins to change. The figures only demonstrate systems of zero sparsity, since as soon as a random non-zero sparsity comes into consideration, it is very difficult to know that the ratio of random zero coefficients are evenly distributed among linear and quadratic variables, which may skew the figures towards one algorithm or the other.

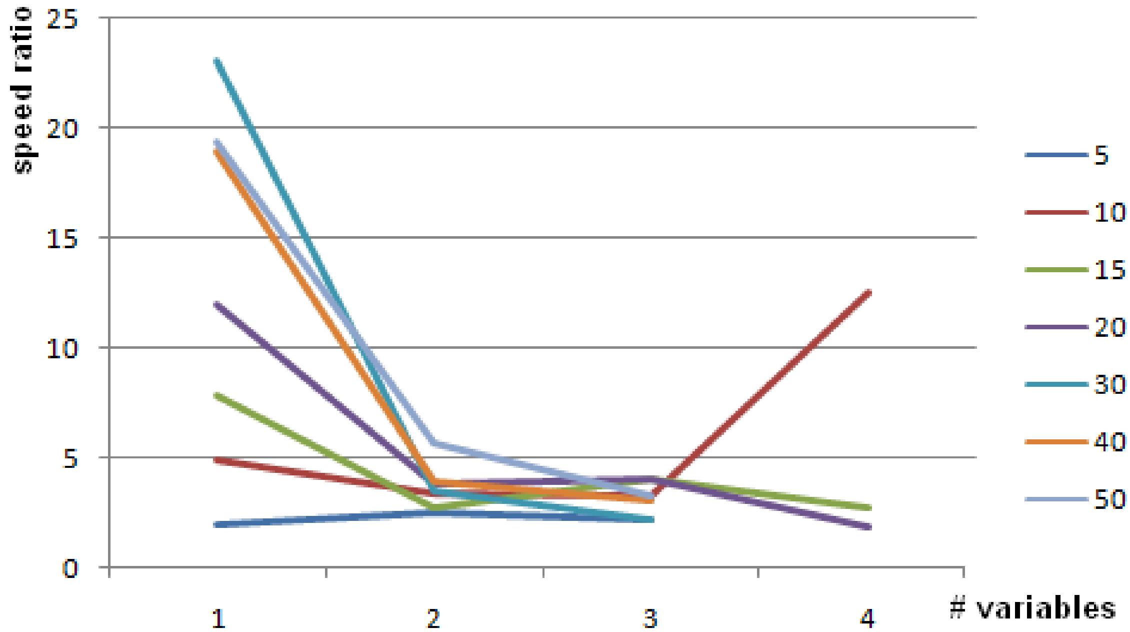

Figure 18: Speed difference between XSL and XSL-variant

15 Room For Improvement

Due to time, space, memory and scope constraints, the objective or size of this project was deliberately limited. However, the design was made as to allow easy expansions in several directions.

To beat the space and memory requirements, one may use more sophisticated machines to run the experiments. This would allow results to be gathered for higher numbers of variables and the experiment graphs to be extended into more explanatory curves. Although we believe the experiments that we have conducted to provide satisfactory results about the general behavioral patterns of the algorithm, experimenting with larger equation systems would take us a few steps closer to understanding how XSL would handle Rijndael in a real-life situation and this information would no doubt be invaluable.

Our software was also designed with the flexibility to accept any equation system, with the condition that the system satisfies strict formatting guidelines. The objective in doing so was to allow any interested scholars and researchers who develop code to produce the MQ equation systems from

Rijndael to be able to have a system implemented and ready to process their data, to perform XSL and print out the results.

Additionally, our software uses Java collection classes to store and manipulate equation systems. These collections are not optimized for our purposes and severely impact the speed of our calculations. While this was a foreseeable issue, we believed that it would affect performance of both equation types on an equal magnitude and thus allow fair comparison. However, if this project is to be taken further and larger experiments are to be conducted, overall performance must be improved, starting with using self-developed and optimized storage classes. Also, it should be noted that the algorithm is in fact parallel-computable; for further improvement of performance on multi-processor or multi-machine environments, the code should be altered to allow parallel execution.

The most significant improvement that can be made, which we would like to list as a potential subject of future work, is to generate a generic formulation of field transformation from more variables of more restricted size to larger ones, making it possible to express the system in even more compact forms. We expect this to speed up the XSL algorithm, as demonstrated by the dramatic performance difference for systems in bytes and bits. This would, however, require us to modify some parts of our software design, especially the class `Type` which can only accommodate byte and bit variables in its current state.

16 Conclusion

The experiments we have conducted have provided rather definitive results as to our primary question: does XSL prefer MQ systems with byte-variables where the representation is more compact but the algebra is more complex, or one with bit-variables where the system is larger but algebra is so much simpler? The difference in XSL's performance in both cases is so dramatically different that it does not leave any room for doubt. While the algorithm can evaluate non-sparse byte-systems of up to 7 bytes and 50 equations with 1 GB memory, evaluating even the bit-equivalent of a 1-variable 1-equation system proves a task too demanding for the same machine.

This difference in perfomance has led to a new idea: can we do the inverse of the byte-to-bit field transformation and, starting from bytes, derive formulae for representing the system in a more compact manner than byte-

variables, such as a block of 8-bytes, or indeed generalize the conversion for blocks of 8^n-byte blocks, and increase the performance of the algorithm? This would pose several major challenges: the formulae would need to be derived to transform a byte-system to an 8-byte-block-system; indexing of variables would need to be taken care of; implementation-specific details of the code would need to be modified and expanded; and finally, if one decides to test the outcome on a real cipher, then she would need to find a way to represent the cipher as an MQ system of 8-byte blocks, which has never been done before. Another matter of concern is that when all calculations are done and the system has been modified, there is no guarantee that the new representation will actually be preferred by XSL. Although one can intuitively suggest that that will be the case, intuition may not always be a reliable guide; if that were the case, our experiment results would have yielded equal results for byte- and bit-systems.

We conclude the paper with reiterating that for the very small equation systems that we have experimented with, XSL prefers byte-systems to bit-systems. As for the suggestion of using a more compact representation than byte-variables, we suffice with merely stating this idea as a prospect for future work and encouraging cryptology enthusiasts to take up our work and move it further in this direction.

References

[1] Eli Biham and Adi Shamir. Differential cryptanalysis of des-like cryptosystems. In *CRYPTO '90: Proceedings of the 10th Annual International Cryptology Conference on Advances in Cryptology*, pages 2–21, London, UK, 1991. Springer-Verlag.

[2] Alex Biryukov and Christoph De Canniere. Block ciphers and systems of quadratic equations, 2003. in Fast Software Encryption, FSE 2003.

[3] Carlos Cid and Gaetan Leurent. An analysis of the xsl algorithm. *ASIACRYPT*, pages 333–352, 2005.

[4] Carlos Cid, Sean Murphy, and Matthew Robshaw. Computational and algebraic aspects of the advanced encryption standard. *Proceedings of the Seventh International Workshop on Computer Algebra in Scientific Computing*, pages 93–103, 2004.

[5] Nicolas Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. *EUROCRYPT*, pages 392–407, 2000.

[6] Nicolas Courtois and Josef Pieprzyk. Cryptanalysis of block ciphers with overdefined systems of equations. In *ASIACRYPT '02: Proceedings of the 8th International Conference on the Theory and Application of Cryptology and Information Security*, pages 267–287, London, UK, 2002. Springer-Verlag.

[7] Joan Daemen and Vincent Rijmen. The block cipher rijndael, 1998. `http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael.pdf`.

[8] Claus Diem. The xl-algorithm and a conjecture from commutative algebra. *Lecture Notes in Computer Science : Advances in Cryptology - ASIACRYPT 2004*, pages 323–337, 2004. `http://www.math.uni-leipzig.de/~diem/preprints/xl.ps`.

[9] Horst Feistel. Block cipher cryptographic system, 1974. U.S. Patent No. 3,798,359.

[10] Niels Ferguson, Richard Schroeppel, and Doug Whiting. A simple algebraic representation of rijndael. In *SAC '01: Revised Papers from the 8th Annual International Workshop on Selected Areas in Cryptography*, pages 103–111, London, UK, 2001. Springer-Verlag.

[11] Ralph Howard. Data encryption standard. *Inf. Age*, 9(4):204–210, 1987.

[12] Aviad Kipnis and Adi Shamir. Cryptanalysis of the hfe public key cryptosystem by relinearization. In *CRYPTO '99: Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, pages 19–30, London, UK, 1999. Springer-Verlag.

[13] Elizabeth Kleiman. The xl and xsl attacks on baby rijndael. Master's thesis, Iowa State University, 2005. http://orion.math.iastate.edu/dept/thesisarchive/MS/EKleimanMSSS05.pdf.

[14] Mitsuru Matsui. The first experimental cryptanalysis of the data encryption standard. In *CRYPTO '94: Proceedings of the 14th Annual International Cryptology Conference on Advances in Cryptology*, pages 1–11, London, UK, 1994. Springer-Verlag.

[15] Tsutomu Matsumoto and Hideki Imai. Public quadratic polynomial-tuples for efficient signature-verification and message-encryption. In *Lecture Notes in Computer Science on Advances in Cryptology-EUROCRYPT'88*, pages 419–453, New York, NY, USA, 1988. Springer-Verlag New York, Inc.

[16] Tzuong-Tsieng Moh. On the method of "xl" and its inefficiency to ttm. http://www.usdsi.com/XL.ps.

[17] Tzuong-Tsieng Moh. A public key system with signature and master key functions, 1999.

[18] Sean Murphy and Matthew Robshaw. Essential algebraic structure within the aes. In *CRYPTO '02: Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology*, pages 1–16, London, UK, 2002. Springer-Verlag.

[19] Jacques Patarin. Hidden field equations (hfe) and isomorphisms of polynomials (ip): Two new families of asymmetric algorithms. *Eurocrypt 96*, pages 33–48, 1996.

[20] Jacques Patarin. Cryptanalysis of the matsumoto and imai public key scheme of eurocrypt98. *Des. Codes Cryptography*, 20(2):175–209, 2000.

[21] Bruce Schneier. AES news, Sep. 2002. http://www.schneier.com/crypto-gram-0209.html#1.

[22] Ralph-Philip Weinmann. Evaluating algebraic attacks on aes. Master's thesis, Technische Universitat Darmstadt, 2003. `http://www.informatik.tu-darmstadt.de/TI/Mitarbeiter/weinmann/Diplomarbeit.pdf`.

[23] Christopher Wolf. Multivariate quadratic polynomials in public key cryptography. Master's thesis, Katholieke Universiteit Leuven, 2005. `https://repository.libis.kuleuven.be/dspace/bitstream/1979/148/2/thesis.pdf`.

A Running the Code

The main method is in the `MainTester` class. It expects 5 command line arguments, and can be run with the command:

java MainTester <num equations> <num variables> <sparsity>
<repeats> <methods to use>

- *num equations* defines the number of equations that the random equation generator to generate.
- *num variables* is the number of variables that the random equation system will contain.
- *degree* , the degree of the equation system, was left to the user to decide; however, since our work is a specific implementation aimed at solving quadratic equations, there are several sections in the program that expect the system to be of degree 2, such as the field transformation. If the user wants only to solve the system in bytes and not do a field transformation, then other values for degree may be used. If a higher degree is used and transformation is requested, then only constant, linear and quadratic terms will be converted and the rest will be discarded.
- *sparsity* is the probability that a random coefficient in the system will be zero.
- *repeats* defines the number of times the whole process is going to be repeated.
- *methods to use* are the techniques to use by which the equation system will be solved:
  - *B* for solving the equation system in bytes,
  - *b* for converting the system to bits and solving it,
  - *X* for doing a brute force search on the byte system. Brute force search for bits was not implemented because it is essentially the same process as the brute force on bytes.

The original equation system is in bytes by default. This is necessary for operations such as field transformation.

To run the program for an equation system with 10 equations, 2 variables, 50% sparsity ratio for 20 times, and to use XSL on bytes as well as doing a brute force search, the command to use would be

```
java MainTester 10 2 .5 20 BX
```

The output is the total time that each method has taken:

XSL for 256: (10,2,.5) * 20 = 10.655 seconds.
BFS for 256: (10,2,.5) * 20 = 34.876 seconds.

B XSL on a Byte-System

Let's assume that we start with the following quadratic equation system, with 3 equations, 2 variables and .4 sparsity. All coefficients are decimal representations of bytes.

$$\begin{aligned} 125 &= 96x + 206xy + 141y^2 \\ 96 &= 189y + 152y^2 \\ 68 &= 202x + 159y + 185xy \end{aligned}$$

When multiplications are made, the system expands into

$$\begin{aligned} 125 &= 96x + 206xy + 141y^2 \\ 96 &= 189y + 152y^2 \\ 68 &= 202x + 159y + 185xy \\ 0 &= 176y^2 + 145xy^2 + 232xy^3 + 76y^4 \\ 0 &= 27y + 222xy + 155xy^2 + 211y^3 \\ 0 &= 244xy + 69x^2y + 142x^2y^2 + 209xy^3 \\ 0 &= 169x + 120x^2 + 112x^2y + 101xy^2 \\ 0 &= 216y + 170xy + 180xy^2 + 194y^3 \\ 0 &= 227xy + 155xy^2 + 232xy^3 \\ 0 &= 48y^2 + 211y^3 + 76y^4 \\ 0 &= 199x + 222xy + 145xy^2 \\ 0 &= 69xy + 126xy^2 + 223xy^3 \\ 0 &= 120x + 89xy + 190xy^2 \\ 0 &= 170y + 151y^2 + 36y^3 \\ 0 &= 94xy + 112x^2y + 180xy^2 + 142x^2y^2 \\ 0 &= 34y^2 + 101xy^2 + 194y^3 + 209xy^3 \\ 0 &= 232x + 120x^2 + 170xy + 69x^2y \\ 0 &= 65y^2 + 190xy^2 + 36y^3 + 223xy^3 \\ 0 &= 86y + 89xy + 151y^2 + 126xy^2 \end{aligned}$$

Every monomial in the extended system is then treated as a new, linear variable:

$$\begin{aligned}
125 &= 96a + 206f + 141j \\
96 &= 189e + 152j \\
68 &= 202a + 159e + 185f \\
0 &= 176j + 145k + 232p + 76t \\
0 &= 27e + 222f + 155k + 211o \\
0 &= 244f + 69g + 142l + 209p \\
0 &= 169a + 120b + 112g + 101k \\
0 &= 216e + 170f + 180k + 194o \\
0 &= 227f + 155k + 232p \\
0 &= 48j + 211o + 76t \\
0 &= 199a + 222f + 145k \\
0 &= 69f + 126k + 223p \\
0 &= 120a + 89f + 190k \\
0 &= 170e + 151j + 36o \\
0 &= 94f + 112g + 180k + 142l \\
0 &= 34j + 101k + 194o + 209p \\
0 &= 232a + 120b + 170f + 69g \\
0 &= 65j + 190k + 36o + 223p \\
0 &= 86e + 89f + 151j + 126k
\end{aligned}$$

In this relinearized system, we perform Gauss-Jordan elimination.

$$\begin{aligned}
175 &= 96a + 82t \\
49 &= 189e + 71t \\
186 &= 195f + 115t \\
158 &= 176j + 26t \\
144 &= 96k + 18t \\
123 &= 69g + 225t \\
181 &= 120b + 74t \\
243 &= 245o + 245t \\
174 &= 158p + 236t \\
241 &= 167t \\
254 &= 148l + 106t \\
111 &= 219t \\
111 &= 219t \\
171 &= 202t \\
90 &= 109t
\end{aligned}$$

We need to find a univariate equation in this system, evaluate it, then substitute the value back into the original equation. However, what seems like a univariate equation in this system may not be univariate on the system before relinearization; we must identify an equation that is also univariate on the first system. Starting from the top, we check each equation for the number of variables it contains. The equation

$$241 = 167t$$

translates to

$$241 = 167y^4$$

in the original equation system. Remember that a univariate-looking equation in the relinearized system could well have contained a multivariate term in the original system, hence the need to convert it back to the original variables at this point.

Solving this equation, we calculate the value of y to be 127. Now, we substitute this value into the original equation system:

$$\begin{aligned} 180 &= 161x \\ 0 &= 0 \\ 185 &= 87x \end{aligned}$$

Normally, we would remove the empty equation from the system and repeat the whole multiply-eliminate-solve-substitute process all over again; however, since we are left with a univariate equation system, solving it is trivial. We know that any equation in a univariate equation system (except for the empty equation) will be univariate; we pick one, solve it, and end up with the result vector:

$$\begin{aligned} x &= 23 \\ y &= 127 \end{aligned}$$

C Transformation: Byte to Bit

Using the transformation formulae derived in Section 6, the following equation system of bytes

$$\begin{aligned} 232 &= 183x + 15x^2 + 17y + 7x^y + 99y^2 \\ 211 &= 31x + 146x^2 + 42y + 106xy + 75y^2 \end{aligned}$$

can be transformed into its equivalent system in bits as follows:

$$1 = x_2+x_3+x_6+x_1y_0+x_2y_0+x_6y_0+y_0^2+y_1+x_1y_1+x_5y_1+x_7y_1+y_0y_1+y_2+x_4y_2+x_6y_2+x_3y_3+x_5y_3+x_2y_4+x_4y_4+x_7y_4+x_1y_5+x_3y_5+x_6y_5+x_7y_5+y_4y_5+y_6+x_2y_6+x_5y_6+x_6y_6+y_3y_6+x_1y_7+x_4y_7+x_5y_7+x_7y_7+y_2y_7+y_6y_7+y_7^2 0 = x_2+x_4+x_7+y_0+x_1y_0+x_3y_0+x_6y_0+x_7y_0+y_1+x_2y_1+x_5y_1+x_6y_1+x_7y_1+y_1^2+x_1y_2+x_4y_2+x_5y_2+x_6y_2+x_7y_2+y_0y_2+y_3+x_3y_3+x_4y_3+x_5y_3+x_6y_3+x_2y_4+x_3y_4+x_4y_4+x_5y_4+x_7y_4+x_1y_5+x_2y_5+x_3y_5+x_4y_5+x_6y_5+y_4y_5+y_5^2+y_6+x_1y_6+x_2y_6+x_3y_6+x_5y_6+x_7y_6+y_3y_6+y_4y_6+y_7+x_1y_7+x_2y_7+x_4y_7+x_6y_7+x_7y_7+y_2y_7+y_3y_7+y_6y_7$$

$$1 = x_3+x_5+x_7+y_0+x_1y_0+x_2y_0+x_4y_0+x_7y_0+y_0^2+y_1+x_1y_1+x_3y_1+x_6y_1+x_7y_1+y_2+x_2y_2+x_5y_2+x_6y_2+x_7y_2+y_1y_2+x_1y_3+x_4y_3+x_5y_3+x_6y_3+x_7y_3+y_0y_3+y_4+x_3y_4+x_4y_4+x_5y_4+x_6y_4+x_2y_5+x_3y_5+x_4y_5+x_5y_5+x_7y_5+y_5^2+x_1y_6+x_2y_6+x_3y_6+x_4y_6+x_6y_6+y_4y_6+y_5y_6+y_7+x_1y_7+x_2y_7+x_3y_7+x_5y_7+x_7y_7+y_3y_7+y_4y_7+y_7^2$$

$$1 = x_2+x_3+x_4+y_0+x_3y_0+x_5y_0+x_6y_0+y_0^2+x_2y_1+x_4y_1+x_5y_1+x_1y_2+x_3y_2+x_4y_2+x_7y_2+y_2^2+y_3+x_2y_3+x_3y_3+x_6y_3+x_7y_3+y_1y_3+x_1y_4+x_2y_4+x_5y_4+x_6y_4+y_0y_4+y_5+x_1y_5+x_4y_5+x_5y_5+x_7y_5+y_4y_5+x_3y_6+x_4y_6+x_6y_6+x_7y_6+y_0y_6+y_3y_6+y_5y_6+y_6^2+x_2y_7+x_3y_7+x_5y_7+x_6y_7+x_7y_7+y_2y_7+y_4y_7+y_5y_7+y_6y_7+y_7^2$$

$$1 = x_2+x_4+x_5+x_6+x_2y_0+x_4y_0+x_7y_0+x_1y_1+x_3y_1+x_6y_1+x_7y_1+y_2+x_2y_2+x_5y_2+x_6y_2+x_1y_3+x_4y_3+x_5y_3+x_7y_3+y_2y_3+y_4+x_3y_4+x_4y_4+x_6y_4+y_1y_4+x_2y_5+x_3y_5+x_5y_5+x_7y_5+y_0y_5+y_4y_5+y_5^2+x_1y_6+x_2y_6+x_4y_6+x_7y_6+y_3y_6+y_4y_6+y_7+x_1y_7+x_3y_7+x_5y_7+x_6y_7+y_2y_7+y_3y_7+y_5y_7$$

$$1 = x_0+x_3+x_5+x_6+x_7+x_3y_0+x_5y_0+x_2y_1+x_4y_1+x_7y_1+x_1y_2+x_3y_2+x_6y_2+x_7y_2+y_3+x_2y_3+x_5y_3+x_6y_3+y_3^2+x_1y_4+x_4y_4+x_5y_4+x_7y_4+y_2y_4+y_5+x_3y_5+x_4y_5+x_6y_5+y_1y_5+y_5^2+x_2y_6+x_3y_6+x_5y_6+x_7y_6+y_0y_6+y_4y_6+y_5y_6+x_1y_7+x_2y_7+x_4y_7+x_6y_7+x_7y_7+y_3y_7+y_4y_7+y_6y_7$$

$$1 = x_0+x_1+x_4+x_6+x_7+y_0+x_4y_0+x_6y_0+x_3y_1+x_5y_1+x_2y_2+x_4y_2+x_7y_2+x_1y_3+x_3y_3+x_6y_3+x_7y_3+y_4+x_2y_4+x_5y_4+x_6y_4+y_3y_4+x_1y_5+x_4y_5+x_5y_5+x_7y_5+y_2y_5+y_6+x_3y_6+x_4y_6+x_6y_6+y_1y_6+y_5y_6+y_6^2+x_2y_7+x_3y_7+x_5y_7+x_7y_7+y_0y_7+y_4y_7+y_5y_7+y_7^2$$

$$0 = x_1+x_2+x_5+x_7+y_0+x_1y_0+x_5y_0+x_7y_0+y_0^2+y_1+x_4y_1+x_6y_1+x_3y_2+x_5y_2+x_2y_3+x_4y_3+x_7y_3+x_1y_4+x_3y_4+x_6y_4+x_7y_4+y_4^2+y_5+x_2y_5+x_5y_5+x_6y_5+y_3y_5+x_1y_6+x_4y_6+x_5y_6+x_7y_6+y_2y_6+y_6^2+y_7+x_3y_7+x_4y_7+x_6y_7+x_7y_7+y_1y_7+y_5y_7+y_6y_7+y_7^2$$

$0 = y_0 + y_1 + y_0y_1 + y_2^2 + y_3 + y_1y_3 + y_3^2 + y_4 + y_0y_4 + y_2y_4 + y_1y_5 + y_5^2 + y_0y_6 + y_4y_6 + y_6^2 + y_7 + y_3y_7 + y_5y_7 + y_6y_7$

$0 = y_0y_1 + y_1^2 + y_2 + y_0y_2 + y_2^2 + y_3 + y_1y_3 + y_2y_3 + y_3^2 + y_0y_4 + y_1y_4 + y_2y_4 + y_3y_4 + y_5 + y_0y_5 + y_1y_5 + y_2y_5 + y_5^2 + y_6 + y_0y_6 + y_1y_6 + y_4y_6 + y_5y_6 + y_6^2 + y_7 + y_0y_7 + y_3y_7 + y_4y_7 + y_5y_7 + y_7^2$

$0 = y_0 + y_1^2 + y_0y_2 + y_1y_2 + y_3 + y_0y_3 + y_2y_3 + y_3^2 + y_4 + y_1y_4 + y_2y_4 + y_3y_4 + y_4^2 + y_0y_5 + y_1y_5 + y_2y_5 + y_3y_5 + y_6 + y_0y_6 + y_1y_6 + y_2y_6 + y_5y_6 + y_6^2 + y_0y_7 + y_1y_7 + y_4y_7 + y_5y_7 + y_6y_7$

$0 = y_0 + y_0^2 + y_0y_1 + y_1y_2 + y_3 + y_0y_3 + y_3y_4 + y_4^2 + y_5 + y_2y_5 + y_3y_5 + y_4y_5 + y_5^2 + y_0y_6 + y_1y_6 + y_2y_6 + y_3y_6 + y_4y_6 + y_0y_7 + y_1y_7 + y_2y_7 + y_3y_7 + y_7^2$

$0 = y_0 + y_0^2 + y_1^2 + y_0y_2 + y_3 + y_3^2 + y_2y_4 + y_4^2 + y_1y_5 + y_3y_5 + y_4y_5 + y_6 + y_0y_6 + y_2y_6 + y_3y_6 + y_5y_6 + y_6^2 + y_7 + y_1y_7 + y_2y_7 + y_4y_7 + y_5y_7 + y_6y_7$

$1 = y_0 + y_1 + y_0y_1 + y_1y_2 + y_0y_3 + y_4 + y_3y_4 + y_2y_5 + y_4y_5 + y_5^2 + y_1y_6 + y_3y_6 + y_4y_6 + y_6^2 + y_7 + y_0y_7 + y_2y_7 + y_3y_7 + y_5y_7 + y_6y_7 + y_7^2$

$1 = y_1 + y_1^2 + y_2 + y_0y_2 + y_2^2 + y_1y_3 + y_0y_4 + y_4^2 + y_5 + y_3y_5 + y_5^2 + y_2y_6 + y_4y_6 + y_5y_6 + y_1y_7 + y_3y_7 + y_4y_7 + y_6y_7 + y_7^2$

$0 = y_0 + y_0^2 + y_2 + y_1y_2 + y_3 + y_0y_3 + y_2y_3 + y_1y_4 + y_0y_5 + y_4y_5 + y_6 + y_3y_6 + y_5y_6 + y_6^2 + y_2y_7 + y_4y_7 + y_5y_7 + y_7^2$

$0 = x_1 + x_2 + y_0 + y_0 + x_1y_0 + x_3y_0 + x_4y_0 + x_6y_0 + x_7y_0 + y_1 + y_1 + x_2y_1 + x_3y_1 + x_5y_1 + x_6y_1 + x_1y_2 + x_2y_2 + x_4y_2 + x_5y_2 + y_3 + x_1y_3 + x_3y_3 + x_4y_3 + y_4 + y_4 + x_2y_4 + x_3y_4 + x_1y_5 + x_2y_5 + y_6 + x_1y_6 + x_7y_6 + y_7 + y_7 + x_6y_7$

$0 = x_0 + x_1 + x_3 + y_0 + x_2y_0 + x_3y_0 + x_5y_0 + x_6y_0 + x_1y_1 + x_2y_1 + x_4y_1 + x_5y_1 + x_7y_1 + y_2 + y_2 + x_1y_2 + x_3y_2 + x_4y_2 + x_6y_2 + y_3 + x_2y_3 + x_3y_3 + x_5y_3 + y_4 + x_1y_4 + x_2y_4 + x_4y_4 + y_5 + y_5 + x_1y_5 + x_3y_5 + y_6 + y_6 + x_2y_6 + x_7y_6 + y_7 + x_1y_7 + x_6y_7 + x_7y_7$

$1 = x_1 + x_2 + x_4 + y_0 + x_3y_0 + x_4y_0 + x_6y_0 + x_7y_0 + y_1 + x_2y_1 + x_3y_1 + x_5y_1 + x_6y_1 + x_1y_2 + x_2y_2 + x_4y_2 + x_5y_2 + x_7y_2 + y_3 + y_3 + x_1y_3 + x_3y_3 + x_4y_3 + x_6y_3 + y_4 + x_2y_4 + x_3y_4 + x_5y_4 + y_5 + x_1y_5 + x_2y_5 + x_4y_5 + y_6 + y_6 + x_1y_6 + x_3y_6 + y_7 + x_2y_7 + x_7y_7$

$0 = x_0 + x_1 + x_3 + x_5 + y_0 + y_0 + x_1y_0 + x_3y_0 + x_5y_0 + x_6y_0 + y_1 + x_2y_1 + x_4y_1 + x_5y_1 + x_7y_1 + y_2 + x_1y_2 + x_3y_2 + x_4y_2 + x_6y_2 + y_3 + x_2y_3 + x_3y_3 + x_5y_3 + x_7y_3 + x_1y_4 + x_2y_4 +$

$x_4y_4+x_6y_4+y_5+x_1y_5+x_3y_5+x_5y_5+y_6+x_2y_6+x_4y_6+x_7y_6+x_1y_7+x_3y_7+x_6y_7$

$0 = x_4+x_6+y_0+y_0+x_2y_0+x_3y_0+x_1y_1+x_2y_1+y_2+x_1y_2+x_7y_2+y_3+y_3+x_6y_3+y_4+x_5y_4+x_7y_4+x_4y_5+x_6y_5+x_3y_6+x_5y_6+x_6y_6+x_7y_6+x_2y_7+x_4y_7+x_6y_7+x_7y_7$

$0 = x_5+x_7+y_0+x_1y_0+x_3y_0+x_4y_0+y_1+y_1+x_2y_1+x_3y_1+x_1y_2+x_2y_2+y_3+x_1y_3+x_7y_3+y_4+y_4+x_6y_4+y_5+x_5y_5+x_7y_5+x_4y_6+x_6y_6+x_3y_7+x_5y_7+x_7y_7$

$1 = x_0 + x_6 + x_1y_0 + x_2y_0 + x_4y_0 + x_5y_0 + y_1 + x_1y_1 + x_3y_1 + x_4y_1 + y_2 + y_2 + x_2y_2 + x_3y_2 + x_7y_2 + x_1y_3 + x_2y_3 + y_4 + x_1y_4 + x_7y_4 + y_5 + y_5 + x_6y_5 + y_6 + x_5y_6 + x_7y_6 + x_4y_7 + x_6y_7$

$0 = x_0 + x_1 + x_7 + y_0 + y_0 + x_2y_0 + x_3y_0 + x_5y_0 + x_6y_0 + x_1y_1 + x_2y_1 + x_4y_1 + x_5y_1 + y_2 + x_1y_2 + x_3y_2 + x_4y_2 + y_3 + y_3 + x_2y_3 + x_3y_3 + x_1y_4 + x_2y_4 + y_5 + x_1y_5 + x_7y_5 + y_6 + y_6 + x_6y_6 + y_7 + x_5y_7 + x_7y_7$