On the Rapid Solution of Systems of Polynomial Equations over Low-Degree Extension Fields of $\mathbb{GF}(2)$, via SAT-Solvers.

Gregory V. Bard*

Department of Mathematics, Fordham University, Bronx, NY, 10458, USA.

Abstract. There are several ways to solve polynomial systems of equations over $\mathbb{GF}(4), \mathbb{GF}(8), \ldots, \mathbb{GF}(64)$. In algebraic cryptanalysis, one can be interested in this problem or its analog over higher-degree extensions. We introduce the method of solving these systems via SAT-solvers, as a natural extension of the work in solving polynomial systems over $\mathbb{GF}(2)$ via SAT-solvers, as shown by Bard, Courtois, and Jefferson in 2006. The SAT-solver timings were comparable with Gröbner Bases approaches, but slower than them, when the Gröbner Bases software did not crash. However, Gröbner Bases approaches often run out of memory. On the other hand, SAT-solvers do not require much memory more than required to store the problem statement. This means they allow a patient user to attack problems that are otherwise infeasible, due to a lack of memory. We further show that SAT-solvers are quite good at *underdefined* systems of equations, with half as many equations as variables. We also provide applications to graph coloring, and radio channel assignment. Finally, we discuss how to engage in distributed computing within this paradigm.

Keywords: Extension Fields of GF(2), Low-Degree Polynomial Systems of Equations, Algebraic Cryptanalysis, SAT-Solvers, Small Finite Fields.

1 Introduction

The algebraic cryptanalysis community has been interested in systems of polynomial equations over finite fields of characteristic two for quite some time. However, the finite field in question has usually been $\mathbb{GF}(2)$, with notable exceptions, including several ciphers. These include Rijndael [8] which later become the Advanced Encryption Standard (AES), which uses $\mathbb{GF}(256)$, and over the same field, the cryptosystem called TTM [13]. Another example, for $\mathbb{GF}(2^{32})$, can be found in [14]. The Courtois Toy Cipher (CTC) [1] [6] can be modeled over $\mathbb{GF}(8)$ for certain settings.

* Nearly all the work on this paper was done using the machine `sage.math.washington.edu`, supported by the NSF Grant 0555776 to Prof. William Stein. I am grateful to both Prof Stein and the NSF.

The problem itself is simple. One is given a set of polynomials over certain variables over a certain field, and one must find a root common to all of the polynomials. Usually, the sources of the equations are pairs of plaintexts and ciphertexts, known to the attacker, and since these messages were indeed sent or at least are correct, there is at least one solution. In cryptanalysis, usually one is interested only in solutions which are elements of the coefficient field, not solutions in the algebraic closure or intermediate fields. A device or algorithm which could solve this problem quickly would be able to break most block and stream ciphers that use the field in question. Therefore, modeling the complexity of this problem helps one model the security the particular stream or block cipher.

As a step toward being able to operate on systems of polynomial equations over fields like $\mathbb{GF}(2^{32})$, we present the following research on smaller extension fields. In particular, we will show how to use a classic matrix representation of these fields to efficiently produce algebraic normal forms (logical sentences) for the multiplication operation of the extension field as viewed from $\mathbb{GF}(2)$. These formulae can then be used to rapidly convert the system from over $\mathbb{GF}(2^n)$ to over $\mathbb{GF}(2)$, with n times as many variables. In the full version, we provide formulas for $\mathbb{GF}(4)$ to $\mathbb{GF}(64)$.

Once these formulae are found, the system can be solved with SAT-solvers (explained below) rather than with traditional Gröbner Bases methods, such as Singular [23] and Magma [20]. Even for small systems, Magma (using F4) would crash due to a lack of memory, for example allocating 29.9 gigabytes for a system of 8 cubic equations in 8 unknowns over $\mathbb{GF}(32)$. In general, SAT-solvers have almost no memory requirements and can be thought of as black-boxes that solve the Conjunctive Normal Form Satisfiability problem (explained below). On the other hand, while Gröbner Bases algorithms like Faugére's F4 [9] and F5 [10], as implemented in Magma, are often faster, they require tremendous quantities of memory, and so problems of interest to cryptanalysis are impossible. Meanwhile, traditional Gröbner Bases methods like the original Buchburger algorithm, implemented in Singular, are far slower than Magma. Thus SAT-solvers provide an interesting medium, for problems that are far too hard for the available memory, but simple enough to not exhaust the user's patience in running time. To help understand the balance between these three options, we have performed experiments, described below. Furthermore, we have found that if the number of equations is roughly half the number of unknowns, that SAT-solvers do quite well, much better than Magma or SAGE. We discuss options for distributed computing, where the low-memory requirement for SAT-solvers, versus the high-memory requirement for Magma, might make a crucial difference.

TOC paragraph

2 Solving $\mathbb{GF}(2)$ Systems via SAT-Solvers

Several options exist when presented with a polynomial system of equations over a finite field, in n variables. If there are nearly n^2 equations, linearization will work. If one is close to but below this threshold, then the XL algorithm [5], will work.

However, if the number of equations is roughly the number of unknowns, Gröbner Bases methods are the usual solution, or brute force guessing all the variables. There is some debate as to whether Gröbner Bases algorithms are faster than brute force in general, but in typical cryptanalytic problems, they can be faster than brute force. The traditional Buchburger algorithm (used by Singular [23], for example) for Gröbner Bases is available, as well as the algorithms F4 and F5 by Faugère (used by Magma [20], for example), which is very fast but requires a great deal of memory in practice.

In 2006, the paper [3] proposed that SAT-solvers are useful for solving polynomial systems of equations over $\mathbb{GF}(2)$, especially if the system is sparse or over-defined. In particular, the SAT-solver approach was slower than Magma if Magma did not crash for lack of memory. But it almost always crashed for this lack, for reasonable sized problems. Conversely, SAT-solvers were slower in practice but use a constant amount of memory, and so actually returned answers.

The sparsity of a polynomial system of equations is represented by β, which is the ratio of non-zero to the number of possible coefficients.So for a cubic system of m equations over n variables, with c non-zero coefficients this would be

$$\beta = \frac{c}{m\left(\binom{n}{3} + \binom{n}{2} + \binom{n}{1} + \binom{n}{0}\right)}$$

if the system is over $\mathbb{GF}(2)$. This is because there is never a need for an exponent in any monomial for a polynomial over $\mathbb{GF}(2)$, as $x^2 = x$. However, in extension fields, this becomes false, as there are x such that $x^2 \neq x$. Thus, we would add $2n$ to the denominator in the previous example to reflect monomials that are squares and cubes of a variable, as well as n^2 terms of the form $x_i^2 x_j$. Over all, however, the values of β are comparable, as these corrections are small compared to the value of the denominator for large n.

Finally, note that β is sometimes called the "density", rather than the "sparsity", to reflect that $\beta = 0.0001$ is rather sparse, and has a low density of non-zero terms.

3 SAT Solvers and $\mathbb{GF}(2)$

3.1 NP-Completeness and $\mathbb{GF}(2)$ Polynomial Systems

Solving even a quadratic system of equations over $\mathbb{GF}(2)$ is NP-hard (determining if a solution exists is NP-complete) as proven by Garety and Johnson [12], if not known earlier. Since all NP-complete problems are reducible to each other by definition, it becomes useful to imagine converting one problem to another to enable solution by approximation algorithms, heuristics, off-the-shelf libraries or other methods such as algorithms that are worse-case exponential time for large problems but fast for small or random problems (e.g. the Simplex Method). In particular, because the logical-satisfiability (SAT) problem is of commercial importance, excellent "SAT-Solvers" exist which can efficiently solve medium-sized problems quickly. The most popular at this time is MiniSAT [22], which has won several SAT-solver competitions.

The SAT problem takes any logical sentence as its input. Here a logical sentence is a collection of variables, connected by the usual operators AND, OR, NOT, XOR, IMPLIES, *et cetera*.... The decision problem is then to determine if there is some collection of assignments of true or false to each variable so that the entire logical sentence evaluates to true. A related problem is to find the actual assignment, which is NP-hard. The SAT-solver MiniSAT solves the latter, thus also solving the former.

3.2 Conjunctive and Algebraic Normal Forms

As it turns out, there is a special form, called Conjunctive Normal Form (CNF) which is very amenable to the SAT problem. Here, the logical sentence is made up of clauses, each of which is a disjunction (OR-gate). The disjunction can be composed of variables and their negations, but nothing else. Then all the clauses are combined in a large conjunction (AND-gate). This means that the entire logical sentence can be true if and only if all of its clauses are true simultaneously. All logical sentences can be written in CNF [4]. Sometimes this is called POS, or Product of Sums, viewing the conjunction as a product, and the disjunction as a sum.

Another form is algebraic normal form. The logical sentence in this case is written as a large sum, which means in this context an exclusive-OR (XOR-gate). Each term of this sum must be a conjunction of variables (not their negations), or the constant 1. But, since logical-AND is merely multiplication in $\mathbb{GF}(2)$, this means that each term is simply a multiplication of variables. Furthermore, the XOR-gate is merely addition in $\mathbb{GF}(2)$, and so the entire sentence is merely a sum of these terms. Of course, mathematicians recognize this as merely the normal way of writing a multivariate polynomial:

$$1 + x_1 + x_2x_3x_4 + x_5x_6 + \dots$$

And so a collection of polynomials over $\mathbb{GF}(2)$, each of which must be satisfied (made zero) simultaneously, can be thought of as a set of Algebraic Normal Forms, each of which must be made true simultaneously. Thus, solving a polynomial system of equations over $\mathbb{GF}(2)$ with a SAT-solver is merely a conversion from ANF to CNF, followed by a single call to the SAT-solver.

3.3 The Conversion

The conversion of a system of polynomial equations over $\mathbb{GF}(2)$ is essentially a conversion between Algebraic Normal Form into Conjunctive Normal Form.

The conversion is covered in detail in [3], but here we highlight the major steps. First, optional preprocessing can be performed. Second, terms of degree other than one will be eliminated as follows. Terms of degree higher than one, for example, xyz, propose a new variable a. Then add the logical clause below, which ties a to xyz.

$$(a = xyz) = (a \Leftrightarrow x \wedge y \wedge z) = \underbrace{(\overline{a} \vee x)(\overline{a} \vee y)(\overline{a} \vee z)(a \vee \overline{x} \vee \overline{y} \vee \overline{z})}_{\text{4 CNF Clauses}}$$

Once this is done, the constant term 1, which is the only term of degree less than zero, can be replaced by a "dummy variable" t, and a clause (t), or if you like $(t \vee t \vee t)$, which forces t to be true in any satisfying assignment.

Now, all of the polynomials are sums of terms of degree one, or more plainly, sums of variables. Logically, this is an XOR of a list of variables. For n variables, it turns out that 2^{n-1} clauses are required to express this. This is unacceptably large. Instead, it is useful to "cut" the sum into a number of smaller sums, of common length around 5 or 6, which is called "the cutting number." Then each of these smaller sums can be encoded with 16 or 32 clauses respectively, and the number of clauses now grows linearly, not exponentially, with the size of the sum. Details can be found in [3]. In this paper, we chose a cutting number of 5, and shut off all preprocessing, called "massaging" in [3].

3.4 SAT-Solvers Currently Available

MiniSAT, like most SAT-solvers, will take a SAT problem in CNF (usually said to be a CNF-SAT problem for short) and either output that the system is unsatisfiable, output a satisfying solution, or run for a very long time. Remarkably, the memory use of SAT-solvers is constant over the execution, and so no additional memory beyond that needed to store the problem and some scratch space is needed. This is in stark contrast to Faugère's algorithms, known as F4 [9] and F5 [10], that are the backbone of Gröbner Basis algorithms used in Magma [20]. The available memory for F4 or F5 is a crucial limiting factor.

Because of the commercial importance of the SAT problem in circuit layout, there have been annual SAT-solver competitions with cash prizes for several years. MiniSAT is the current champion, as of early 2008. For this reason, in the author's dissertation, MiniSAT was chosen as the primary method of solving quadratic systems of equations. This technique was first proposed in [3], and later described in [2].

Because of the memory dependance of Gröbner Bases algorithms, they would usually crash on problems large enough to be of interest to Algebraic Cryptanalysis. This was not the case with SAT-solvers, though often the running time was extremely long. It is noteworthy that when Magma did not crash, it was faster than the SAT-solver approach. And while SAT-solvers are quite fast, their running time is *highly* variable, given by a Gibrat distribution, see [3]. On the other hand, while the algorithm is randomized, there is no probability of error. If an answer is returned, it is correct.So far as the author is aware, all Gröbner Bases algorithms are fully deterministic, and so their running times are the same on repeat trials even if they are hard to predict.

3.5 Distributed Computing

The SAT-solver approach is also highly adaptable to massively parallel systems. If thousands of PCs are available, for example using something like BOINC [19] then an assumption of 10 particular values for 10 variables can be given to each PC, which must solve for the remaining variables. This makes $2^{10} = 1024$ sub-cases, which are totally independent. In the case of the 1023 false assumptions (assuming only a unique solution exists), then the result will be "unsatisfiable", but the remaining machine would come up with a solution. The author is unaware of any effort to parallelize Gröbner Bases calculations but the above technique would also work in that context also. However, if one wishes to, for example, delegate the 1024 cases to several hundred office-PCs at a university, the fact that Magma requires so much memory, while MiniSAT does not, would be a crucial difference.

4 Polynomial Systems over Extension Fields of $\mathbb{GF}(2)$

The extension field will be modeled as a quotient ring of $\mathbb{GF}(2)[\alpha]$, the ring of polynomials in a single variable α with coefficients in $\mathbb{GF}(2)$. The quotient will be in terms of the principle ideal generated by a particular polynomial, of the degree of the required extension, and irreducible over $\mathbb{GF}(2)$. Specifically, the following polynomials were used

$$\begin{array}{ll} \mathbb{GF}(4) & \alpha^2+\alpha+1 \\ \mathbb{GF}(8) & \alpha^3+\alpha+1 \\ \mathbb{GF}(16) & \alpha^4+\alpha+1 \\ \mathbb{GF}(32) & \alpha^5+\alpha^2+1 \\ \mathbb{GF}(64) & \alpha^6+\alpha+1 \end{array}$$

Of course, all finite fields of the same size are isomorphic, so the particular polynomial chosen is of little importance.

4.1 Extensions of the Coefficient Field

We should stress here that we are interested only in solutions over the coefficient field. If one is interested in solutions over an extension field of the coefficient field, then one can carry-out all operations here as if one were in the extension field from the beginning. In other words, if the coefficients are in $\mathbb{GF}(8)$ and one is interested in $\mathbb{GF}(512)$ solutions, then one can treat the entire system as over $\mathbb{GF}(512)$, since all elements of $\mathbb{GF}(8)$ are elements of $\mathbb{GF}(512)$ for the right choice of basis.

4.2 Difficulty in Bits

Some cryptographers measure the difficulty of a problem in bits, meaning the logarithm base two of the number of possible solutions. In this case, for $\mathbb{GF}(2^k)$ polynomial systems with n variables, the difficulty is kn-bits. Thus, for $\mathbb{GF}(32)$, used in our trials, one can see that 7 variables was roughly the threshold of feasibility for one hour. But that is 35 bits, an otherwise significant difficulty, as 2^{35} is roughly 32 billion.

5 Finding Efficient Arithmetic Representations via Matrices

The technique below, of representing field elements by matrices is classic, and has been known for a long time. However, we will use a property of those particular cells of the matrix which happen to be 1 for one basis element, but 0 for all other basis elements, to help us rapidly generate formulae for the finite field arithmetic.

We will use $\mathbb{GF}(16)$ as an extended example, but the same arguments work for any other choice. The companion matrix of $\alpha^4 + \alpha + 1$ has the property that when substituted for α in that equation, it results in the zero matrix. Call this matrix A. We can calculate the powers of A, which are listed below:

$$A^0 = \begin{bmatrix} 1&0&0&0 \\ 0&1&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \end{bmatrix}, A^1 = \begin{bmatrix} 0&0&0&1 \\ 1&0&0&1 \\ 0&1&0&0 \\ 0&0&1&0 \end{bmatrix}, A^2 = \begin{bmatrix} 0&0&1&0 \\ 0&0&1&1 \\ 1&0&0&1 \\ 0&1&0&0 \end{bmatrix}, A^3 = \begin{bmatrix} 0&1&0&0 \\ 0&1&1&0 \\ 0&0&1&1 \\ 1&0&0&1 \end{bmatrix}, A^4 = \begin{bmatrix} 1&0&0&1 \\ 1&1&0&1 \\ 0&1&1&0 \\ 0&0&1&1 \end{bmatrix}, \ldots$$

By virtue of the polynomial $\alpha^4 + \alpha + 1$, we know that $A^4 = -A^1 - A^0$, which since we are in characteristic two is $A + I$. Therefore α^4 and all higher powers can be discarded, and one can show that I, A, A^2, A^3 is a basis for the field. Now we can represent the element $f_0 + f_1\alpha + f_2\alpha^2 + f_3\alpha^3$ as $f_0 I + f_1 A + f_2 A^2 + f_3 A^3$, a single 4×4 matrix. This is a very old technique, and is found in the free interent textbook *Applied Abstract Algebra* by Joyner, *et al* [15], and so we will not include proofs at this point.

Consider $M = f_0 I + f_1 A + f_2 A^2 + f_3 A^3$. Observe that the M_{11} cell is 1 for A^0 but 0 for all other powers of A. Likewise, the M_{14} is 1 for A and zero for the other powers. The equivalent cell for A^2 is M_{13} and for A^3 is M_{12}. Thus after converting some fields elements into matrices, and performing a series of matrix operations on them to obtain M, we can recover the field element in $f_0 + f_1\alpha + f_2\alpha^2 + f_3\alpha^3$ form by noting that

$$f_0 = M_{11}, f_1 = M_{14}, f_2 = M_{13}, f_3 = M_{12}$$

This is significant because it might otherwise be hard to convert back from a single matrix to the four-dimensional notation. These cells that are zero for all powers of A with one exception, are "dead give-aways" of the normal field representation.

Now, we will obtain a boolean formula for the triple product $p = abc$, where a, b, and c, are all elements of $\mathbb{GF}(16)$. One can define $X = a_0 I + a_1 A + a_2 A^2 + a_3 A^3$, where $a = a_0 + a_1\alpha + a_2\alpha^2 + a_3\alpha^3$ and likewise Y and Z for b and c. Note that the a_i are elements of $\mathbb{GF}(2)$, and since $\mathbb{GF}(16)$ is a $\mathbb{GF}(2)$-vector space of dimension 4, the addition operation $T = X + Y + Z$ is just vectorial addition, with $s_i = a_i + b_i + c_i$, and $T = s_0 + s_1\alpha + s_2\alpha^2 + s_3\alpha^3$.

The multiplication operation is much more complex. Using one's favorite algebra software (the author used Maple), one can calculate $M = XYZ$ symbolically, as a matrix product. Then using the cells M_{11}, M_{14}, M_{13}, and M_{12}, one can recover

$$\begin{aligned}
p_0 &= c_0a_0b_0 + c_0a_3b_1 + c_0a_2b_2 + c_0a_1b_3 + c_1a_0b_3 + c_1a_3b_0 + c_1a_3b_3 + c_1a_2b_1 + c_1a_1b_2 \\
&\quad + c_2a_0b_2 + c_2a_3b_2 + c_2a_3b_3 + c_2a_2b_0 + c_2a_2b_3 + c_2a_1b_1 + c_3a_0b_1 + c_3a_3b_1 + c_3a_3b_2 \\
&\quad + c_3a_2b_2 + c_3a_2b_3 + c_3a_1b_0 + c_3a_1b_3 \\
p_1 &= c_0a_3b_1 + c_1a_0b_3 + c_0a_1b_3 + c_1a_0b_0 + c_2a_3b_2 + c_2a_2b_0 + c_2a_2b_3 + c_2a_1b_1 + c_2a_0b_2 \\
&\quad + c_3a_0b_1 + c_3a_3b_1 + c_3a_2b_2 + c_3a_1b_0 + c_3a_1b_3 + a_3b_3c_3 + a_0b_3c_2 + c_1a_1b_2 \\
&\quad + a_3b_0c_2 + a_2b_1c_2 + a_1b_2c_2 + a_0b_2c_3 + a_3b_2c_0 + a_2b_0c_3 + a_2b_3c_0 + a_1b_1c_3 + c_1a_3b_1 \\
&\quad + c_1a_2b_2 + c_1a_1b_3 + a_0b_1c_0 + a_1b_0c_0 + c_0a_2b_2 + c_1a_3b_0 + c_1a_3b_3 + c_1a_2b_1 \\
p_2 &= c_2a_1b_3 + a_3b_0c_3 + a_2b_1c_3 + a_1b_2c_3 + c_2a_3b_3 + c_3a_3b_2 + c_3a_2b_3 + a_0b_3c_2 + a_0b_3c_3 \\
&\quad + a_3b_0c_2 + a_2b_1c_2 + a_1b_2c_2 + a_0b_2c_3 + a_3b_2c_0 + a_2b_0c_3 + a_2b_3c_0 + a_1b_1c_3 + c_1a_3b_1 \\
&\quad + c_1a_2b_2 + c_1a_1b_3 + c_2a_0b_0 + c_2a_3b_1 + c_2a_2b_2 + a_0b_2c_0 + a_3b_3c_0 + a_2b_0c_0 + a_1b_1c_0 \\
&\quad + c_1a_0b_1 + c_1a_3b_2 + c_1a_2b_3 + c_1a_1b_0 \\
p_3 &= a_0b_0 + c_2a_1b_3 + a_3b_0c_3 + a_2b_1c_3 + a_1b_2c_3 + c_2a_3b_2 + c_2a_2b_3 + c_3a_3b_1 + c_3a_2b_2 \\
&\quad + c_3a_1b_3 + a_3b_3c_3 + a_0b_3c_0 + a_3b_0c_0 + a_2b_1c_0 + c_2a_3b_1 + c_2a_2b_2 + c_1a_3b_3 + a_1b_2c_0 \\
&\quad + c_1a_0b_2 + c_1a_2b_0 + c_1a_1b_1 + c_2a_0b_1 + c_2a_1b_0 + a_3b_3c_0 + c_1a_3b_2 + c_1a_2b_3 + a_0b_3c_3 \\
M &= p_0 + \alpha p_1 + \alpha^2 p_2 + \alpha^3 p_3
\end{aligned}$$

Likewise, for the double product, $p = ab$, one obtains

$$\begin{aligned}
p_0 &= a_0b_0 + a_3b_1 + a_2b_2 + a_1b_3 \\
p_1 &= a_0b_1 + a_3b_1 + a_3b_2 + a_2b_2 + a_2b_3 + a_1b_0 + a_1b_3 \\
p_2 &= a_0b_2 + a_3b_2 + a_3b_3 + a_2b_0 + a_2b_3 + a_1b_1 \\
p_3 &= a_0b_3 + a_3b_0 + a_3b_3 + a_2b_1 + a_1b_2
\end{aligned}$$

$$\begin{aligned}
p_0 = {} & d_0c_0a_0b_0 + d_0c_0a_3b_1 + d_0c_0a_2b_2 + d_0c_0a_1b_3 + d_0c_1a_0b_3 + d_0c_1a_3b_0 + d_0c_1a_3b_3 + d_0c_1a_2b_1 \\
& + d_0c_1a_1b_2 + d_0c_2a_0b_2 + d_0c_2a_3b_2 + d_0c_2a_3b_3 + d_0c_2a_2b_0 + d_0c_2a_2b_3 + d_0c_2a_1b_1 + d_0c_3a_0b_1 \\
& + d_0c_3a_3b_1 + d_0c_3a_3b_2 + d_0c_3a_2b_2 + d_0c_3a_2b_3 + d_0c_3a_1b_0 + d_0c_3a_1b_3 + d_1c_3a_0b_0 + d_1c_2a_3b_2 \\
& + d_1a_1b_2c_3 + d_1a_1b_2c_0 + d_1a_2b_1c_3 + d_1a_2b_1c_0 + d_1a_3b_3c_3 + d_1a_3b_3c_0 + d_1a_3b_0c_3 + d_1a_3b_0c_0 \\
& + d_1a_0b_3c_3 + d_1c_3a_1b_3 + d_1c_3a_2b_2 + d_1c_2a_1b_3 + d_1c_2a_1b_0 + d_1c_2a_2b_2 + d_1c_2a_3b_1 + d_1c_2a_0b_1 \\
& + d_1c_1a_1b_1 + d_1c_1a_2b_3 + d_1c_1a_2b_0 + d_1c_1a_3b_2 + d_1c_1a_0b_2 + d_1c_1a_3b_3 + d_1c_3a_3b_1 + d_1c_2a_2b_3 \\
& + d_1a_0b_3c_0 + d_2c_2a_3b_3 + d_2c_2a_0b_0 + d_2a_1b_2c_3 + d_2a_2b_1c_3 + 2d_2a_3b_3c_3 + d_2a_3b_3c_0 + d_2a_3b_0c_3
\end{aligned}$$

$$
\begin{aligned}
&+d_2a_0b_3c_3 + d_2c_3a_2b_3 + d_2c_3a_3b_2 + d_2a_2b_3c_0 + d_2a_2b_0c_3 + d_2a_2b_0c_0 + d_2a_3b_2c_0 + d_2a_0b_2c_3 \\
&+d_2a_0b_2c_0 + d_2a_1b_2c_2 + d_2a_2b_1c_2 + d_2a_3b_0c_2 + d_2a_0b_3c_2 + d_2c_2a_1b_3 + d_2c_2a_2b_2 + d_2c_2a_3b_1 \\
&+d_2c_1a_2b_3 + d_2c_1a_3b_2 + d_2c_1a_1b_0 + d_2c_1a_2b_2 + d_2c_1a_3b_1 + d_2c_1a_0b_1 + d_2a_1b_1c_3 + d_2a_1b_1c_0 \\
&+d_2c_1a_1b_3 + d_3c_1a_1b_2 + 2d_3c_2a_3b_3 + d_3c_2a_1b_1 + d_3c_2a_2b_3 + d_3c_1a_1b_3 + d_3c_2a_2b_0 + d_3c_0a_2b_2 \\
&+d_3c_1a_0b_0 + d_3c_2a_3b_2 + d_3c_1a_0b_3 + d_3c_0a_1b_3 + d_3c_0a_3b_1 + d_3a_3b_3c_3 + d_3c_3a_1b_3 + d_3c_3a_1b_0 \\
&+d_3c_3a_2b_2 + d_3c_2a_0b_2 + d_3c_1a_3b_0 + d_3a_2b_3c_0 + d_3a_2b_0c_3 + d_3c_1a_2b_1 + d_3a_3b_2c_0 + d_3a_0b_2c_3 \\
&+d_3a_1b_2c_2 + d_3a_2b_1c_2 + d_3a_3b_0c_2 + d_3a_0b_3c_2 + d_3c_1a_3b_3 + d_3a_1b_0c_0 + d_3a_0b_1c_0 + d_3c_3a_0b_1 \\
&+d_3c_3a_3b_1 + d_3c_1a_2b_2 + d_3c_1a_3b_1 + d_3a_1b_1c_3
\end{aligned}
$$

$$
\begin{aligned}
p_1 = {}& c_1a_0b_0d_0 + a_3b_3c_3d_0 + a_2b_3c_0d_0 + a_2b_0c_3d_0 + a_3b_2c_0d_0 + a_0b_2c_3d_0 + a_1b_2c_2d_0 + a_2b_1c_2d_0 \\
&+a_3b_0c_2d_0 + a_0b_3c_2d_0 + a_1b_0c_0d_0 + a_0b_1c_0d_0 + c_1a_2b_2d_0 + c_1a_3b_1d_0 + a_1b_1c_3d_0 + c_1a_1b_3d_0 \\
&+c_1a_0b_2d_2 + c_1a_3b_3d_2 + c_3a_3b_1d_2 + c_2a_2b_3d_2 + a_0b_3c_0d_2 + c_2a_0b_0d_3 + a_1b_2c_3d_3 + a_2b_1c_3d_3 \\
&+a_3b_3c_0d_3 + a_3b_0c_3d_3 + a_0b_3c_3d_3 + a_2b_0c_0d_3 + a_0b_2c_0d_3 + c_2a_1b_3d_3 + c_2a_2b_2d_3 + c_2a_3b_1d_3 \\
&+c_1a_2b_3d_3 + c_1a_3b_2d_3 + c_1a_1b_0d_3 + c_1a_0b_1d_3 + a_1b_1c_0d_3 + d_0c_0a_3b_1 + d_0c_0a_2b_2 + d_0c_0a_1b_3 \\
&+d_0c_1a_0b_3 + d_0c_1a_3b_0 + d_0c_1a_3b_3 + d_0c_1a_2b_1 + d_0c_1a_1b_2 + d_0c_2a_0b_2 + d_0c_2a_3b_2 + 2d_0c_2a_3b_3 \\
&+d_0c_2a_2b_0 + d_0c_2a_2b_3 + d_0c_2a_1b_1 + d_0c_3a_0b_1 + d_0c_3a_3b_1 + 2d_0c_3a_3b_2 + d_0c_3a_2b_2 + d_0c_3a_1b_0 \\
&+d_0c_3a_1b_3 + d_1c_3a_0b_0 + 2d_1c_2a_3b_2 + d_1a_1b_2c_3 + d_1a_1b_2c_0 + d_1a_2b_1c_3 + d_1a_2b_1c_0 + d_1a_3b_3c_3 \\
&+d_1a_3b_3c_0 + d_1a_3b_0c_3 + d_1a_3b_0c_0 + d_1a_0b_3c_3 + 2d_1c_3a_1b_3 + 2d_1c_3a_2b_2 + d_1c_2a_1b_3 + d_1c_2a_1b_0 \\
&+d_1c_2a_2b_2 + d_1c_2a_3b_1 + d_1c_2a_0b_1 + d_1c_1a_1b_1 + d_1c_1a_2b_3 + d_1c_1a_2b_0 + d_1c_1a_3b_2 + d_1c_1a_0b_2 \\
&+d_1a_0b_3c_0 + d_2c_2a_3b_3 + d_2c_2a_0b_0 + 2d_2a_1b_2c_3 + d_2a_3b_3c_3 + d_2c_3a_2b_3 + d_2c_3a_3b_2 + d_2a_2b_3c_0 \\
&+d_2a_2b_0c_3 + d_2a_2b_0c_0 + d_2a_3b_2c_0 + d_2a_0b_2c_3 + d_2a_0b_2c_0 + d_2a_1b_2c_2 + d_2a_2b_1c_2 + d_2a_3b_0c_2 \\
&+d_2a_0b_3c_2 + 2d_2c_2a_1b_3 + 2d_2c_2a_2b_2 + 2d_2c_2a_3b_1 + 2d_2c_1a_2b_3 + 2d_2c_1a_3b_2 + d_2c_1a_1b_0 \\
&+d_2c_1a_2b_2 + d_2c_1a_3b_1 + d_2c_1a_0b_1 + d_2a_1b_1c_3 + d_2a_1b_1c_0 + d_2c_1a_1b_3 + d_3c_1a_1b_2 + d_3c_2a_3b_3 \\
&+d_3c_2a_1b_1 + d_3c_2a_2b_3 + d_3c_2a_2b_0 + d_3c_0a_2b_2 + d_3c_1a_0b_0 + d_3c_2a_3b_2 + d_3c_1a_0b_3 + d_3c_0a_1b_3 \\
&+d_3c_0a_3b_1 + d_3a_3b_3c_3 + d_3c_3a_1b_3 + d_3c_3a_1b_0 + 3d_3c_3a_2b_3 + d_3c_3a_2b_2 + 3d_3c_3a_3b_2 + d_3c_2a_0b_2 \\
&+d_3c_1a_3b_0 + d_3c_1a_2b_1 + d_3c_1a_3b_3 + d_3a_1b_0c_0 + d_3a_0b_1c_0 + d_3c_3a_0b_1 + d_3c_3a_3b_1 + d_1c_0a_0b_0 \\
&+d_1c_0a_3b_1 + d_1c_0a_2b_2 + d_1c_0a_1b_3 + d_1c_1a_0b_3 + d_1c_1a_3b_0 + d_1c_1a_2b_1 + d_1c_1a_1b_2 + d_1c_2a_0b_2 \\
&+d_1c_2a_3b_3 + d_1c_2a_2b_0 + d_1c_2a_1b_1 + d_1c_3a_0b_1 + d_1c_3a_3b_2 + d_1c_3a_2b_3 + d_1c_3a_1b_0 + c_3a_0b_0d_2 \\
&+c_2a_3b_2d_2 + a_1b_2c_0d_2 + a_2b_1c_0d_2 + a_3b_0c_0d_2 + c_3a_1b_3d_2 + c_3a_2b_2d_2 + c_2a_1b_0d_2 + c_2a_0b_1d_2 \\
&+c_1a_1b_1d_2 + c_1a_2b_0d_2
\end{aligned}
$$

$$
\begin{aligned}
p_2 = {}& a_2b_3c_0d_0 + a_2b_0c_3d_0 + a_3b_2c_0d_0 + a_0b_2c_3d_0 + a_1b_2c_2d_0 + a_2b_1c_2d_0 + a_3b_0c_2d_0 + a_0b_3c_2d_0 \\
&+c_1a_2b_2d_0 + c_1a_3b_1d_0 + a_1b_1c_3d_0 + c_1a_1b_3d_0 + c_1a_0b_2d_2 + a_0b_3c_0d_2 + c_2a_0b_0d_3 + a_2b_0c_0d_3 \\
&+a_0b_2c_0d_3 + c_1a_1b_0d_3 + c_1a_0b_1d_3 + a_1b_1c_0d_3 + d_0c_2a_3b_3 + d_0c_3a_3b_2 + d_0c_3a_2b_3 + d_1c_2a_3b_2 \\
&+d_1a_3b_3c_3 + d_1c_3a_1b_3 + d_1c_3a_2b_2 + d_1c_1a_3b_3 + d_1c_3a_3b_1 + d_1c_2a_2b_3 + d_2c_2a_3b_3 + d_2a_1b_2c_3
\end{aligned}
$$

$$
\begin{aligned}
&+d_2a_2b_1c_3 + d_2a_3b_3c_3 + d_2a_3b_3c_0 + d_2a_3b_0c_3 + d_2a_0b_3c_3 + d_2c_3a_2b_3 + d_2c_3a_3b_2 + d_2c_2a_1b_3\\
&+d_2c_2a_2b_2 + d_2c_2a_3b_1 + d_2c_1a_2b_3 + d_2c_1a_3b_2 + d_3c_2a_3b_3 + d_3c_2a_2b_3 + d_3c_1a_1b_3 + d_3c_2a_3b_2\\
&+d_3a_3b_3c_3 + d_3c_3a_1b_3 + d_3c_3a_2b_3 + d_3c_3a_2b_2 + d_3c_3a_3b_2 + d_3a_2b_3c_0 + d_3a_2b_0c_3 + d_3a_3b_2c_0\\
&+d_3a_0b_2c_3 + d_3a_1b_2c_2 + d_3a_2b_1c_2 + d_3a_3b_0c_2 + d_3a_0b_3c_2 + d_3c_1a_3b_3 + d_3c_3a_3b_1 + d_3c_1a_2b_2\\
&+d_3c_1a_3b_1 + d_3a_1b_1c_3 + d_1c_0a_3b_1 + d_1c_0a_2b_2 + d_1c_0a_1b_3 + d_1c_1a_0b_3 + d_1c_1a_3b_0 + d_1c_1a_2b_1\\
&+d_1c_1a_1b_2 + d_1c_2a_0b_2 + d_1c_2a_2b_0 + d_1c_2a_1b_1 + d_1c_3a_0b_1 + d_1c_3a_1b_0 + c_3a_0b_0d_2 + a_1b_2c_0d_2\\
&+a_2b_1c_0d_2 + a_3b_0c_0d_2 + c_2a_1b_0d_2 + c_2a_0b_1d_2 + c_1a_1b_1d_2 + c_1a_2b_0d_2 + d_2c_0a_0b_0 + d_2c_0a_3b_1\\
&+d_2c_0a_2b_2 + d_2c_0a_1b_3 + d_2c_1a_0b_3 + d_2c_1a_3b_0 + d_2c_1a_2b_1 + d_2c_1a_1b_2 + d_2c_2a_0b_2 + d_2c_2a_2b_0\\
&+d_2c_2a_1b_1 + d_2c_3a_0b_1 + d_2c_3a_1b_0 + c_3a_0b_0d_3 + a_1b_2c_0d_3 + a_2b_1c_0d_3 + a_3b_0c_0d_3 + c_2a_1b_0d_3\\
&+c_2a_0b_1d_3 + c_1a_1b_1d_3 + c_1a_2b_0d_3 + c_1a_0b_2d_3 + a_0b_3c_0d_3 + c_2a_0b_0d_0 + a_1b_2c_3d_0 + a_2b_1c_3d_0\\
&+a_3b_3c_0d_0 + a_3b_0c_3d_0 + a_0b_3c_3d_0 + a_2b_0c_0d_0 + a_0b_2c_0d_0 + c_2a_1b_3d_0 + c_2a_2b_2d_0 + c_2a_3b_1d_0\\
&+c_1a_2b_3d_0 + c_1a_3b_2d_0 + c_1a_1b_0d_0 + c_1a_0b_1d_0 + a_1b_1c_0d_0 + d_1c_1a_0b_0 + d_1a_2b_3c_0 + d_1a_2b_0c_3\\
&+d_1a_3b_2c_0 + d_1a_0b_2c_3 + d_1a_1b_2c_2 + d_1a_2b_1c_2 + d_1a_3b_0c_2 + d_1a_0b_3c_2 + d_1a_1b_0c_0 + d_1a_0b_1c_0\\
&+d_1c_1a_2b_2 + d_1c_1a_3b_1 + d_1a_1b_1c_3 + d_1c_1a_1b_3\\
p_3 = {}& a_3b_3c_3d_0 + c_1a_3b_3d_2 + c_3a_3b_1d_2 + c_2a_2b_3d_2 + a_1b_2c_3d_3 + a_2b_1c_3d_3 + a_3b_3c_0d_3 + a_3b_0c_3d_3\\
&+a_0b_3c_3d_3 + c_2a_1b_3d_3 + c_2a_2b_2d_3 + c_2a_3b_1d_3 + c_1a_2b_3d_3 + c_1a_3b_2d_3 + d_0c_1a_3b_3 + d_0c_2a_3b_2\\
&+d_0c_2a_2b_3 + d_0c_3a_3b_1 + c_3a_0b_0d_0 + a_0b_3c_0d_0 + a_3b_0c_0d_0 + a_2b_1c_0d_0 + a_1b_2c_0d_0 + c_1a_2b_0d_0\\
&+c_1a_1b_1d_0 + c_2a_1b_0d_0 + d_1c_2a_0b_0 + d_1a_0b_2c_0 + d_1a_2b_0c_0 + d_1a_1b_1c_0 + d_1c_1a_0b_1 + d_1c_1a_1b_0\\
&+d_2c_1a_0b_0 + d_2a_0b_1c_0 + d_2a_1b_0c_0 + d_3c_0a_0b_0 + c_2a_0b_1d_0 + c_1a_0b_2d_0 + d_0c_3a_2b_2 + d_0c_3a_1b_3\\
&+d_1a_1b_2c_3 + d_1a_2b_1c_3 + 2d_1a_3b_3c_3 + d_1a_3b_3c_0 + d_1a_3b_0c_3 + d_1a_0b_3c_3 + d_1c_2a_1b_3 + d_1c_2a_2b_2\\
&+d_1c_2a_3b_1 + d_1c_1a_2b_3 + d_1c_1a_3b_2 + 2d_2c_2a_3b_3 + d_2a_3b_3c_3 + 2d_2c_3a_2b_3 + 2d_2c_3a_3b_2\\
&+d_2a_2b_3c_0 + d_2a_2b_0c_3 + d_2a_3b_2c_0 + d_2a_0b_2c_3 + d_2a_1b_2c_2 + d_2a_2b_1c_2 + d_2a_3b_0c_2 + d_2a_0b_3c_2\\
&+d_2c_1a_2b_2 + d_2c_1a_3b_1 + d_2a_1b_1c_3 + d_2c_1a_1b_3 + d_3c_1a_1b_2 + d_3c_2a_3b_3 + d_3c_2a_1b_1 + 2d_3c_2a_2b_3\\
&+d_3c_2a_2b_0 + d_3c_0a_2b_2 + 2d_3c_2a_3b_2 + d_3c_1a_0b_3 + d_3c_0a_1b_3 + d_3c_0a_3b_1 + d_3a_3b_3c_3 + 2d_3c_3a_1b_3\\
&+d_3c_3a_1b_0 + d_3c_3a_2b_3 + 2d_3c_3a_2b_2 + d_3c_3a_3b_2 + d_3c_2a_0b_2 + d_3c_1a_3b_0 + d_3c_1a_2b_1 + 2d_3c_1a_3b_3\\
&+d_3c_3a_0b_1 + 2d_3c_3a_3b_1 + d_1c_2a_3b_3 + d_1c_3a_3b_2 + d_1c_3a_2b_3 + c_2a_3b_2d_2 + c_3a_1b_3d_2 + c_3a_2b_2d_2\\
&+d_2c_0a_3b_1 + d_2c_0a_2b_2 + d_2c_0a_1b_3 + d_2c_1a_0b_3 + d_2c_1a_3b_0 + d_2c_1a_2b_1 + d_2c_1a_1b_2 + d_2c_2a_0b_2\\
&+d_2c_2a_2b_0 + d_2c_2a_1b_1 + d_2c_3a_0b_1 + d_2c_3a_1b_0 + c_3a_0b_0d_3 + a_1b_2c_0d_3 + a_2b_1c_0d_3 + a_3b_0c_0d_3\\
&+c_2a_1b_0d_3 + c_2a_0b_1d_3 + c_1a_1b_1d_3 + c_1a_2b_0d_3 + c_1a_0b_2d_3 + a_0b_3c_0d_3 + a_1b_2c_3d_0 + a_2b_1c_3d_0\\
&+a_3b_3c_0d_0 + a_3b_0c_3d_0 + a_0b_3c_3d_0 + c_2a_1b_3d_0 + c_2a_2b_2d_0 + c_2a_3b_1d_0 + c_1a_2b_3d_0 + c_1a_3b_2d_0\\
&+d_1a_2b_3c_0 + d_1a_2b_0c_3 + d_1a_3b_2c_0 + d_1a_0b_2c_3 + d_1a_1b_2c_2 + d_1a_2b_1c_2 + d_1a_3b_0c_2 + d_1a_0b_3c_2\\
&+d_1c_1a_2b_2 + d_1c_1a_3b_1 + d_1a_1b_1c_3 + d_1c_1a_1b_3
\end{aligned}
$$

which is shockingly worse. Therefore, one can see that it is essential that the process of obtaining these formulae be automated, at least partially, if one is to tackle 10th degree polynomials over $\mathbb{GF}(256)$.

We used the the triple product form in our software for the tests, and gained a speed and conciseness improvement over using just the double product. We hope to try the quadruple product shortly, but were unable to do so before the conference deadline.

6 Using the Algebraic Normal Forms

Each equation is to be independently converted into a set of $\mathbb{GF}(2)$ equations. Once this is done for all equations, the union of these sets of equations is converted into a SAT problem as described in [3]. Then a SAT-solver is called to produce a solution or declare unsatisfiability.

We used $\mathbb{GF}(32)$ in our experiments. Each equation over $\mathbb{GF}(32)$, consists of a sum of terms or monomials. Each of these monomials is a product of some number of variables and some number of constants. Using the ANFs of the product operation, as found in the previous section, we have a logical formula for each of the "bits" or $\mathbb{GF}(2)$ terms of any $\mathbb{GF}(32)$ product. Thus, we simply apply these formulas to the constant(s) and variable(s) present, if present.

Once this is done for each monomial in a polynomial, these formulae can be simply added, adding them component wise as 5 dimensional vectors. This is because the addition operation over $\mathbb{GF}(32)$ is just 5-dimensional vector addition over $\mathbb{GF}(2)$. These sums are logical formulae, and are stored in a file. There should be 5 sums per original polynomial.

These sums now form a $\mathbb{GF}(2)$ system of equations, with five times as many unknowns and equations. That, in turn, can be converted into a SAT-problem via the techniques found in [3]. In fact, the author simply used his old code without modification to do so.

6.1 Remarks on Degree

We chose quadratic equations, principally because the logical formula for a $\mathbb{GF}(32)$ quadruple product (multiplication of 4 variables) is huge. In fact, it was too large to copy down from Maple. However, for smaller fields, the 4 product formula was not as bad, and so one could write the converter for cubics as well.

It is noteworthy to mention that, by the addition of new variables, any system of equations can be written as degree 2, regardless of the original degree of the polynomials. This can be done while introducing zero spurious solutions and destroying zero original solutions.

For example, if one had originally $a+bcd = 1$, then one can say let $x = bc$, and one has $a + xd = 1$ and $x = bc$ as two equations, now of quadratic degree. One can show that if the degree of the original system of equations is fixed, that only polynomially many new variables are introduced. One can see that the order in which these new variables are introduced can have a tremendous impact on the number required.

This is discussed in [2]. However, by dropping to cubic instead of quadratic, fewer new variables would be introduced, and this would surely be more efficient for a SAT-solver. If the quadruple product formula over $\mathbb{GF}(32)$ were short enough to be cut-and-pasted, out of Maple and into our C program, we surely would have done so.

6.2 Remarks on Coefficients

Unlike $\mathbb{GF}(2)$ polynomials, the polynomials over $\mathbb{GF}(2^k)$ (with $k > 1$) have coefficients. Since a formula for a triple product is available, it makes sense for the first of the three multiplicands to be the coefficient. A short-cut makes the resulting program more efficient. One should simply delete any terms with coefficient zero. This may sound obvious but in a sparse system, it can be a large savings. Second, one could imagine 31 extra variables, one for each of the 31 extra constants. These additional variables are fixed to the constants by the insertion of 31 additional equations of the form $x_i = c_i$ for the constant c_i. Then the entire system of polynomials can have its coefficients removed and be a series of products of variables. This is very inefficient as it turns out. Instead, when c_{i1} or c_{i2} is needed, signifying the first or second bit of the constant c_i, one should substitute "0" or "1" as required.

6.3 Solving with Gröbner Bases

When solving with a Gröbner Bases algorithm, one need merely state the polynomials to be solved. They form an ideal, and all polynomials in that ideal are zero on a set of points called the variety, or set of solutions. If there is only one point in this set, then the Gröbner Bases will look like $x_1 = 1$, $x_2 = 0$, etc...

One problem is that we are only interested in solutions that have values in the coefficient field. Taking $\mathbb{GF}(32)$ as an example, there are 31 elements in the multiplicative group, which has 1 as its identity. Therefore, $x^{31} = 1$, or alternatively $x^{32} - x = 0$. The second equation has the added property that it is true for 0, the only field element excluded from the multiplicative group.

Thus, an element $x \in \overline{\mathbb{GF}(2)}$ is in the field $\mathbb{GF}(32)$ if and only if $x^{32} - x = 0$. We can add an equation of this form for each variable in the polynomial system. Then, finally, we will be restricted to solutions in the coefficient field, as desired. However, 32 is a relatively high degree, and this is why the Gröbner Bases may have performed badly in these experiments.

In Gröbner Bases approaches, it is important to note the choice of variable ordering. We used `degrevlex`, on the advice of M. Albrecht.

7 Experimental Results

The experimental results can be found in the table on Page 25.

Special Symbols in Results Table: The following special indicators are used in the results table. First, "crashed" signifies that the software aborted due to a lack of memory, usually about 30 gigabytes. Second, "$>$ 70 mins" signifies that the software exceeded the time limit that etiquette requires on a shared machine. Third, "no trial" signifies that since a smaller version of the same problem either crashed or timed out, this size was not tried.

7.1 Computers Used

For Magma and Singular, we used sage.math.washington.edu, a large-scale computer provided by the National Science Foundation for numerically-intensive research on SAGE. It is a special-purpose 64-bit computer built by Western Scientific that has 64GB of RAM and 16 AMD Opteron cores.

For MiniSAT, we used `Taylor, Legendre, Lagrange, Gauss,` and `Kummer`, ordinary PCs with 1 gigabyte of RAM and one 2 GHz processor, running Linux, at the University of Maryland Mathematics Department.

7.2 Polynomial Systems Used

The polynomials were generated randomly, and were degree 2. Every coefficient was present with probability β, and if present, had a coefficient chosen uniformly at random from the 31 available non-zero coefficients in $\mathbb{GF}(32)$. The numbers of equations, unknowns, and β is listed in the table.

First, we analyzed the $m = n$ case. This was done with $\beta = 1.0$ for dense systems, and $\beta = 0.2$ for sparse systems. While this may sound like a very pessimistic β for a sparse system, as linear systems often have $\beta = 1/1000$, one should note that in an n variable quadratic system, there are only $(n^2 + 3n + 2)/2$ possible coefficients. Thus with $n = 5$, we would have 21 possible coefficients, and a β much below 0.1 would allow for an all-zero equation somewhere in the system. As was the case with $\mathbb{GF}(2)$, the SAT-solver method does better in the sparse case than the dense. This is also the case with Gröbner Bases approaches, in fact the difference is rather dramatic.

Note that SAT-solvers need only find one solution, while Gröbner Bases solutions find a basis for the set of solutions, which might be complicated if there are several solutions. For this reason, we decided to try $n = 2m$, a system with half as many equations as variables, rounded up. One would expect several solutions in this case, but still finitely many because we are in a finite field. Here, SAT-solvers did much better than expected. Also, Singular did spectacularly bad with this problem.

8 Conclusions

The distinguishing features of this class of problems are that first, the polynomials are over fields of characteristic two, and second, that we are only interested in the base field points (sometimes called rational points). This is natural in cryptanalysis, as bits are $\mathbb{GF}(2)$ elements and a fractional or irrational bit makes little sense. Without these properties, SAT-solvers would be of little use.

Another property is that we anticipate one solution (key) in cryptanalysis, and so finding the first available solution means finding all solutions. Rarely are there two keys under which the same plaintext will become the same ciphertext. In the case of finitely many but several solutions, a Gröbner Basis must describe all of the finitely many solutions, and so would be rather complex, whereas SAT-solvers stop at the first solution. Note that infinitely many solutions in a finite field is impossible, unless there are infinitely many variables, which we do not consider here.

And so given these constraints, especially the requirement of adding the field equations

$$x_i^{2^n} - x_i = 0$$

for $\mathbb{GF}(2^n)$, renders Gröbner Bases approaches disadvantaged compared to SAT-solvers. While there is a wealth of theory about Gröbner Bases algorithms, far less is known about the Grasp algorithm and modern SAT-Solvers. Clearly, more work in both areas is waiting to be done.

Despite the advantages of SAT-solvers, Magma was always faster when it did not crash. However, the memory required was substantial, and so for large problems where the user is patient, SAT-solvers will be slow but will work. On the other hand, Magma might require too much memory to operate. It is also noteworthy to realize that Magma is actually quite expensive, whereas Singular and MiniSAT are free. The performance differences between Singular and Magma in the table are interesting, and may justify the use of expensive software in research on this topic.

9 Other Applications

9.1 Graph Coloring

Since $x^{31} = 1$ is true for all non-zero elements of $\mathbb{GF}(32)$, and false for the zero element, then $(y - z)^{31} = 1$ is satisfied if and only if $y \neq z$. Thus the simple formula $(y - z)^{31} = 1$ is an encoding of $y \neq z$ in the language of polynomials.

Suppose one desires to color a graph $G = (V, E)$ with 32 colors. Simply associate each color with a field element. Make one variable x_i for each vertex v_i, which will represent its color. If there is an edge between v_i and v_j, then add the equation $(x_i - x_j)^{31} = 1$, which will then ensure that those two nodes are not colored with the same coloring.

Usually, one is interested in a 4-coloring or a 3-coloring. This can be done as follows. If one wants to make a c-coloring, with $c < 32$, then for each of the $32 - c$ "superfluous" colorings, add a "dummy vertex", and connect it to each of the other vertexes of the graph, including the other dummy vertexes. This ensures that every dummy vertex will be colored with a color that no other vertex in the graph is using, thus removing the superfluous colors.

9.2 Radio Channel Assignments

Graph coloring problems come up in radio channel assignments, where cities near each other can cause interference, if they have radio channels on the same frequencies. This is complicated by the fact that it is not merely a geographic proximity, but mountains, for example, will block signals while the Midwestern Plains of the USA create a "clear channel" of millions of square miles.

To assign frequencies without interference, generate a vertex for each transmitting tower. If two towers t_i and t_j would interfere if they were assigned the same frequency, then draw an edge between v_i and v_j. Then perform a coloring. All towers with the same color can safely be assigned the same frequency for broadcasting.

9.3 Microprocessor Scheduling

Instructions in a modern microprocessor require several clock cycles to process. In a microprocessor with many cores, there can be a problem if one instruction depends on the answer from a previous one, because the latter might not be able to start until the previous one has finished. Graph coloring can solve this problem [16].

Imagine a graph with a vertex for each instruction. There is an edge from i to j if the instruction j depends on instruction i. The graph is then colored, for example with 32 colors if there are 32 processors. Each color can then be given to a separate core because it is composed entirely of independent instructions. It should be noted that the graph is directed and acyclic, and so the NP-complete problem of graph coloring becomes polynomial-time.

10 Acknowledgements

First, the author is very indebted to Martin Albrecht, who explained to him some syntax of SAGE. Second, the author is highly grateful for the access given to him to sage.math.washington.edu by Professor William Stein, and his forgiveness when the experiments in this paper caused the entire system to collapse (due to lack of memory for Magma). Professor Dan Bernstien gave thoughtful advice and encouragement. Most importantly, the author would like to thank Professors Janusz Golec, Cris Poor, and Mak Trifkovic, of Fordham University, for filling in during the author's lectures, so that the author could leave Fordham and attend the "Fast Software Encryption" conference FSE, where all of this paper was written. Last but not least, thanks to the National Science Foundation for NSF Grant 0555776, which allowed for this excellent machine to be at the disposal of SAGE developers.

References

1. Martin Albrecht, *Algebraic Attacks on the Courtois Toy Cipher*, M.S. Thesis (Diplomarbeit), University of Bremen (Universität Bremen), Department of Computer Science, 2006.
2. Gregory Bard. "Algorithms for the solution of linear and polynomial systems of equations over finite fields, with applications to cryptanalysis." PhD Thesis, Department of Applied Mathematics and Scientific Computation, University of Maryland at College Park, Summer of 2007.

3. Gregory Bard, Nicolas Courtois, and Christopher Jefferson. "Efficient Methods for Conversion and Solution of Sparse Systems of Low-Degree Multivariate Polynomials over GF(2) via SAT-Solvers." Cryptology ePrint Archive, Report 2007/024, 2006. Available at: http://eprint.iacr.org/2007/024.pdf
4. Thomas Cormen, Charles Leiserson, Ronald Rivest, and Clifford Stein. *Introduction to Algorithms*. Second Edition. McGraw-Hill. 2002.
5. Nicolas Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir. "E?cient algorithms for solving overdened systems of multivariate polynomial equations." In Proceedings Of Eurocrypt 2000, LNCS 1807, pages 392407. Springer, 2000.
6. Nicolas Courtois. "How fast can be algebraic attacks on block ciphers?" Cryptology ePrint Archive, Report 2006/168, 2006. Available at: http://eprint.iacr.org/2006/168.pdf
7. Nicolas Courtois. "Algebraic attacks over $\mathbb{GF}(2^k)$, application to HFE challenge 2 and sash-v2." In Public Key Cryptography: PKC 2004, pages 201217. Springer, 2004.
8. Joan Daemen and Vincent Rijmen. "Aes proposal: Rijndael", 1999. Available at http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael-ammended.pdf
9. Jean-Charles Faugère. "A new efficient algorithm for computing Gröbner basis (f4)", 1999. Available at http://modular.ucsd.edu/129-05/refs/faugeref4.pdf
10. Jean-Charles Faugère. "A new efficient algorithm for computing Gröbner bases without reduction to zero (f5)." In Proceedings Of ISSAC, pages 7583. ACM Press, 2002.
11. Niels Ferguson, Richard Schroeppel, and Doug Whiting. "A simple algebraic representation of Rijndael." Proceedings of Selected Areas in Cryptography: SAC01, Vol. 2259, *Lecture Notes in Computer Science*, Pages 103111. Springer-Verlag, 2001.
12. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. 1979.
13. Louis Goubin and Nicolas Courtois. "Cryptanalysis of the TTM Cryptosystem." ASIACRYPT 2000: 44-57
14. Thomas Jakobsen. "Cryptanalysis of block ciphers with probabilistic non-linear relations of low degree." Proceedings of CRYPTO 1998. *Lecture Notes in Computer Science 1462*. 1998.
15. D. Joyner and R. Kreminski and J. Turisco. *Applied Abstract Algebra*. Free Internet Textbook. 2002. http://web.ew.usna.edu/ wdj/book/book.html
16. Stephan Lucks. Private conversation with Stephan Lucks.
17. Stein, William, *Sage Mathematics Software (Version 2.7)*, Computer Algebra software package. The Sage Group, 2007, http://www.sagemath.org.
18. National Institute of Standards and Technology (NIST) publication U.S. FIPS PUB 197 on November 26, 2001.
19. BOINC. Berkeley Open Infrastructure for Networked Computing. http://boinc.berkeley.edu/
20. Magma. Computer Algebra software package. http://magma.maths.usyd.edu.au/magma/
21. Maple. Computer Algebra software package. http://www.maplesoft.com/
22. MiniSAT. Freely available SAT-solver. http://minisat.se/Papers.html
23. Singular. Computer Algebra software package. http://www.singular.uni-kl.de/

A Inverses and Determinants

We return to $\mathbb{GF}(16)$ and $M_4(\mathbb{GF}(2))$ as our example. Though it is not necessarily relevant to polynomial systems of equations, the determinant or inverse of the matrix $M = a_0 I + a_1 A + a_2 A^2 + a_3 A^3$ can be calculated.

A.1 Determinants

Note that the a_i are elements of $\mathbb{GF}(2)$ and so it does not make any sense for those elements to have exponents, as $1 \times 1 = 1$ and $0 \times 0 = 0$. Therefore, the formula below looks deceptively simple, because we are in $\mathbb{GF}(2)$.

$$\begin{aligned} det(M) = {} & a_0a_1a_2a_3 + a_0a_1a_2 + a_0a_1a_3 + a_0a_2a_3 + a_1a_2a_3 + a_0a_3 + a_0a_1 \\ & + a_0a_2 + a_1a_2 + a_1a_3 + a_2a_3 + a_0 + a_1 + a_2 + a_3 \end{aligned}$$

Further examination yields that this is always 1, unless $0 = a_0 = a_1 = a_2 = a_3$. However, this should make sense as we are in a field, and so each non-zero element is required to have a multiplicative inverse. Thus every non-zero element's matrix must have a determinant that is non-zero. Since the determinant is from the base field, it must be 1, the only element left in $\mathbb{GF}(2)$.

A.2 Inverses

Then we can inquire as to the inverse of M, and we can do this in the form $det(M)M^{-1}$, as this means multiplying by one, (otherwise M^{-1} does not exist).

We obtain the following 4×4 matrix.

$$\left[\begin{matrix} a_0 + a_1 + a_2 + a_3 + a_1a_2 + a_0a_2 + a_0a_1a_2 + a_1a_2a_3 & a_1 + a_2 + a_3 + a_0a_3 + a_1a_3 + a_2a_3 + a_1a_2a_3 \\ a_3 + a_0a_1 + a_0a_2 + a_1a_2 + a_1a_3 + a_0a_1a_3 & a_0 + a_0a_2 + a_0a_3 + a_1a_2 + a_1a_3 + a_2a_3 + a_0a_1a_2 \\ a_2 + a_3 + a_0a_1 + a_0a_2 + a_0a_3 + a_0a_2a_3 & a_3 + a_0a_1 + a_0a_2 + a_1a_2 + a_1a_3 + a_0a_1a_3 \\ a_1 + a_2 + a_3 + a_0a_3 + a_1a_3 + a_2a_3 + a_1a_2a_3 & a_2 + a_3 + a_0a_1 + a_0a_2 + a_0a_3 + a_0a_2a_3 \end{matrix}\right.$$

$$\left.\begin{matrix} a_2 + a_3 + a_0a_1 + a_0a_2 + a_0a_3 + a_0a_2a_3 & a_3 + a_0a_1 + a_0a_2 + a_1a_2 + a_1a_3 + a_0a_1a_3 \\ a_1 + a_0a_1 + a_0a_2 + a_1a_3 + a_2a_3 + a_0a_2a_3 + a_1a_2a_3 & a_2 + a_1a_2 + a_0a_3 + a_1a_3 + a_0a_1a_3 + a_0a_2a_3 \\ a_0 + a_0a_2 + a_0a_3 + a_1a_2 + a_1a_3 + a_2a_3 + a_0a_1a_2 & a_1 + a_0a_1 + a_0a_2 + a_1a_3 + a_2a_3 + a_0a_2a_3 + a_1a_2a_3 \\ a_3 + a_0a_1 + a_0a_2 + a_1a_2 + a_1a_3 + a_0a_1a_3 & a_0 + a_0a_2 + a_0a_3 + a_1a_2 + a_1a_3 + a_2a_3 + a_0a_1a_2 \end{matrix}\right]$$

Reading off the four cells that give us the basis coefficients from the usual spots, we learn that

$$\begin{aligned} \left(a_0 + a_1\alpha + a_2\alpha^2 + a_3\alpha^3\right)^{-1} = {} & (a_0 + a_1 + a_2 + a_3 + a_1a_2 + a_0a_2 + a_0a_1a_2 + a_1a_2a_3) + \\ & (a_3 + a_0a_1 + a_0a_2 + a_1a_2 + a_1a_3 + a_0a_1a_3)\,\alpha + \\ & (a_2 + a_3 + a_0a_1 + a_0a_2 + a_0a_3 + a_0a_2a_3)\,\alpha^2 + \\ & (a_1 + a_2 + a_3 + a_0a_3 + a_1a_3 + a_2a_3 + a_1a_2a_3)\,\alpha^3 \end{aligned}$$

A.3 Rijndael and the Para-Inverse Operation

In order to create an operation that is relatively complex, the Rijndael cipher [8], which later became AES [18], uses an operation like an inverse for elements of $\mathbb{GF}(256)$. For non-zero inputs, it is the inverse, and for the input zero, the output is zero. This is called inv0 by some authors in cryptanalysis, but we denote it here as the para-inverse. (Note that the term pseudo-inverse is already taken, and means $(A^TA)^{-1}A^T$ for rectangular matrices).

This operation can be represented by the inversion formulas in previous subsection, because if $a_0 = a_1 = a_2 = a_3 = 0$ is the input, the answer comes out 0, even though this violates the assumptions under which the formula was derived. Thus the previous formula is not only a $\mathbb{GF}(16)$ inverse, it also can serve as the para-inverse. The applications to the continued fraction model of the AES are obvious [11].

B Tables of Arithmetic Formulae

B.1 For $\mathbb{GF}(4)$

For the product $p = ab$

$$\begin{aligned} p_0 &= a_0b_0 + a_1b_1 \\ p_1 &= a_1b_0 + a_0b_1 + a_1b_1 \end{aligned}$$

For the product $p = abc$

$$\begin{aligned} p_0 &= a_0b_0c_0 + a_1b_1c_0 + c_1a_1b_0 + c_1a_0b_1 + c_1a_1b_1 \\ p_1 &= a_1b_0c_0 + a_0b_1c_0 + a_1b_1c_0 + 2c_1a_1b_1 + a_0b_0c_1 + c_1a_0b_1 + c_1a_1b_0 \end{aligned}$$

For the product $p = abcd$

$$\begin{aligned} p_0 = {} & d_0a_0b_0c_0 + d_0a_1b_1c_0 + d_0c_1a_1b_0 + d_0c_1a_0b_1 + d_0c_1a_1b_1 + d_1a_1b_0c_0 + d_1a_0b_1c_0 + d_1a_1b_1c_0 \\ & + d_1a_0b_0c_1 + d_1c_1a_0b_1 + d_1c_1a_1b_0 \\ p_1 = {} & d_1a_0b_0c_0 + 2d_1a_1b_1c_0 + 2d_1c_1a_1b_0 + 2d_1c_1a_0b_1 + 3d_1c_1a_1b_1 + a_1b_0c_0d_0 + d_1a_1b_0c_0 \\ & + a_0b_1c_0d_0 + d_1a_0b_1c_0 + d_0a_1b_1c_0 + 2d_0c_1a_1b_1 + a_0b_0c_1d_0 + d_1a_0b_0c_1 + d_0c_1a_0b_1 \\ & + d_0c_1a_1b_0 \end{aligned}$$

B.2 For $\mathbb{GF}(8)$

For the product $p = ab$

$$\begin{aligned}
p_0 &= a_0b_0 + a_2b_1 + a_1b_2 \\
p_1 &= a_0b_2 + a_2b_0 + a_2b_2 + a_1b_1 \\
p_2 &= a_0b_1 + a_2b_1 + a_2b_2 + a_1b_0 + a_1b_2
\end{aligned}$$

For the product $p = abc$

$$\begin{aligned}
p_0 = {} & c_0a_0b_0 + c_0a_2b_1 + c_0a_1b_2 + c_1a_0b_2 + c_1a_2b_0 + c_1a_2b_2 + c_1a_1b_1 + c_2a_0b_1 + c_2a_2b_1 + c_2a_2b_2 \\
& + c_2a_1b_0 + c_2a_1b_2 \\
p_1 = {} & c_1a_0b_0 + c_1a_2b_1 + c_1a_1b_2 + c_1a_0b_2 + a_0b_2c_2 + c_1a_2b_0 + a_2b_0c_2 + c_1a_2b_2 + 2c_2a_2b_2 + c_1a_1b_1 \\
& + a_1b_1c_2 + a_0b_1c_0 + c_2a_0b_1 + c_0a_2b_1 + c_2a_2b_1 + a_2b_2c_0 + a_1b_0c_0 + c_2a_1b_0 + c_0a_1b_2 + c_2a_1b_2 \\
p_2 = {} & c_2a_0b_0 + c_2a_2b_1 + c_2a_1b_2 + a_0b_2c_0 + a_0b_2c_2 + a_2b_0c_0 + a_2b_0c_2 + a_2b_2c_0 + c_2a_2b_2 + a_1b_1c_0 \\
& + a_1b_1c_2 + c_1a_0b_1 + c_1a_2b_1 + c_1a_2b_2 + c_1a_1b_0 + c_1a_1b_2
\end{aligned}$$

For the product $p = abcd$

$$\begin{aligned}
p_0 = {} & d_0c_0a_0b_0 + d_0c_0a_1b_2 + d_0c_0a_2b_1 + d_0c_1a_0b_2 + d_0c_1a_1b_1 + d_0c_1a_2b_2 + d_0c_1a_2b_0 + d_0c_2a_2b_2 \\
& + d_0c_2a_2b_1 + d_0c_2a_0b_1 + d_1c_2a_2b_1 + d_1c_2a_0b_0 + d_0c_2a_1b_2 + d_0c_2a_1b_0 + d_1a_0b_2c_2 + d_1a_0b_2c_0 \\
& + d_1c_2a_1b_2 + d_2c_2a_0b_1 + d_2c_0a_2b_1 + d_2c_2a_2b_1 + d_2a_2b_2c_0 + d_2a_1b_0c_0 + d_2c_2a_1b_0 + d_2c_0a_1b_2 \\
& + d_2c_2a_1b_2 + d_1a_1b_1c_0 + d_1c_2a_2b_2 + d_1a_2b_2c_0 + d_1a_2b_0c_2 + d_1a_2b_0c_0 + d_1c_1a_0b_1 + d_1a_1b_1c_2 \\
& + d_1c_1a_1b_0 + d_1c_1a_2b_2 + d_1c_1a_2b_1 + d_1c_1a_1b_2 + d_2c_1a_0b_2 + d_2c_1a_1b_2 + d_2c_1a_2b_1 + d_2c_1a_2b_0 \\
& + d_2a_0b_2c_2 + d_2a_2b_0c_2 + d_2c_1a_2b_2 + 2d_2c_2a_2b_2 + d_2c_1a_0b_0 + d_2c_1a_1b_1 + d_2a_1b_1c_2 + d_2a_0b_1c_0 \\
p_1 = {} & d_0c_0a_1b_2 + d_0c_0a_2b_1 + d_0c_1a_0b_2 + d_0c_1a_1b_1 + d_0c_1a_2b_2 + d_0c_1a_2b_0 + 2d_0c_2a_2b_2 + d_0c_2a_2b_1 \\
& + d_0c_2a_0b_1 + 2d_1c_2a_2b_1 + d_1c_2a_0b_0 + d_0c_2a_1b_2 + d_0c_2a_1b_0 + d_1a_0b_2c_2 + d_1a_0b_2c_0 + \\
& + d_2c_2a_0b_1 + d_2c_0a_2b_1 + 2d_2c_2a_2b_1 + 2d_2a_2b_2c_0 + d_2a_1b_0c_0 + d_2c_2a_1b_0 + d_2c_0a_1b_2 + d_1a_1b_1c_0 \\
& + 2d_1c_2a_2b_2 + d_1a_2b_2c_0 + d_1a_2b_0c_2 + d_1a_2b_0c_0 + d_1c_1a_0b_1 + d_1a_1b_1c_2 + d_1c_1a_1b_0 + d_1c_1a_2b_1 \\
& + d_1c_1a_1b_2 + a_0b_2c_0d_2 + a_2b_0c_0d_2 + a_1b_1c_0d_2 + c_1a_0b_1d_2 + d_1c_1a_1b_1 + d_1c_2a_0b_1 + d_1c_2a_1b_0 \\
& + c_2a_0b_0d_2 + c_1a_1b_0d_2 + c_1a_0b_0d_0 + c_1a_2b_1d_0 + c_1a_1b_2d_0 + a_0b_2c_2d_0 + a_2b_0c_2d_0 + a_1b_1c_2d_0 \\
& + a_0b_1c_0d_0 + a_2b_2c_0d_0 + a_1b_0c_0d_0 + d_1c_0a_2b_1 + d_2c_1a_0b_2 + d_2c_1a_2b_0 + d_2c_2a_2b_2 + d_1c_0a_1b_2 \\
& + d_2c_1a_0b_0 + d_1c_1a_2b_0 + d_2c_1a_1b_1 + 2d_2a_1b_1c_2 + d_1c_1a_0b_2 + d_1c_0a_0b_0 + d_2a_0b_1c_0 \\
p_2 = {} & d_0c_1a_2b_2 + d_0c_2a_2b_2 + d_0c_2a_2b_1 + d_1c_2a_2b_1 + d_0c_2a_1b_2 + d_1a_0b_2c_2 + d_1c_2a_1b_2 + d_2c_2a_0b_1 \\
& + d_2c_0a_2b_1 + 2d_2c_2a_2b_1 + d_2a_2b_2c_0 + d_2c_2a_1b_0 + d_2c_0a_1b_2 + 2d_2c_2a_1b_2 + d_1a_2b_2c_0 \\
& + d_1a_2b_0c_2 + d_1a_1b_1c_2 + d_1c_1a_2b_2 + d_1c_1a_2b_1 + d_1c_1a_1b_2 + a_0b_2c_0d_2 + a_2b_0c_0d_2 + a_1b_1c_0d_2 \\
& + c_1a_0b_1d_2 + d_1c_1a_1b_1 + d_1c_2a_0b_1 + d_1c_2a_1b_0 + c_2a_0b_0d_2 + c_1a_1b_0d_2 + c_1a_2b_1d_0 + c_1a_1b_2d_0
\end{aligned}$$

$$+a_0b_2c_2d_0 + a_2b_0c_2d_0 + a_1b_1c_2d_0 + a_2b_2c_0d_0 + c_2a_0b_0d_0 + a_0b_2c_0d_0 + a_2b_0c_0d_0 + a_1b_1c_0d_0$$
$$+c_1a_0b_1d_0 + c_1a_1b_0d_0 + d_1c_0a_2b_1 + d_2c_0a_0b_0 + d_2c_1a_0b_2 + d_2c_1a_1b_2 + d_2c_1a_2b_1 + d_2c_1a_2b_0$$
$$+d_2a_0b_2c_2 + d_2a_2b_0c_2 + d_1c_0a_1b_2 + d_1c_1a_0b_0 + d_1a_0b_1c_0 + d_1a_1b_0c_0 + d_1c_1a_2b_0 + d_2c_1a_1b_1$$
$$+d_2a_1b_1c_2 + d_1c_1a_0b_2$$

B.3 For $\mathbb{GF}(16)$

See the body of the paper.

B.4 For $\mathbb{GF}(32)$

For the product $p = ab$

$$p_0 = a_0b_0 + a_4b_1 + a_3b_2 + a_2b_3 + b_4a_1 + a_4b_4$$
$$p_1 = a_1b_0 + a_0b_1 + a_4b_2 + a_3b_3 + a_2b_4$$
$$p_2 = a_0b_2 + a_4b_3 + a_3b_2 + a_3b_4 + a_2b_0 + a_2b_3 + a_1b_1 + b_4a_1 + a_4b_1 + a_4b_4$$
$$p_3 = a_0b_3 + a_4b_4 + a_3b_0 + a_3b_3 + a_2b_1 + a_2b_4 + a_1b_2 + a_4b_2$$
$$p_4 = a_0b_4 + a_4b_0 + a_3b_1 + a_3b_4 + a_2b_2 + a_1b_3 + a_4b_3$$

For the product $p = abc$

$$p_0 = c_1a_4b_3 + c_0a_2b_3 + c_4a_4b_2 + c_4a_4b_0 + c_1a_4b_0 + c_3a_3b_2 + c_1a_2b_2 + c_3a_1b_1 + c_4a_0b_4 + c_2a_3b_0$$
$$+c_3a_2b_0 + c_2b_2a_1 + c_3a_3b_4 + c_3a_4b_1 + c_2a_0b_3 + c_0a_4b_1 + c_1a_3b_4 + c_2a_4b_2 + c_3a_2b_3 + c_3b_4a_1$$
$$+c_0a_4b_4 + c_4a_4b_3 + c_2a_2b_4 + c_4a_3b_3 + c_4a_1b_0 + c_2a_4b_4 + c_2a_2b_1 + c_1a_3b_1 + c_2a_3b_3 + c_3a_4b_4$$
$$+c_4a_3b_4 + c_0a_0b_0 + c_4a_2b_4 + c_1a_0b_4 + c_3a_0b_2 + c_1b_3a_1 + c_4b_3a_1 + c_4a_2b_2 + c_0b_4a_1 + c_4a_0b_1$$
$$+c_3a_4b_3 + c_4a_3b_1 + c_0a_3b_2$$
$$p_1 = c_0a_2b_4 + c_0a_0b_1 + c_0a_1b_0 + c_3a_0b_3 + c_1a_4b_1 + c_3a_1b_2 + c_1a_4b_4 + c_2a_1b_3 + c_0a_4b_2 + c_2a_0b_4$$
$$+c_4a_4b_3 + c_1a_3b_2 + c_1a_0b_0 + c_3a_4b_2 + c_3a_2b_4 + c_2a_3b_1 + c_3a_4b_4 + c_4a_3b_4 + c_2a_4b_0 + c_3a_2b_1$$
$$+c_3a_3b_3 + c_2a_3b_4 + c_1a_2b_3 + c_2a_4b_3 + c_2a_2b_2 + c_4a_1b_1 + c_4a_1b_4 + c_4a_0b_2 + c_4a_4b_1 + c_4a_4b_4$$
$$+c_4a_3b_2 + c_4a_2b_0 + c_4a_2b_3 + c_3a_3b_0 + c_1a_1b_4 + c_0a_3b_3$$
$$p_2 = c_1a_4b_3 + c_0a_2b_3 + c_4a_4b_0 + c_1a_4b_0 + c_3a_3b_2 + c_2a_1b_4 + c_1a_2b_2 + c_3a_1b_1 + c_4a_0b_4$$
$$+c_2a_3b_0 + c_3a_2b_0 + c_2b_2a_1 + c_3a_2b_2 + 2c_3a_3b_4 + c_3a_4b_1 + c_2a_0b_3 + c_0a_4b_1 + c_1a_3b_4 + c_2a_4b_2$$
$$+c_3a_2b_3 + c_3b_4a_1 + c_3a_0b_4 + c_0a_4b_4 + c_4a_4b_3 + c_2a_2b_4 + 2c_4a_3b_3 + c_4a_1b_0 + c_3a_1b_3 + 2c_2a_4b_4$$
$$+c_2a_2b_1 + c_1a_3b_1 + c_2a_3b_3 + c_2a_4b_1 + c_3a_4b_4 + a_3b_0c_4 + c_4a_3b_4 + a_2b_1c_4 + c_3a_4b_0 + 2c_4a_2b_4$$
$$+c_1a_0b_4 + c_3a_0b_2 + c_1b_3a_1 + c_4b_3a_1 + c_4a_2b_2 + c_0b_4a_1 + c_4a_0b_1 + c_2a_3b_2 + 2c_3a_4b_3 + c_4a_3b_1$$
$$+a_1b_2c_4 + c_4a_4b_4 + c_2a_2b_3 + c_0a_3b_2 + a_0b_3c_4 + c_2a_0b_0 + c_3a_3b_1 + a_1b_1c_0 + a_0b_2c_0 + a_4b_3c_0$$
$$+a_3b_4c_0 + a_2b_0c_0 + a_0b_1c_1 + a_4b_2c_1 + a_3b_3c_1 + a_2b_4c_1 + a_1b_0c_1$$

$$
\begin{aligned}
p_3 = {} & c_1a_4b_3 + c_0a_2b_4 + c_3a_0b_3 + c_4a_4b_0 + c_1a_4b_1 + c_3a_1b_2 + c_3a_3b_2 + c_1a_4b_4 + c_4a_0b_4 + c_3a_4b_1 \\
& + c_3a_0b_0 + c_2a_1b_3 + c_1a_3b_4 + c_2a_4b_2 + c_0a_4b_2 + c_3a_2b_3 + c_3b_4a_1 + c_2a_0b_4 + c_0a_4b_4 + c_1a_3b_2 \\
& + c_2a_2b_4 + c_3a_4b_2 + c_3a_2b_4 + c_2a_3b_3 + c_2a_3b_1 + 2c_3a_4b_4 + 2c_4a_3b_4 + c_2a_4b_0 + c_3a_2b_1 + c_3a_3b_3 \\
& + c_4b_3a_1 + c_4a_2b_2 + c_2a_3b_4 + c_4a_3b_1 + c_1a_2b_3 + c_2a_4b_3 + c_2a_0b_1 + c_2a_1b_0 + a_0b_3c_0 + a_3b_0c_0 \\
& + a_2b_1c_0 + b_2a_1c_0 + a_0b_2c_1 + a_2b_0c_1 + a_1b_1c_1 + c_2a_2b_2 + c_4a_1b_1 + c_4a_1b_4 + c_4a_0b_2 + c_4a_4b_1 \\
& + c_4a_4b_4 + c_4a_3b_2 + c_4a_2b_0 + c_4a_2b_3 + c_3a_3b_0 + c_1a_1b_4 + c_0a_3b_3 \\
p_4 = {} & c_4a_4b_2 + c_2a_1b_4 + c_1a_4b_4 + c_3a_2b_2 + c_3a_3b_4 + c_3a_0b_4 + c_4a_3b_3 + c_3a_1b_3 + c_2a_4b_4 + c_3a_4b_2 \\
& + c_0b_3a_1 + c_3a_2b_4 + c_4a_0b_0 + c_2a_4b_1 + a_3b_0c_4 + a_2b_1c_4 + c_3a_4b_0 + c_3a_3b_3 + c_4a_2b_4 + c_2a_3b_4 \\
& + c_2a_3b_2 + c_3a_4b_3 + c_2a_4b_3 + a_1b_2c_4 + c_4a_1b_4 + c_4a_4b_1 + 2c_4a_4b_4 + c_4a_3b_2 + c_4a_2b_3 + c_2a_2b_3 \\
& + a_0b_3c_4 + c_3a_3b_1 + c_0a_0b_4 + c_0a_4b_0 + c_0a_3b_1 + a_4b_3c_0 + a_3b_4c_0 + a_4b_2c_1 + a_3b_3c_1 + a_2b_4c_1 \\
& + c_0a_2b_2 + a_0b_3c_1 + a_3b_0c_1 + a_2b_1c_1 + b_2a_1c_1 + c_2a_0b_2 + c_2a_2b_0 + c_2a_1b_1 + c_3a_0b_1 + c_3a_1b_0
\end{aligned}
$$

For the product $p = abcd$, the formulae were too large to be efficiently copied from Maple, unfortunately.

B.5 For $\mathbb{GF}(64)$

For the product $p = ab$

$$
\begin{aligned}
p_0 &= a_0b_0 + a_5b_1 + a_4b_2 + a_3b_3 + a_2b_4 + a_1b_5 \\
p_1 &= a_0b_1 + a_5b_1 + a_5b_2 + a_4b_2 + a_4b_3 + a_3b_3 + a_3b_4 + a_2b_4 + a_2b_5 + a_1b_0 + a_1b_5 \\
p_2 &= a_0b_2 + a_5b_2 + a_5b_3 + a_4b_3 + a_4b_4 + a_3b_4 + a_3b_5 + a_2b_0 + a_2b_5 + a_1b_1 \\
p_3 &= a_0b_3 + a_5b_3 + a_5b_4 + a_4b_4 + a_4b_5 + a_3b_0 + a_3b_5 + a_2b_1 + a_1b_2 \\
p_4 &= a_0b_4 + a_5b_4 + a_5b_5 + a_4b_0 + a_4b_5 + a_3b_1 + a_2b_2 + a_1b_3 \\
p_5 &= a_0b_5 + a_5b_0 + a_5b_5 + a_4b_1 + a_3b_2 + a_2b_3 + a_1b_4
\end{aligned}
$$

For the product $p = abc$

$$
\begin{aligned}
p_0 = {} & c_0a_0b_0 + c_0a_5b_1 + c_0a_4b_2 + c_0a_3b_3 + c_0a_2b_4 + c_0a_1b_5 + c_1a_0b_5 + c_1a_5b_0 + c_1a_5b_5 + c_1a_4b_1 \\
& + c_1a_3b_2 + c_1a_2b_3 + c_1a_1b_4 + c_2a_0b_4 + c_2a_5b_4 + c_2a_5b_5 + c_2a_4b_0 + c_2a_4b_5 + c_2a_3b_1 + c_2a_2b_2 \\
& + c_2a_1b_3 + c_3a_0b_3 + c_3a_5b_3 + c_3a_5b_4 + c_3a_4b_4 + c_3a_4b_5 + c_3a_3b_0 + c_3a_3b_5 + c_3a_2b_1 + c_3a_1b_2 \\
& + c_4a_0b_2 + c_4a_5b_2 + c_4a_5b_3 + c_4a_4b_3 + c_4a_4b_4 + c_4a_3b_4 + c_4a_3b_5 + c_4a_2b_0 + c_4a_2b_5 + c_4a_1b_1 \\
& + c_5a_0b_1 + c_5a_5b_1 + c_5a_5b_2 + c_5a_4b_2 + c_5a_4b_3 + c_5a_3b_3 + c_5a_3b_4 + c_5a_2b_4 + c_5a_2b_5 + c_5a_1b_0 \\
& + c_5a_1b_5 \\
p_1 = {} & c_5a_5b_1 + c_5a_4b_2 + c_5a_3b_3 + c_5a_2b_4 + c_5a_1b_5 + c_1a_5b_5 + c_2a_5b_4 + c_2a_4b_5 + c_3a_5b_3 + c_3a_4b_4 \\
& + c_3a_3b_5 + c_4a_5b_2 + 2a_4b_4c_4 + a_4b_5c_4 + a_3b_0c_3 + a_3b_0c_4 + 2a_3b_5c_4 + a_2b_1c_3 + a_2b_1c_4 + a_1b_2c_3 \\
& + a_1b_2c_4 + a_0b_2c_4 + a_0b_2c_5 + 2a_5b_2c_5 + a_5b_3c_5 + 2a_4b_3c_5 + a_4b_4c_5 + 2a_3b_4c_5 + a_3b_5c_5 + a_2b_0c_4
\end{aligned}
$$

$$
\begin{aligned}
&+a_2b_0c_5 + 2a_2b_5c_5 + a_1b_1c_4 + a_1b_1c_5 + a_0b_1c_0 + a_0b_1c_5 + a_5b_1c_0 + a_5b_2c_0 + a_4b_2c_0 + a_4b_3c_0 \\
&+a_3b_3c_0 + a_3b_4c_0 + a_2b_4c_0 + a_2b_5c_0 + a_1b_0c_0 + a_1b_0c_5 + a_1b_5c_0 + c_4a_4b_3 + c_4a_3b_4 + c_4a_2b_5 \\
&+c_1a_0b_0 + c_1a_5b_1 + c_1a_4b_2 + c_1a_3b_3 + c_1a_2b_4 + c_1a_1b_5 + a_0b_5c_1 + a_0b_5c_2 + a_5b_0c_1 + a_5b_0c_2 \\
&+2a_5b_5c_2 + a_4b_1c_1 + a_4b_1c_2 + a_3b_2c_1 + a_3b_2c_2 + a_2b_3c_1 + a_2b_3c_2 + a_1b_4c_1 + a_1b_4c_2 + a_0b_4c_2 \\
&+a_0b_4c_3 + 2a_5b_4c_3 + a_5b_5c_3 + a_4b_0c_2 + a_4b_0c_3 + 2a_4b_5c_3 + a_3b_1c_2 + a_3b_1c_3 + a_2b_2c_2 + a_2b_2c_3 \\
&+a_1b_3c_2 + a_1b_3c_3 + a_0b_3c_3 + a_0b_3c_4 + 2a_5b_3c_4 + a_5b_4c_4 \\
p_2 = {}&c_1a_0b_1 + a_5b_0c_3 + a_0b_5c_3 + c_2a_1b_5 + c_2a_2b_4 + c_2a_3b_3 + c_2a_4b_2 + c_2a_5b_1 + c_2a_0b_0 + a_5b_4c_5 \\
&+a_0b_3c_5 + a_1b_3c_4 + a_2b_2c_4 + a_3b_1c_4 + a_4b_0c_4 + a_5b_5c_4 + a_0b_4c_4 + a_1b_4c_3 + a_2b_3c_3 + a_3b_2c_3 \\
&+a_4b_1c_3 + c_1a_1b_0 + c_1a_2b_5 + c_1a_3b_4 + c_1a_4b_3 + c_1a_5b_2 + a_1b_1c_0 + a_2b_0c_0 + a_3b_5c_0 + a_4b_4c_0 \\
&+a_5b_3c_0 + a_0b_2c_0 + a_1b_2c_5 + a_2b_1c_5 + a_3b_0c_5 + a_4b_5c_5 + a_4b_4c_4 + 2a_4b_5c_4 + a_3b_0c_4 + a_3b_5c_4 \\
&+a_2b_1c_4 + a_1b_2c_4 + a_0b_2c_5 + a_5b_2c_5 + 2a_5b_3c_5 + a_4b_3c_5 + 2a_4b_4c_5 + a_3b_4c_5 + 2a_3b_5c_5 \\
&+a_2b_0c_5 + a_2b_5c_5 + a_1b_1c_5 + a_5b_2c_0 + a_4b_3c_0 + a_3b_4c_0 + a_2b_5c_0 + c_1a_5b_1 + c_1a_4b_2 + c_1a_3b_3 \\
&+c_1a_2b_4 + c_1a_1b_5 + a_0b_5c_2 + a_5b_0c_2 + a_5b_5c_2 + a_4b_1c_2 + a_3b_2c_2 + a_2b_3c_2 + a_1b_4c_2 + a_0b_4c_3 \\
&+a_5b_4c_3 + 2a_5b_5c_3 + a_4b_0c_3 + a_4b_5c_3 + a_3b_1c_3 + a_2b_2c_3 + a_1b_3c_3 + a_0b_3c_4 + a_5b_3c_4 + 2a_5b_4c_4 \\
p_3 = {}&a_5b_5c_5 + a_5b_0c_3 + a_0b_5c_3 + c_2a_1b_5 + c_2a_2b_4 + c_2a_3b_3 + c_2a_4b_2 + c_2a_5b_1 + 2a_5b_4c_5 + a_0b_3c_5 \\
&+a_1b_3c_4 + a_2b_2c_4 + a_3b_1c_4 + a_4b_0c_4 + 2a_5b_5c_4 + a_0b_4c_4 + a_1b_4c_3 + a_2b_3c_3 + a_3b_2c_3 + a_4b_1c_3 \\
&+c_1a_2b_5 + c_1a_3b_4 + c_1a_4b_3 + c_1a_5b_2 + a_3b_5c_0 + a_4b_4c_0 + a_5b_3c_0 + a_1b_2c_5 + a_2b_1c_5 + a_3b_0c_5 \\
&+c_2a_0b_1 + c_2a_5b_2 + c_2a_4b_3 + c_2a_3b_4 + c_2a_2b_5 + c_2a_1b_0 + a_4b_5c_4 + a_5b_3c_5 + a_4b_4c_5 + a_3b_5c_5 \\
&+a_5b_5c_3 + a_5b_4c_4 + a_4b_0c_5 + a_3b_1c_5 + a_2b_2c_5 + a_1b_3c_5 + a_0b_3c_0 + a_5b_4c_0 + a_4b_5c_0 + a_3b_0c_0 \\
&+a_2b_1c_0 + a_1b_2c_0 + c_1a_0b_2 + c_1a_5b_3 + c_1a_4b_4 + c_1a_3b_5 + c_1a_2b_0 + c_1a_1b_1 + c_3a_0b_0 + c_3a_5b_1 \\
&+c_3a_4b_2 + c_3a_3b_3 + c_3a_2b_4 + c_3a_1b_5 + a_0b_5c_4 + a_5b_0c_4 + a_4b_1c_4 + a_3b_2c_4 + a_2b_3c_4 + a_1b_4c_4 \\
&+a_0b_4c_5 \\
p_4 = {}&a_0b_5c_5 + a_5b_0c_5 + a_5b_5c_0 + 2a_5b_5c_5 + a_4b_1c_5 + a_3b_2c_5 + a_2b_3c_5 + a_1b_4c_5 + c_1a_5b_4 + c_1a_4b_5 \\
&+a_5b_4c_5 + a_5b_5c_4 + a_4b_5c_5 + c_2a_5b_2 + c_2a_4b_3 + c_2a_3b_4 + c_2a_2b_5 + c_2a_5b_3 + c_2a_4b_4 + c_2a_3b_5 \\
&+c_3a_5b_2 + c_3a_4b_3 + c_3a_3b_4 + c_3a_2b_5 + c_4a_5b_1 + c_4a_4b_2 + c_4a_0b_0 + a_0b_4c_0 + a_4b_0c_0 + a_3b_1c_0 \\
&+a_2b_2c_0 + a_1b_3c_0 + c_1a_0b_3 + c_1a_3b_0 + c_1a_2b_1 + c_1a_1b_2 + c_2a_0b_2 + c_2a_2b_0 + c_2a_1b_1 + c_3a_0b_1 \\
&+c_3a_1b_0 + c_4a_3b_3 + c_4a_2b_4 + c_4a_1b_5 + a_4b_0c_5 + a_3b_1c_5 + a_2b_2c_5 + a_1b_3c_5 + a_5b_4c_0 + a_4b_5c_0 \\
&+c_1a_5b_3 + c_1a_4b_4 + c_1a_3b_5 + c_3a_5b_1 + c_3a_4b_2 + c_3a_3b_3 + c_3a_2b_4 + c_3a_1b_5 + a_0b_5c_4 + a_5b_0c_4 \\
&+a_4b_1c_4 + a_3b_2c_4 + a_2b_3c_4 + a_1b_4c_4 + a_0b_4c_5 \\
p_5 = {}&a_0b_5c_5 + a_5b_0c_5 + a_5b_5c_0 + a_5b_5c_5 + a_4b_1c_5 + a_3b_2c_5 + a_2b_3c_5 + a_1b_4c_5 + c_1a_5b_4 + c_1a_4b_5 \\
&+c_5a_0b_0 + c_5a_5b_1 + c_5a_4b_2 + c_5a_3b_3 + c_5a_2b_4 + c_5a_1b_5 + a_5b_0c_0 + c_2a_5b_3 + c_2a_4b_4 + c_2a_3b_5 \\
&+c_3a_5b_2 + c_3a_4b_3 + c_3a_3b_4 + c_3a_2b_5 + c_4a_5b_1 + c_4a_4b_2 + c_4a_3b_3 + c_4a_2b_4 + c_4a_1b_5 + a_0b_5c_0 \\
&+a_4b_1c_0 + a_3b_2c_0 + a_2b_3c_0 + a_1b_4c_0 + c_1a_0b_4 + c_1a_5b_5 + c_1a_4b_0 + c_1a_3b_1 + c_1a_2b_2 + c_1a_1b_3
\end{aligned}
$$

$$+c_2a_5b_4 + c_2a_4b_5 + c_2a_3b_0 + c_2a_2b_1 + c_2a_1b_2 + c_3a_0b_2 + c_3a_5b_3 + c_3a_4b_4 + c_3a_3b_5 + c_3a_2b_0$$
$$+c_3a_1b_1 + c_4a_0b_1 + c_2a_0b_3 + c_4a_5b_2 + c_4a_4b_3 + c_4a_3b_4 + c_4a_2b_5 + c_4a_1b_0$$

For the product $p = abcd$, the formulae were too large to be efficiently copied from Maple, unfortunately.

C Experimental Results

Num Vars	Num Eqns	β or Sparsity	Magma	Singular	SAT
2	2	1.0	0.02 sec	0.01 sec	0.01 sec
3	3	1.0	0.04 sec	0.04 sec	0.07 sec
4	4	1.0	0.43 sec	423.48 sec	213.56 sec
5	5	1.0	4.32 sec	>75 mins	19278.9 sec
6	6	1.0	42.78 sec	no trial	>75 mins
7	7	1.0	1139.8 sec	no trial	no trial
8	8	1.0	crashed[a]	no trial	no trial
9	9	1.0	no trial	no trial	no trial
4	4	0.2	0.03 sec	0.08 sec	0.02 sec
5	5	0.2	0.55 sec	14.89 sec	61.89 sec
6	6	0.2	10.04 sec	6.74 sec	0.03 sec[b]
7	7	0.2	52.89 sec	>70 mins	4111.71 sec
8	8	0.2	crashed[c]	no trial	>75 mins
9	9	0.2	no trial	no trial	no trial
4	4	1.0	20.39 sec	>70 mins	0.14 sec
5	5	1.0	192.69 sec	no trial	20.51 sec
6	6	1.0	>70 mins	no trial	17.44 sec
7	7	1.0	no trial	no trial	5388.9 sec
8	8	1.0	no trial	no trial	no trial

Experimental Results: Magma, Singular, and Mini-SAT.

[a] At one point the process had allocated 29.9 Gigabytes of RAM.
[b] This phenomenon remains unexplained, but is reproducible on repeated trials.
[c] At one point the process had allocated 24.9 Gigabytes of RAM.