Efficient Methods for Conversion and Solution of Sparse Systems of Low-Degree Multivariate Polynomials over $GF(2)$ via SAT-Solvers

Gregory V. Bard[1], Nicolas T. Courtois[2], and Chris Jefferson[3]

[1] University of Maryland, USA, gregory.bard@ieee.org
[2] University College of London, UK, n.courtois@ucl.ac.uk
[3] Oxford University, UK, chris.jefferson@comlab.ox.ac.uk

Abstract. The computational hardness of solving large systems of sparse and low-degree multivariate equations is a necessary condition for the security of most modern symmetric cryptographic schemes. Notably, most cryptosystems can be implemented with inexpensive hardware, and have a low gate counts, resulting in a sparse system of equations, which in turn renders such attacks feasible. On one hand, numerous recent papers on the XL algorithm and more sophisticated Gröbner-bases techniques [5, 7, 13, 14] demonstrate that systems of equations are efficiently solvable when they are sufficiently overdetermined or have a hidden internal algebraic structure that implies the existence of some useful algebraic relations. On the other hand, most of this work, as well as most successful algebraic attacks, involve dense, not sparse systems, at least until linearization by XL or a similar algorithm. No polynomial-system-solving algorithm we are aware of, demonstrates that a significant benefit is obtained from the extreme sparsity of some systems of equations.
In this paper, we study methods for efficiently converting systems of low-degree sparse multivariate equations into a conjunctive normal form satisfiability (CNF-SAT) problem, for which excellent heuristic algorithms have been developed in recent years. A direct application of this method gives very efficient results: we show that sparse multivariate quadratic systems (especially if over-defined) can be solved much faster than by exhaustive search if $\beta \leq 1/100$. In particular, our method requires no additional memory beyond that required to store the problem, and so often terminates with an answer for problems that cause Magma and Singular to crash. On the other hand, if Magma or Singular do not crash, then they tend to be faster than our method, but this case includes only the smallest sample problems.

Keywords: Algebraic Cryptanalysis, over-defined sparse multivariate systems of equations, logical cryptanalysis, SAT solvers, MQ, quadratic polynomials over GF(2).

1 Introduction

It is well known that the problem of solving a multivariate simultaneous system of quadratic equations over $GF(2)$ (the MQ problem) is NP-hard. Another NP-hard problem is finding a satisfying assignment for a logical expression in several variables (the SAT problem). Inspired by the possibility that either could be an efficient tool for the solution of the other, since all NP-Complete problems are polynomially equivalent, we began this investigation.

There exist several off-the-shelf SAT-solvers, such as MiniSAT [12], which can solve even relatively large SAT problems on an ordinary PC. We investigate the use of SAT-solvers as a tool for solving a random MQ problem. In particular, we show that if the system of equations is sparse or over-defined, then the SAT-solver technique works faster than brue-force exhaustive search. If the system is both sparse and over-defined, then the systems can be solved quite effectively (See Section 5).

In Section 1.1 we describe how this work applies to algebraic cryptanalysis. We define some notation and terms in Section 2, and describe the method of conversion of MQ problems into CNF-SAT problems in Section 3. A brief overview of SAT solvers is given in Section 4 and our results

in Section 5 and review previous work in Section 6. Finally, we note possible applications to cubic systems in Appendix A.

1.1 Application to Cryptanalysis

Algebraic Cryptanalysis can be summarized as a two-step process. First, given a cipher system, one converts it into a system of equations. Second, the system of equations is solved to retrieve either a key or a plaintext. Furthermore, note that all systems of equations over finite fields can be written as polynomial systems.

As pointed out in Courtois and Pieprzyk [7], this system of equations will be sparse, since efficient implementations of real-world systems require a low gate-count. In practice, the systems are very sparse—the system used to break six rounds of DES in [6] has β near zero. It is also known that any system of any degree can be written as a degree 2 system. This is done by using the following step, repeatedly:

$$\{m = wxyz\} \Rightarrow \{a = wx; b = yz; m = ab\}$$

Finally, it is usually the case that one can write additional equations by assuming that a particular number of plaintext-ciphertext pairs are available. While this is not literally unbounded, as many stream ciphers have a limit of 2^{40} bits before a new key is required, generally one has an over-abundance of equations. Therefore, we include in this study only systems where the number of equations is greater than or equal to the number of unknowns.

2 Notation and Definitions

An instance of the MQ problem is a set of functions

$$f_1(x_1, \ldots, x_n) = y_1, f_2(x_1, \ldots, x_n) = y_2, \ldots, f_m(x_1, \ldots, x_n) = y_m$$

where each f_i is a second degree polynomial. By adjusting the constant term of each polynomial, it becomes sufficient to consider only those problems with $y_j = 0$ for all j. Note, n is the number of variables and m is the number equations.

If we define $c = m/n$ or $cn = m$, then $c = 1$ will imply an exactly defined system, $c > 1$ an over-defined system and $c < 1$ an under-defined system. We will not consider under-defined systems here. The value of c will be called "the over-definition" of a system. Let M denote the number of possible monomials, including the constant monomial. Since we consider only quadratic polynomials (except for Section A on cubics)

$$M = \binom{n}{2} + \binom{n}{1} + 1$$

The system will be generated by flipping a weighted coin for each of the M coefficients for each equation. The value $\beta \in (0, 1]$ will be called the sparsity, and is the probability that a randomly selected coefficient is non-zero (equal to one). If $\beta << 1/2$, the system is considered sparse.

An instance of the Conjunctive Normal Form SAT or CNF-SAT problem is a set of clauses. Each clause is a large disjunction (OR-gate) of several variables, which can appear negated or not negated. If a set of values for all n variables makes every clause evaluate to true, then it is said to be a satisfying assignment. In this way, the set of clauses can be thought of as one long logical expression, namely a conjunction (AND-gate) of all the clauses.

3 Converting MQ to SAT

3.1 The Conversion

The conversion proceeds by three major steps. First, some preprocessing might be performed to make the system more amenable to this conversion (more detail will follow). Next, the system of polynomials will be converted to a (larger) linear system and a set of CNF clauses that render each monomial equivalent to a variable in that linear system. Lastly, the linear system will be converted to an equivalent set of clauses.

Minor Technicality The CNF form does not have any constants. Adding the clause consisting of (T), or equivalently $(T \vee T \vee \cdots \vee T)$, would require the variable T to be true in any satisfying solution, since all clauses must be true in such a solution. Once this is done, the variable T will serve the place of the constant 1, and if needed, the variable $\bar{T}$ will serve the place of the constant 0.

Step One: From a Polynomial System to a Linear System Based on the above technicality, we can consider the constant term 1 to be a variable. After that, every polynomial is now a sum of linear and higher degree terms. Those terms of higher degree will be handled as follows.

The logical expression

$$(w \vee \bar{a})(x \vee \bar{a})(y \vee \bar{a})(z \vee \bar{a})(a \vee \bar{w} \vee \bar{x} \vee \bar{y} \vee \bar{z})$$

is tautologically equivalent to $a \iff (w \wedge x \wedge y \wedge z)$, or the $GF(2)$ equation $m = wxyz$. Therefore, for each monomial of degree $d > 1$ that appears in the system of equations, we shall introduce one dummy variable. One can see that $d + 1$ clauses are required, and the total length of those clauses is $3d + 1$.

Obviously, if a monomial appears more than once, there is no need to encode it twice, but instead, it should be replaced by its previously defined dummy variable. On the other hand, in a large system, particularly an over-defined one, it is likely that every possible monomial appears at least once in some equation in the system. Therefore we will assume this is the case, but in extremely sparse systems that are not very over-defined, this is pessimistic, particularly for high degree systems.

Step Two: From a Linear System to a Conjunctive Normal Form Expression Each polynomial is now a sum of variables, or equivalently a logical XOR. Unfortunately, long XORs are known to be hard problems for SAT solvers [9]. In particular, the sum $(a + b + c + d) = 0$ is equivalent to

$$\begin{aligned}&(\bar{a} \vee b \vee c \vee d)(a \vee \bar{b} \vee c \vee \bar{d})(a \vee b \vee \bar{c} \vee d)(a \vee b \vee c \vee \bar{d}) \\ &(\bar{a} \vee \bar{b} \vee \bar{c} \vee d)(\bar{a} \vee \bar{b} \vee c \vee \bar{d})(\bar{a} \vee b \vee \bar{c} \vee \bar{d})(a \vee \bar{b} \vee \bar{c} \vee \bar{d})\end{aligned} \tag{1}$$

which is to say, all arrangements of the four variables, with one and three negations, or all odd numbers less than four. For a sum of length ℓ, where $2\lfloor \ell/2 \rfloor = j$, this requires

$$\binom{\ell}{1} + \binom{\ell}{3} + \binom{\ell}{5} + \cdots + \binom{\ell}{j} = 2^{\ell-1}$$

clauses, which is exponential. To remedy this, cut each sum, into subsums of length 4. For example, the equation $x_1 + x_2 + \cdots + x_\ell = 0$ is clearly equivalent to

$$\begin{aligned} x_1 + x_2 + x_3 + y_1 &= 0 \\ y_1 + x_6 + x_7 + y_2 &= 0 \\ &\vdots \\ y_i + x_{4i+2} + x_{4i+3} + y_{i+1} &= 0 \\ &\vdots \\ y_h + x_{\ell-2} + x_{\ell-1} + x_\ell &= 0 \end{aligned}$$

if ℓ is even. (If ℓ is odd, the final sum is length 3, this is more efficient because a sum or XOR of length 3 requires only 4 clauses. Therefore it is safe to be pessimistic and assume all equations are of even length). In either case, one can calculate $h = \lceil \ell/2 \rceil - 2$. Thus there will be $h + 1$ subsums, and each will require 8 clauses of length 4 each, via Equation 2.

The Cutting Number Here all sums were cut into subsums of length 4. One could have done this with lengths 3, 5 or higher. In particular, we performed experiments on 3, 4, 5 and 6 as the cutting number. It turns out that 6 is optimal. See Section 5.2 and Table 2.

3.2 Measures of Efficiency

Three common measures of the size of a CNF-SAT problem are the number of clauses, the total length of all the clauses, and the number of variables. It is not known which of these is a better model of the difficulty of a CNF expression. Initially we have n variables, and 0 clauses of total length 0.

For a quadratic system of polynomials, the cost for each monomial in Step One of the conversion is 1 dummy variable, 3 clauses, of total length 7. This needs to be done for all possible $M - n - 1$ quadratic monomials. The constant monomial requires 1 dummy variable, and 1 clause of length 1.

The cost in Step Two requires an estimate of the expected value of the length of each equation. Since there are M possible coefficients, then this is equal to $M\beta$. For the moment, assume the cutting number is 4. There will be (in expected value) $M\beta/2 - 1$ subsums per equation, requiring $M\beta/2 - 2$ dummy variables, $4M\beta - 8$ clauses and total length $16M\beta - 32$.

This is a total of

- Variables: $n + 1 + (M - n - 1)(1) + m(M\beta/2 - 1)$.
- Clauses: $0 + 1 + (M - n - 1)(3) + m(4M\beta - 8)$.
- Length: $0 + 1 + (M - n - 1)(7) + m(16M\beta - 32)$.

Substituting $m = cn$ and $M = n^2/2 + n/2 + 1$, one obtains

- Variables: $\sim n^2/2 + cn^3\beta/4$.
- Clauses: $\sim (3/2)n^2 + 2cn^3\beta$.
- Length: $\sim (7/2)n^2 + 8cn^3\beta$.

where $f(x) \sim g(x)$ if and only if $\lim_{x\to\infty} f(x)/g(x) = 1$. Furthermore, so long as β is $\omega(1/cn)$ then the first term of each of those expressions can be discarded. This would be the case in all but the most sparse systems. These expressions are summarized, for several values of cutting number, in Table 3.2

Cutting Number	Variables	Clauses	Length
Cut by 3	$\sim cn^3\beta/2$	$\sim 2cn^3\beta$	$\sim 6cn^3\beta$
Cut by 4	$\sim cn^3\beta/4$	$\sim 2cn^3\beta$	$\sim 8cn^3\beta$
Cut by 5	$\sim cn^3\beta/6$	$\sim (8/3)cn^3\beta$	$\sim (40/3)cn^3\beta$
Cut by 6	$\sim cn^3\beta/8$	$\sim 4cn^3\beta$	$\sim 24cn^3\beta$
Cut by 7	$\sim cn^3\beta/10$	$\sim (6.4)cn^3\beta$	$\sim 44.8cn^3\beta$
Cut by 8	$\sim cn^3\beta/12$	$\sim (32/3)cn^3\beta$	$\sim (128/3)cn^3\beta$

Table 1. CNF Expression Difficulty Measures for Quadratic Systems, by Cutting Number

3.3 Preprocessing

It is clear from the above expressions that n is the crucial variable in determining the number of dummy variables, clauses, and total lengths of clauses. With this in mind, we devised the following preprocessing scheme, based on the idea of Gaussian Elimination. It is executed before the conversion begins. For any polynomial one can consider the monomial as

$$x_{a_0} = x_{a_1} + x_{a_2} + \cdots + x_{a_n} + (\text{quadratic terms}) + (+1)$$

where the $+1$ term is optional, and $a_i \in \{1, \ldots, n\}$. This is, in a sense, a re-definition of x_{a_0}, and so we add this equation to every polynomial in the system where x_{a_0} appears. Afterword, x_{a_0} will appear nowhere in the system of equations, except in its definition, effectively eliminating it as a variable. Since SAT-solvers tend to choose the most-frequently-appearing variables when deciding which cases to branch on (except in a constant fraction of cases when they select randomly, e.g. 1% of the time), x_{a_0} will not be calculated until all other variables have been set.

If there are t equations of short length in the system, then after preprocessing these t variables only appear in their own definitions (not even the definitions of each other), and so far as the main system is concerned, there are now $n - t$ variables. In practice, the effect of this was slightly less than a doubling of performance, see Section 5.2.

We would only consider a polynomial for elimination if it were of length 4 or shorter (called "light massage") or length 10 or shorter (called "deep massage"). The reason for the length limit is to minimize the increase of β that occurs as follows. When performing Gaussian Elimination on an $m \times n$ sparse boolean matrix A, in the ith iteration, the β in the region $A_{i+1,i+1} \ldots A_{m,n}$ will tend to be larger (a higher fraction of ones) than that of $A_{i,i} \ldots A_{m,n}$ in the previous iteration [1, 11]. Even in "Structured Gaussian Elimination", when the lowest weight row is selected for pivoting at each step, this tends to occur. By adding two rows, the new row will have as many ones as the sum of the weights of the two original rows, minus any accidental cancellations. Therefore, by only utilizing low weight rows, one can reduce the increase in β. See the experiments in Section 5.2, and Table 2, for the effect.

3.4 Fixing Variables in Advance

Since cryptographic keys are generated uniformly at random, it makes sense to generate the x_i's as fair coins. But suppose g of these are directly revealed to the SAT solver by including the short equations $x_1 = 1, x_2 = 0, \ldots, x_g = 1$, and that a satisfying solution is found in time t_{SAT}. A real world adversary would not have these g values of course, and would have to guess them, requiring total time $2^{g-1}t_{SAT}$. As in algebraic cryptanalysis [5] it turns out that $g = 0$ is not the optimal solution. In our experiments, we tried all g within the neighborhood of values which produced t_{SAT} between 1 second and 1 hour, to locate the optimum.

Since exhaustive search requires checking 2^{n-1} possible values of $x_1, \ldots, x_n$ on average, then this method is faster than brute force if and only if the time required to check one potential key, t_{var} satisfies

$$t_{ver} > t_{SAT} 2^{-(n-g)}$$

This method is useful for the cryptanalysis of a specific system, e.g. DES [6]. In addition to having fewer variables, note that $m/n < m/(n-g)$, and so the "over-definition" or c will increase, yielding further benefit to fixing variables.

However, for random systems of quadratic equations, fixing variables g and substitution their values results in another system, which is an example of a random system with m equations and $n-g$ unknowns, but with slightly different sparsity. Therefore, we did not have to try different values of g in our final performance experiments, but chose $g = 0$.

Parallelization Suppose g bits are to be fixed, and 2^p processors (for some p) are available, with $p < g$. Then of the 2^g possible values of the g fixed bits, each processor could be assigned 2^{g-p} of them. After that, no communication between processors is required, nor can processors block each other. Therefore parallelization is very efficient. If interprocess communication is possible, then the "learned clauses" (explained in Section 4) can be propagated to all running SAT-solvers.

In the event that thousands of volunteers could be found, as in the DES challenge of 1997, or DESCHALL Project [10], then the low communications overhead would be very important.

4 SAT Solvers

SAT solvers have made a lot of progress in recent years, with both theoretical and practical improvements. In the major SAT competition [19], held annually, each year almost all the previous year's winners are beaten by new, more efficient solvers. Because of these competitions, SAT solvers are carefully designed to run on a large range of problems with no tuning required by users.

The huge success of SAT solvers means that both in research and industry many problems are solved by mapping them to CNF and solving them using these highly tuned SAT solvers. This may appear inefficient at first, as the mapping to CNF can lose much of the structure of the original problem. However the performance of SAT solvers is often able to offset this loss of structural information.

The basis of most SAT solvers is the Davis-Putnam backtrack search. This searches for a solution to a problem by recursively choosing a variable, first trying to assign it one value and then the other. At each stage of search a propagation step is performed, which attempts to imply the assignments to as many unassigned variables as possible based on the assignments made so far. This may also uncover a clause which cannot be satisfied, so search backtracks.

In most SAT solvers the only reasoning step used is to look for clauses where all literals are false except one, so the remaining one must be true. The authors of Chaff [24] found this simple reasoning step was found to take the majority of the time (>90%) in most SAT solvers, so designed an improved algorithm, called "watched literals", which allows a large number of large clauses to be efficiently propagated. The major strengths of this algorithm is that during search it only watched two variables in each clause, because as long as there are two unassigned variables in a clause it could not possibly perform any propagation. Further, due to clever data structure design it is not necessary to save and restore the state of the algorithm when search backtracks.

Another important feature of SAT solvers, which existed before Chaff, is *conflict resolution*. The solver will know exactly which clause failed, and for each variable in it what caused that

variable to be assigned. By constructing a graph of all the decisions which lead to the failure, it is possible to construct a minimal clause which "explains" the reason for failure. The hope is that this explanation will be a new clause not present in the original problem. In Chaff this new clause has three main uses.

1. *Conflict Related Backjumping*: At the point of failure, the new learned clause must be false by definition. It may be necessary to backtrack many levels in the search before the clause stops being false. This allows for many parts of the search to be skipped.
2. *Learned Clauses*: The new clause is added to the problem, and will therefore hopefully stop search earlier in other parts of the search.
3. *Conflict-Based Heuristics*: The heuristic which chooses which variable should be chosen next during search looks for the variable which has occurred in the most of the new conflict clauses. The hope is that this variable is important to search, and will cause search to fail faster.

During search, many new clauses will be learned, and simply storing all of them can become very expensive. Chaff keeps a list of which learned clauses have been most useful in terms of which have propagated most often and periodically throws away the ones which have been least productive.

The final feature of Chaff is that it periodically restarts the search, although keeping the information gained to guide the variable heuristics and the learnt clauses. This helps particularly on hard problems, as choosing the wrong variable for the first few branches appears to have a massive impact. This also provides the ability to learn new clauses over an entirely new set of variables. The number of search nodes between restarts is increased as search progresses, so that eventually search must complete.

The solver used in this paper is MiniSAT 2.0 [12], a minimalist open-source SAT solver which has won a series of awards including the three industrial categories in the SAT 2005 competition and first place in SAT-Race 2006. Mini-SAT is based on a similar basis to Chaff, but the algorithms involved have been optimized and carefully implemented. Also, Mini-SAT has carefully optimized variants of the variable order heuristics and learned clause removal heuristics.

Note About Randomness: The program MiniSAT is a randomized algorithm in the sense of occasionally using probabilistic reasoning. However, in order to guarantee reproducibility, the randomness is seeded from a hash of the input file. Therefore, running the same input file several times yields the same running time, to within 1%. Obviously, this "locked" randomness maybe a lucky choice, or an unlucky one. Since the actual performance of MiniSAT on these problems is log-normal (see Section 5.1), the consequences of an unlucky choice are drastic. Therefore, one should generate 20–50 CNF files of the same system, each perhaps different by fixing a different subset of g of the original n variables, or perhaps by reordering the clauses in a random shuffle.

The latter is very computationally cheap, but the former is better, as casual experimentation has shown there are definitely "lucky" and "unlucky" choices of variables to fix. More precisely, the running time is not dependent on g alone, but also the specific g out of n monomials chosen to be fixed. The expected value of the running time in practice can then be calculated as the mean of the running times of the 20–50 samples, each with a distinct random choice of fixed variables.

5 Experimental Results

In general, the running times are *highly* variable. We propose that the log-normal distribution, sometimes called Gibrat's distribution, is a reasonable model of the running time for a given system. This implies merely that the running time t is distributed as e^x, where x is some random variable

with the normal (Gaussian) distribution. In practice, however, this presents an experimental design challenge.

The distributions of the running times vary so wildly that at absolute minimum, 50 experiments must be performed to get an estimate of the expectation. Also, minor improvements, such as parameters of massaging, are only statistically significant after hundreds of repeated trials—which makes careful tuning of the massaging process impossible.

5.1 The Log-Normal Distribution of Running Times

Examine Figures 1 and 2, which plot the probability distribution of the running time, and its natural logarithm, respectively. One can observe that the second figure "looks normal", in the sense of being a bell curve that has had its right end truncated.

The kurtosis of a random variable is a measure of "how close to normal" it is, and takes values in $[-3, \infty)$. The normal distribution has a kurtosis of zero, and positive kurtosis implies a leptokurtic distribution, (one with values near the mean being more common than in the Gaussian), and negative kurtosis implies a platykurtic distribution. The plot of running times suggests an exponential of some kind, and so upon taking the natural logarithm of each point, a set of values with very low kurtosis (0.07) was found. The plot is close to a bell curve, and is from 443 data points, 14 of which were longer than the manually set 1800 sec time out, and 427 of which were plotted. Since $\log(1800) \approx 7.496$, this explains why the graph seems truncated at $\log t > 7.50$.

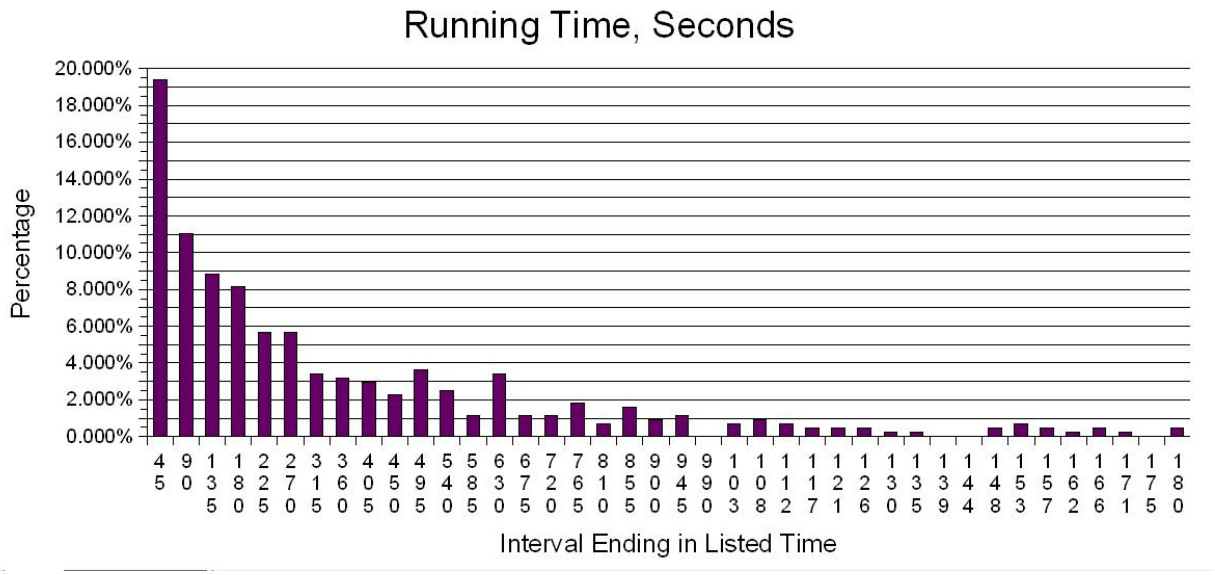

Fig. 1. The Distribution of Running Times, Experiment 1

These trials were of a system of equations with $n = 64, m = 640, c = 10, \beta = 1/100$, with $g = 15$ variables fixed in advance. The cutting number was 5, and light massaging was applied. The average running time of those that completed was 326.25 seconds, on one processor; since brute force would have to make an expectation of 2^{48} guesses to find the 49 bits not guessed, this is faster than brute force if and only if one guess can be verified in less than $t_{ver} = 0.001159$ nanoseconds, on one processor, which is absurd for modern technological means. Of course, newer and faster processors would improve the running time of both the SAT-solver and the brute force approach, though it is not possible to predict if these improvements will happen at the same rate.

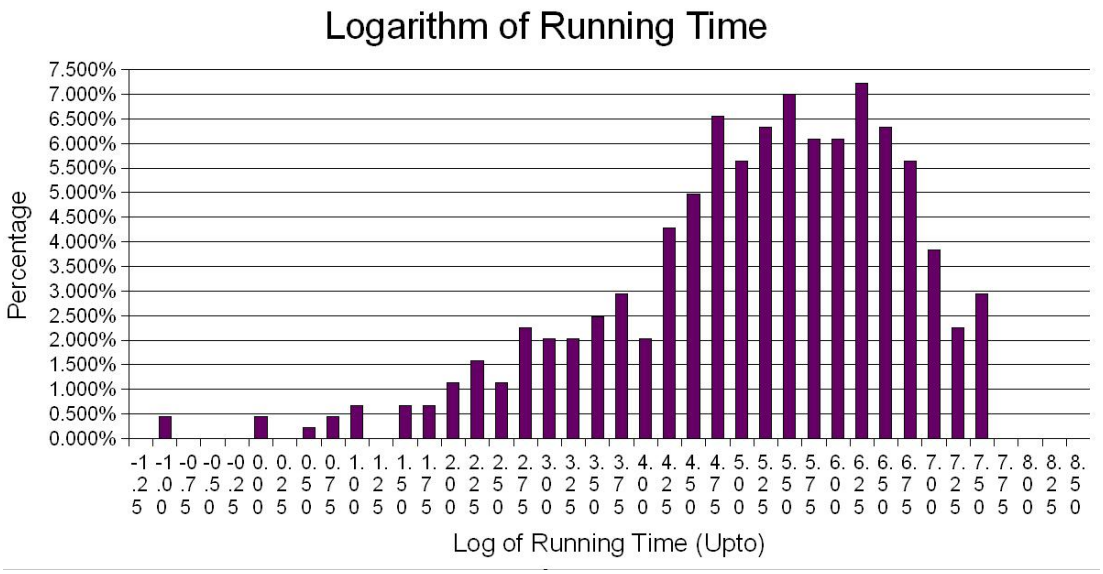

Fig. 2. The Distribution of the Logarithm of Running Times, Experiment 1

5.2 The Optimal Cutting Number

See Table 2. The system solved here is identical to that in the previous experiment, except different cutting numbers and massaging numbers were used during the conversion. Also, only 50 experiments were run. The result shows that deep massaging is a worthwhile step, as it cuts the running time by half and takes only a few seconds. Furthermore, it shows that cutting by six is optimal, at least for this system. Note, cutting by 8 would produce extremely large files (around 11 Mb)—those for cutting by 7 were already 5.66 Mb. Both in this case, and in casual experiments with other systems of equations, the running time does not depend too much on cutting number (also visible in Table 2), and that cutting by six remains efficient.

The kurtosis is seen to vary considerably in the table. Also, some of the modes have kurtosis near zero, which would imply a normal and not log-normal distribution. This is an artifact of having "only 50" experiments per mode. Among statisticians, a common rule is that a kurtosis of ± 1 is "reasonably close to Gaussian," which is the case in all but two of the systems in Table 2 for the logarithm of the running time.

The massage ratio is the quotient of the running time with massaging to that of the running time without. As one can see, the effects of a deep massage were slightly less than doubling the speed of the system. A light massage was even detrimental at times. This is because the requirement that a polynomial only be length 4 is quite severe (very few polynomials are that short). Therefore, there is only a small reduction in the number of variables, which might not be sufficient to offset the increase in β.

5.3 Comparison with MAGMA, Singular

See Table 3. The following experiments were performed using deep massaging, and cutting number equal to six, on a 2 GHz PC. By Singular, we refer to version 3.0.2 [28]. By MAGMA, we refer to version 2.11-2, and by MiniSAT, we refer to version 2.0 [12]. Various values of n, β and c were chosen to highlight the role of the number of variables, the sparsity, and the over-definition of the system.

	Cut by 3	Cut by 4	Cut by 5	Cut by 6	Cut by 7
No Massaging					
Naïve Average	393.94	279.71	179.66	253.15	340.66
Naïve StDev	433.13	287.33	182.18	283.09	361.04
Naïve Kurtosis	0.93	5.12	0.79	1.16	2.47
Average(ln)	5.11	4.96	4.55	4.72	5.2
StDev(ln)	1.63	1.46	1.35	1.51	1.27
Kurtosis(ln)	0.51	0.8	0.43	-0.5	-0.32
Light Massaging					
Naïve Average	413.74	181.86	269.59	217.54	259.73
Naïve StDev	439.71	160.23	301.48	295.88	237.52
Naïve Kurtosis	0.04	0.08	3.68	6.85	0.01
Massage Ratio	1.05	0.65	1.5	0.86	0.76
Average(ln)	5.3	4.64	4.84	4.52	4.87
StDev(ln)	1.39	1.29	1.5	1.47	1.5
Kurtosis(ln)	-0.38	0.07	0.09	-0.14	0.52
Deep Massaging					
Naïve Average	280.22	198.15	204.48	144.94	185.44
Naïve StDev	363.64	292.21	210.53	150.88	49.53
Naïve Kurtosis	5.67	9.24	3.74	0.62	4.69
Massage Ratio	0.71	0.71	1.14	0.57	0.54
Average(ln)	4.82	4.34	4.54	4.07	4.33
StDev(ln)	1.48	1.68	1.63	1.73	1.54
Kurtosis(ln)	1.1	2.41	0.75	-0.06	-0.23

Table 2. Running Time Statistics in Seconds

In particular, this method is much worse than brute force for dense systems, but far better than brute force for sparse systems (a $t_{ver} \approx 10^{-9}$ seconds would be the smallest value that could represent present capabilities). The transition appears somewhere near $\beta = 1/100$. The line marked $n = 49$ represents the experiments done in the previous part of this paper.

Finally, it is interesting to note that if Magma and Singular do not crash, then they out-perform our method. However, they do crash for many of the systems in this study, with an "out of memory" error. In practice, SAT-solvers do not require more memory than that required to hold the problem. This is not the case for Gröbner Bases algorithms.

n	m	c	β	Magma	Singular	ANFtoCNF & MiniSAT	t_{ver}
22	220	10	0.5	1.7 sec	1.0 sec	4021.99 sec	1.92×10^{-3} sec
30	150	5	0.1	3.5 sec	560 sec	$\approx 11,000$ sec	2.05×10^{-5} sec
52	520	10	0.01	277.890 sec	crashed	789.734 sec	3.51×10^{-13} sec
136	1360	10	10^{-3}	crashed	crashed	??	??
263	2630	10	10^{-4}	??	??	2846.95 sec	6.54×10^{-38} sec
22	25	1.1	0.5	65.5 sec	≈ 7200 sec	1451.62 sec	6.92×10^{-4} sec
30	33	1.1	0.1	crashed	crashed	15,021.4 sec	2.80×10^{-5} sec
52	58	1.1	0.01	??	??	??	??
133	157	1.1	10^{-3}	??	??	??	??
128	1280	10	10^{-3}	< 1 sec	crashed	0.25 sec	1.47×10^{-39} sec
250	2500	10	10^{-4}	??	91.5 sec	0.26 sec	1.44×10^{-76} sec
49	640	10.06	0.01	n/a	n/a	326.25 sec	1.159×10^{-12} sec

Table 3. Speeds of Comparison Trials between Magma, Singular and ANFtoCNF-MiniSAT

6 Previous Work

The exploration of SAT-solver enabled cryptanalysis is often said to have begun with Massacci and Marraro [21, 20, 22, 16], who attempted cryptanalysis of DES with the SAT-solvers Tableau, Sato, and Rel-SAT. This was successful to three rounds. However, this was a head-on approach, encoding cryptographic properties directly as CNF formulae. A more algebraic approach has recently been published by Courtois and Bard [6], which breaks six rounds (of sixteen). Fiorini, Martinelli and Massacci have also explored forging an RSA signature by encoding modular root finding as a SAT problem in [15].

The application of SAT-solvers to the cryptanalysis of hash-functions, or more correctly, collision search, began with [18] which showed how to convert hash-theoretic security objectives into logical formulae. The paper [23], by Mironov and Zhang, continued the exploration of hash functions via SAT-solvers by finding collisions in MD4 and MD5.

The authors believe this is the first successful application of SAT-solvers to solving systems of equations over finite fields. However, the approach was mentioned in [3], upon the suggestion of Jacques Stern.

7 Conclusions

The problem of solving a multivariate system of equations over $GF(2)$ is important to cryptography. We demonstrate that it is possible to efficiently covert such a problem into a CNF-SAT problem. We further demonstrate that solving such a CNF-SAT problem on a SAT-solver is faster than brute force for sparse cases. On most problems of even intermediate size, Gröbner Bases oriented methods, like Magma and Singular, crash due to a lack of sufficient memory. Our method, on the other hand, requires little more memory than that required to store the problem. In examples where Magma and Singular do not crash, these tools are faster than our methods. However, our method is still much faster than brute force approximately when $\beta \leq 1/100$.

An important consequence of these early experiments is that attention will have to be given to the sparsity of systems of equations that arise in cryptanalysis. The Data Encryption Standard [6] has already been attacked with this tool. The methods in this paper achieve attacks on 6 rounds, whereas more direct SAT-solver methods only achieved 3 rounds. As another example, the provably secure stream cipher QUAD is based on the hardness of the MQ problem, and uses the dense case [2]. However, if the sparse case were to be used (as a proposal to speed-up the stream cipher suggests), then it might be vulnerable to an attack via this method.

8 Acknowledgments

The idea of using SAT-solvers to solve the MQ problem has been mentioned, but, the application of that technique to this problem began as a discussion between Gregory Bard, Paul Bello, Kostas Arkoudas and Selmar Bringsjord in 2004. The work would have been impossible without ECRYPT, generously supporting joint work between Gregory Bard and Nicolas Courtois during the Summer of 2006. During that time, Professor Peter Jeavons of the Oxford Computing Laboratory introduced Chris Jefferson to the other two authors of this paper. At CRYPTO 2005, Ilya Mironov, working simultaneously on cryptographic applications of SAT solvers [23], but for hash functions, gave very useful advice. Part of this work was also supported by the National Science Foundation VIGRE program, in the form of a Dissertation Completion Fellowship. Blondine Debraize was very helpful by coding our equations in MAGMA. Finally, we thank Lawrence Washington and Patrick Studdard for commenting on early versions of this document.

References

[1] Gregory Bard. "Achieving a log(n) Speed Up for Boolean Matrix Operations and Calculating the Complexity of the Dense Linear Algebra step of Algebraic Stream Cipher Attacks and of Integer Factorization Methods." IACR E-print 2006/163.

[2] C. Berbain, H. Gilbert, and J. Patarin. "QUAD: A Practical Stream Cipher with Provable Security." *Advances in Cryptology: EUROCRYPT'05*. 2005.

[3] Nicolas Courtois. PhD Dissertation.

[4] Nicolas Courtois. "General Principles of Algebraic Attacks and New Design Criteria for Components of Symmetric Ciphers," *AES 4 Conference*, Bonn May 10-12 2004, LNCS 3373, pp. 67-83, Springer.

[5] Nicolas Courtois, Adi Shamir, Jacques Patarin, Alexander Klimov, "Efficient Algorithms for solving Overdefined Systems of Multivariate Polynomial Equations", *In Advances in Cryptology, Eurocrypt 2000*, LNCS 1807, Springer, pp. 392-407.

[6] Nicolas Courtois, and Gregory Bard. "Algebraic Cryptanalysis of the Data Encryption Standard." In preparation. See IACR E-print.

[7] Nicolas Courtois and Josef Pieprzyk. "Cryptanalysis of Block Ciphers with Over-defined Systems of Equations," *In Advances in Cryptology Asiacrypt 2002*, LNCS 2501, pp.267-287, Springer.

[8] Nicolas Courtois and Josef Pieprzyk. "Cryptanalysis of Block Ciphers with Over-defined Systems of Equations," Available at `http://eprint.iacr.org/2002/044/`.

[9] Nadia Creignou, and Hervé Daude. "Satisfiability Threshold for Random XOR-CNF Formulas." *Discrete Applied Mathematics*, 1999.
[10] M. Curtin. *Brute Force: Cracking the Data Encryption Standard*. Springer: 2005.
[11] Tim Davis. *Direct Methods for Sparse Linear Systems* Society for Industrial and Applied Mathematics. 2006.
[12] Niklas Een, Niklas Sorensson, " MiniSat - A SAT Solver with Conflict-Clause Minimization". *Proc. Theory and Applications of Satisfiability Testing (SAT'05)*. 2005.
[13] Jean-Charles Faugére: "A new efficient algorithm for computing Gröbner bases (F_4)", *Journal of Pure and Applied Algebra*. Vol. 139. (1999) pp. 61-88.
[14] Jean-Charles Faugére: "A new efficient algorithm for computing Gröbner bases without reduction to zero (F5)." *Workshop on Applications of Commutative Algebra*, Catania, Italy, 3-6 April 2002, ACM Press.
[15] C. Fiorini, E. Martinelli, and F. Massacci. "How to Fake an RSA Signature by Encoding Modular Root Finding as a SAT Problem." *Discrete Applied Mathematics*. Vol 130, Number 2: Pp 101–127. 2003.
[16] M. Hietalahti, F. Massacci, and I. Niemela. "DES: A Challenge Problem for Nonmonotonic Reasoning Systems." *Proc. 8th International Workshop on Non-Monotonic Reasoning (NMR'00)*. 2000.
[17] Thomas Jakobsen: "Cryptanalysis of Block Ciphers with Probabilistic Non-Linear Relations of Low Degree", *In Advances in Cryptology, Crypto 1998*, LNCS 1462, Springer, pp. 212-222, 1998.
[18] D. Jovanovic, and P. Janicic. "Logical Analysis of Hash Functions." *Frontiers of Combining Systems (FroCoS)*, Vol 3717 of *Lecture Notes in Artificial Intelligence, Springer*. Pp 200–215. 2005.
[19] Daniel Le Berre, Laurent Simon. "Special Volume on the SAT 2005 competitions and evaluations." *Journal of Satisfiability (JSAT)*. March 2006.
[20] L. Marraro, and F. Massacci. "Towards the Formal Verification of Ciphers: Logical Cryptanalysis of DES." *Proc. Third LICS Workshop on Formal Methods and Security Protocols, Federated Logic Conferences (FLOC-99)*. 1999.
[21] F. Massacci. "Using Walk-SAT and Rel-SAT for cryptographic key search." *Proc. 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*. 1999.
[22] F. Massacci, and L. Marraro. "Logical Cryptanalysis as a SAT-problem: Encoding and Analysis of the U.S. Data Encryption Standard." *Journal of Automated Reasoning*. Vol. 24, Numbers 1–2. 2000.
[23] I. Mironov, and L. Zhang. "Applications of SAT Solvers to Cryptanalysis of Hash Functions." *Proc. Theory and Applications of Satisfiability Testing (SAT'06)*. 2006. Also available as IACR E-print 2006/254.
[24] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, Sharad Malik. "Chaff: Engineering an Efficient SAT Solver". *Proc. of 28th Design Automation Conference (DAC' 01)*. 2001.
[25] Sean Murphy, Matt Robshaw: "Essential Algebraic Structure within the AES." *In Advances in Cryptology, Crypto 2002*, LNCS 2442, Springer.
[26] Claude Elwood Shannon: "Communication theory of secrecy systems," *Bell System Technical Journal* Vol. 28 (1949), see in particular page 704.
[27] MAGMA, High performance software for Algebra, Number Theory, and Geometry, — a large commercial software package: `http://magma.maths.usyd.edu.au/`
[28] Singular: A Free Computer Algebra System for polynomial computations. `http://www.singular.uni-kl.de/`

A Cubic Systems

While no experiments were performed on random cubic systems, the cryptanalysis of the first 6-rounds of the Data Encryption Standard, by Courtois and Bard [6], was carried out using the method in this paper. It was much faster than brute force, however, it was necessary to perform a great deal of human-powered preprocessing. See that paper for details.

In particular, the conversion for cubics will proceed identically to quadratics. The number of possible monomials is also much higher. This means our assumption that every monomial "is probably present" might not be true and the expected length of each equation is longer.

There are $\binom{n}{3} \sim n^3/6$ cubic monomials possible, each requiring 1 dummy variable, 4 clauses of total length 10. There are as before $\binom{n}{2} \sim n^2/2$ quadratic monomials possible, each requiring 1 dummy variable, 3 clauses of total length 7. The total number of monomials possible is thus

$$M = \binom{n}{3} + \binom{n}{2} + \binom{n}{1} + \binom{n}{0} \sim n^3/6$$

The expected length of any polynomial is $\beta M \sim \beta n^3/6$. Taking cutting by four as an example, this would require $\sim \beta n^3/12$ dummy variables, and $\sim (2/3)\beta n^3$ clauses of total length $\sim (8/3)\beta n^3$, for each of the m equations. Therefore, so long as β is $\omega(cn)$, then the cost of converting the monomials is negligible compared to that of representing the sums, as before.

Cutting Number	Variables	Clauses	Length
Cut by 3	$\sim cn^4\beta/6$	$\sim (2/3)cn^4\beta$	$\sim 2cn^4\beta$
Cut by 4	$\sim cn^4\beta/12$	$\sim (2/3)cn^4\beta$	$\sim (8/3)cn^4\beta$
Cut by 5	$\sim cn^4\beta/18$	$\sim (8/9)cn^4\beta$	$\sim (40/9)cn^4\beta$
Cut by 6	$\sim cn^4\beta/24$	$\sim (4/3)cn^4\beta$	$\sim 8cn^4\beta$
Cut by 7	$\sim cn^4\beta/30$	$\sim (32/15)cn^4\beta$	$\sim (224/15)cn^4\beta$
Cut by 8	$\sim cn^4\beta/36$	$\sim (32/9)cn^4\beta$	$\sim (256/9)cn^4\beta$

Table 4. CNF Expression Difficulty Measures for Cubic Systems, by Cutting Number

An interesting note is that as explained earlier, any polynomial system of equations in $GF(2)$ can be rewritten as a (larger) quadratic system. It is unclear if it is better to convert a cubic system via this method, and then construct a CNF-SAT problem, or construct the CNF-SAT problem directly from the cubic system of equations.