Tatra Mt. Math. Publ. **45** (2010), 107–136
DOI: 10.2478/v10127-010-0009-6

TATRA MOUNTAINS
Mathematical Publications

METHODS TO SOLVE ALGEBRAIC EQUATIONS IN CRYPTANALYSIS

Igor Semaev — Michal Mikuš

ABSTRACT. The goal of the present paper is a survey of methods to solve equation systems common in cryptanalysis. The methods depend on the equation representation and fall into three categories: Gröbner basis algorithms, SAT-solving methods and Agreeing-Gluing algorithms.

1. Introduction

Let (q, l, n, m) be a quadruple of natural numbers, where q is a prime power. Then F_q denotes a finite field with q elements and $X = \{x_1, x_2, \ldots, x_n\}$ is a set of variables from F_q. By X_i, $1 \leq i \leq m$ we denote subsets of X of size $l_i \leq l$. The system of equations

$$f_1(X_1) = 0, \ldots, f_m(X_m) = 0 \tag{1}$$

is considered, where f_i are polynomials over F_q and they only depend on variables X_i. Such equations are called l-sparse. A solution to (1) over F_q is an assignment in F_q to all n variables X that satisfies all equations (1). That is a vector of length n over F_q provided the variables X are somehow ordered. The main goal is to find all solutions over F_q.

A way how the equations are written often determines the Solving algorithm. Polynomial representation suggests using Gröbner basis family algorithms: they are surveyed in Section 2. Sparsity of the equations is a quite restrictive factor. Equation systems common in cryptanalysis which depend on large variable sets should be somehow represented in computer memory. So anyway they should be sparse in that or other sense. For instance, suitable sparsity is also low degree

2010 Mathematics Subject Classification: Primary: 68W30, 11T71, Secondary: 94A60, 13P15.
Keywords: equation systems, finite fields, Gröbner basis, k-SAT, agreeing, gluing, MRHS, cryptanalysis.
This material is based upon work supported under the grant NIL-I-004 from Iceland, Liechtenstein and Norway through the EEA Financial Mechanism and the Norwegian Financial Mechanism.

polynomials or polynomials that admit only bounded number of non-linear terms in polynomial representation. In present survey we focus on l-sparse equations over finite fields. This definition allows more freedom in operating equations for low l and generally results in a more efficient solution than with Gröbner basis algorithms. Besides polynomials there are two common ways to write (1): by CNF formulas and by solution lists to particular equations $f_i(X_i) = 0$. These methods are complimentary as clauses which make part of the CNF are produced from the solutions to $f_i(X_i) \neq 0$.

SAT-solving methods for operating CNF formulas are surveyed in Section 3. Representing equations by lists of their local solutions is central for this paper; see Section 4. We shall review specific to such representation solving techniques, which may be viewed as extensions to SAT-solving methods. In applications, at least in those of cryptanalysis, the problem is mostly to solve a set of random-looking polynomial equations, in fact they are very structured, rather than to satisfy a set of randomly chosen clauses. A Boolean polynomial in l variables has 2^{l-1} zeros (local solutions) on the average, while any clause with l literals admits $2^l - 1$ satisfying assignments. So there should be specific methods taking that distinction into account. For instance, Agreeing algorithm in Section 4.3 propagates $x_{i_1}, \ldots, x_{i_s} \neq a_1, \ldots, a_s$ from one equation to another by reducing the number of local solutions. Obviously, its performance is enhanced if the number of local solutions is low. The same is true for another main technique called Gluing in Section 4.5.

2. Gröbner bases

The notion of Gröbner bases was first introduced by B. B u c h b e r g e r in [4], who also proposed the basic algorithm for computing Gröbner basis from an arbitrary basis of an ideal. The Gröbner bases are general and quite powerful tool for solving systems of equations such as (1). The uniqueness property of reduced Gröbner basis of an ideal enables us to separate the computation of Gröbner basis and its application to any system of equations.

In the first part we introduce the notion and provide background definitions. We also shortly describe the Buchberger method of computing such Gröbner basis and state the (most important) uniqueness property.

In the second part we provide survey of existing algorithms for computing Gröbner basis. These algorithms have exponential theoretical-complexity bounds, but they achieve much better results in practical applications. The reason behind this fact is that the theoretical bound is derived from an average system of equations, but the system derived from concrete cryptosystem has

usually some structure, which can be exploited to reduce the complexity of the algorithm.

There are many improvements of Buchberger's algorithm, the most important is the F4 algorithm [18], which is a modification of an algorithm due to D. L a z a r d [27] and the F5 algorithm [19] by J. F a u g è r e.

2.1. Preliminaries

Here we provide the necessary notions *admissible ordering, reduction* and *Gröbner basis* and explain the idea of the basic Buchberger's algorithm for finding the reduced Gröbner basis for an arbitrary basis of an ideal in $F_q[X]$.

First, recall that a monomial in $X = \{x_1, \ldots, x_n\}$ is a term in X together with its corresponding coefficient. We denote set of all terms in X by $\mathcal{T}_X$. The $\mathcal{T}_X$ can be ordered in several ways, we mention only the two that are used most in the literature:

(1) total degree ordering (e.g., degree reverse-lexicographical ordering),

(2) lexicographical ordering.

For a polynomial $f(X)$ over F_q we denote by $LT(f)$ the leading term of f with respect to the used ordering, $LC(f)$ the leading coefficient and by $LM(f)$ we denote the leading monomial of f.

For any non-zero $p, v \in F_q[X]$ and some admissible ordering, we say that the polynomial p can be reduced mod v if there exists a monomial $\alpha \cdot t$ in p ($\alpha \in F \setminus 0$, $t \in \mathcal{T}_X$) that is divisible by $LT(v)$. Let $u = t/LT(v)$ and $p = \alpha \cdot t + r$, for some $r \in F_q[X]$, then:

$$p \mapsto_v p', \qquad \text{where} \quad p' = p - \frac{\alpha \cdot t}{LM(v)} \cdot v = p - \frac{\alpha}{LC(v)} \cdot uv.$$

If $p \mapsto_v p'$ for some $v \in V = \{v_1, \ldots, v_m\}$, then we say that p is reducible mod V and write $p \mapsto_V p'$. In the opposite case we say that p is irreducible mod V (or reduced mod V). By $\mapsto_V^*$ we denote the transitive closure of the relation $\mapsto_V$.

The reduction introduces an analogy of the classical division of two univariate polynomials with remainder. The most useful property of the reduction is that for any set V and an admissible ordering $<_T$, there does not exist an infinite sequence of polynomials such that

$$p \mapsto_V p_1 \mapsto_V \ldots$$

In other words the reduction always terminates. Furthermore, it is clear that if $p \mapsto_V 0$ then $p \in \langle V \rangle$. The reversed implication is not true in general. There exist V and $p \in \langle V \rangle$ such that $p \mapsto_V p' \neq 0$. The output of the reduction process depends on the sequence of chosen polynomials of V. It can be proved that the property $\forall p \in F_q[X] : p \in \langle V \rangle \Leftrightarrow p \mapsto_V^* 0$ is an equivalent to the definition of Gröbner basis.

In the following we describe the simple idea of the construction of a Gröbner basis from an arbitrary basis V. The basic Buchberger method is based on addition of a finite number of S-polynomials to the basis V. By the addition of a polynomial q that is already in $\langle V\rangle$ we are able to reduce some polynomials that were irreducible mod V while $\langle V\rangle$ remains the same. The S-polynomial of $v_1, v_2 \in F_q[X]$ is of the form:

$$S(v_1, v_2) = lcm\big(LM(v_1), LM(v_2)\big)\left(\frac{v_1}{LM(v_1)} - \frac{v_2}{LM(v_2)}\right).$$

The equivalence of following three facts was proved in [5]:

(i) G is a Gröbner basis,

(ii) $\forall\, v_1, v_2 \in G$: $S(v_1, v_2) \mapsto_G 0$,

(iii) if $p \mapsto_G p_1$ and $p \mapsto_G p_2$ then $p_1 = p_2$.

The condition (ii) enables us to test the Gröbner property of a basis easily. It implies that it is sufficient to test the reductions of $m(m-1)/2$ S-polynomials. If all the reductions are equal to 0 then G is a Gröbner basis. The third condition (iii) implies that the reduction via a Gröbner basis is a canonical function. In other words, there is one and only one $p' \in F_q[X]$ such that $p \mapsto_G p'$. We will denote such p' as $\mathrm{Red}(p, G)$.

The Buchberger method for finding a Gröbner basis to an arbitrary basis $V = \{v_1, \ldots, v_m\}$ is as follows:

(1) Select an S-polynomial $S(v_i, v_j)$ and let $r = \mathrm{Red}(S(v_i, v_j), Q)$.

(2) If $r = 0$, then return to step 1.

(3) If $r \neq 0$, then $V = V \cup \{r\}$. Return to step 1.

This process is repeated until every S-polynomial has been checked. Note that after adding r to basis V, new S-polynomials were added to the set that needs to be checked. For the formal algorithm we refer the reader to [4].

The resulting Gröbner basis is not unique and depends on the sequence of S-polynomials chosen as well as on the sequence of polynomials from V that were used for the reduction. The following paragraph shows how Gröbner basis can be simplified (reduced) to a unique set.

For any g_1, g_2 in a Gröbner basis, where

$$g_1 \neq g_2 \quad \text{and} \quad LT(g_2) \mid LT(g_1),$$

we can leave out polynomial g_1 because every reduction with g_1 can be done with g_2 instead. With this method we get a minimal Gröbner basis G (for every $g_1, g_2 \in G$: $LT(g_1) \not| LT(g_2)$ and $LC(g) = 1$). All minimal bases have the same cardinality and their members have the same set of leading polynomials. It can

be proved that $\langle G \rangle$ remains unchanged after reduction of a selected g_i by every g_j (with $j \neq i$). On this simple observation is based the following procedure:

$$\begin{aligned}
g_1 &\mapsto^*_{H_1} h_1, \quad \text{where } H_1 = \{g_2, \ldots, g_m\},\\
g_2 &\mapsto^*_{H_2} h_2, \quad \text{where } H_2 = \{h_1, g_3, \ldots, g_m\},\\
g_3 &\mapsto^*_{H_3} h_3, \quad \text{where } H_3 = \{h_1, h_2, g_4, \ldots, g_m\},\\
&\vdots\\
g_m &\mapsto^*_{H_m} h_m, \quad \text{where } H_m = \{h_1, \ldots, h_{m-1}\}.
\end{aligned}$$

The basis G, where any $g_i \in G$ cannot be reduced by g_j for any $j \neq i$ is called reduced Gröbner basis and the uniqueness of the reduced basis was proved by B u c h b e r g e r in [4].

2.2. Cryptanalysis with Gröbner bases

Finding solution of equation system (1) with the help of Gröbner bases is done in two steps. First being the construction of the reduced Gröbner basis and the second extracting the solution from the set of polynomials in the Gröbner basis.

2.2.1. Solving system of equations

Let $V = \{(z_1, \ldots, z_n) \in F_q \mid f_i(z_1, \ldots, z_n) = 0, \text{ for } i = 1, \ldots, m\}$ denote the set of all solutions of the system (1). It should be noted that in the cryptanalysis of some cryptosystem, the system of equations has usually one solution. The relationship between a Gröbner basis and the system of equations (1) with $q = 2$ (i.e., solutions over F_2) is stated in the following theorem [20]:

Theorem 1. *The reduced Gröbner basis of* $I = \langle f_1, \ldots, f_m, x_1^2 - x_1, \ldots, x_n^2 - x_n \rangle$, $I \subset F_2[X]$ *describes all solutions of* V. *Particular useful cases are:*

(i) $V = \emptyset$ *(no solution) if and only if* $G = [1]$.

(ii) V *has exactly one solution* $(a_1, \ldots, a_n)$ *if and only if* $G = \{x_1 - a_1, \ldots, x_n - a_n\}$, *where* $a_i \in F_2$.

2.3. Algorithms for computing the Gröbner basis

The basic algorithm leaves a lot of space to further improvements, e.g., the selection of a particular S-polynomial or the selection of a sequence of reductors q_i. The main "waste of time" during the algorithm is the reduction of S-polynomials—most of them are reduced to zero and are not added to the resulting basis. To avoid such reductions, the Buchberger criteria were introduced in [7] and improved in [19]. The criteria allow the detection of some polynomials that will be reduced to zero and therefore reduce the time-complexity [21].

The F4 algorithm

The F4 algorithm [18] keeps the selection step unchanged (e.g., with the Buchberger criteria) and improves the reduction phase with the help of sparse linear algebra methods. The main idea is to represent a set of polynomials by a matrix over F_q with columns representing terms and rows representing polynomials. The $m_{r,c}$ entry in the matrix is thus the coefficient of cth term in rth polynomial.

The polynomials in the matrix are members of the basis and all S-polynomials that are to be tested. The matrix is reduced to the row echelon form and resulting non-zero polynomials (reductions of S-polynomials) are added to the basis. This is repeated until no polynomials are added to the basis. The result is a Gröbner basis of the input ideal.

Another improvement lies in computation of successive "truncated" Gröbner bases, where the basis G_{d+1} is computed from G_d. By [18], we define a truncated Gröbner basis G_d to be the result of the Buchberger algorithm truncated to degree d (i.e., we discard all critical pairs of total degree greater than d). There exists a D_0 such that for all $d > D_0$, $G_d = G_{D_0}$.

An estimate of D_0 given by the Nullstellensatz can be used to analyze the time-complexity of the F4 algorithm. However, the real performance is much better in practice since D_0 is often overestimated. When published, the F4 algorithm was several times faster than any other previous implementation. The worst-case complexity, however, remains the same as the one of the basic Buchberger method.

The F5 algorithm

For both the basic Buchberger and the F4 algorithms, 90 % of computation time is spent on reductions to zero. The F5 algorithm [19] focuses on optimizing the Buchberger criteria, so that a significant part of these reductions is avoided. Under the assumption that the input system of polynomials forms a regular sequence (or semi-regular sequence [3]), there are no reductions to zero. Experiments show that this is the case for most systems in practice.

However, this algorithm has a drawback in the performance of the reduction step. Due to the signature compatibility conditions, many reductions are forbidden and even when the polynomials are top-reduced, their tails are left almost unreduced, which prolongs the reduction [24]. The F5 algorithm was the most efficient algorithm for computing Gröbner bases and has been successfully used to break several cryptosystems.

Although the F5 is not generally faster than the F4 algorithm, it is expected to excel for polynomial systems when the number of equations is less or equal to the number of variables. This was the case in the HFE system cryptanalysis or HFE (80 bits) Challenge [20].

Complexity of the F4, F5 algorithms

The running time of the Gröbner basis algorithm may be bounded by a value

proportional to $\binom{n+D}{D}^{\omega}$ ground field operations, where ω is the exponent in matrix multiplication complexity. The parameter D, called regularity degree, is only computed for semi-regular equations as they are defined in [3]. Theoretical complexity of the Gröbner basis algorithms as F4 or F5 on general polynomial equation systems remains unknown. It is also unknown whether an average equation system behaves semi-regularly, though this seems plausible [3].

The estimate simplifies to $\binom{n}{D}^{\omega}$ for any Boolean equations [3]. In case of sparse(l-sparse) Boolean equation systems each polynomial admits bounded number of monomials. Then each row in Macaulay matrices has bounded number of nonzero terms. Wiedemann algorithm [45] may likely be used to do the linear algebra step. Then one can put $\omega = 2$. The regularity degree for semi-regular Boolean equations for $m = n$ was estimated as $D \approx \alpha_d n$, where α_d depends on the equations maximal algebraic degree d. So that $\alpha_2 = 0.09, \alpha_3 = 0.15, \alpha_4 = 0.2$ and so on; see [2]. By estimating the binomial coefficient, the complexity is then $2^{2H(\alpha_d)n}$ up to a polynomial factor, where $H(\alpha)$ is the binary entropy function. One can see that only for quadratic semi-regular polynomials the running time is lower than 2^n, brute force complexity, and equal to 1.832^n bit operations. The best heuristic bound is then of order 1.724^n [43], where the method was combined with variable guessing.

A variant of the F4 algorithm

Recently, a variant of the F4 algorithm was proposed in [24]. The variant uses a (slightly modified) run of the F4 algorithm on a system of equations to reduce the complexity of the F4 algorithm on other similarly-looking systems. The first run is a kind of precomputation, where a list of all relevant polynomial multiples (coming from the critical pairs) is stored. This list is used for all other polynomial systems to determine which critical pairs are to be selected for the reduction phase.

However, the strategy proposed in this variant has a non-zero probability that the result will not be a Gröbner basis. As analyzed in [24], the probability of a correct result is greater than $(\frac{q-1}{q})^{\frac{q-1}{q}n}$ and results from practical experiments were presented that indicate a "quite good" probability for even small finite fields. This variant is better than the F5 algorithm only if the base field is large enough and several Gröbner bases have to be computed. Specifically the F5 remains the fastest for equation systems over F_2.

2.3.1. Comparison of F4, F5 with XL

Another general method for solving system (1) is the linearization technique. The terms of degree greater than one are substituted for new variables and the resulting linear (sparse) system is then solved. The restriction of the linearization technique is the condition that the number of linearly independent equations has

to be approximately the same as the number of different terms of the system. The extended linearization (XL) algorithm [12] deals with this problem. The XL algorithm generates new equations by multiplying the original equations by some prescribed monomial x^k. The resulting system is solved and terms containing one variable are eliminated last. The success of this method is based on the assumption that the elimination process yields at least one univariate equation. The equation is then solved and the original system can be simplified. The algorithm is repeated until the solutions for all variables are found.

Another aspect of the XL algorithm is studied in [1], where XL is compared to the F4 and F5 algorithms. In comparison to the F4, the XL also produces a Gröbner basis, but considers a greater number of polynomials than is the number of critical pairs in the F4 algorithm.

The experimental comparison between XL and the F5 (on semi-regular sequences) shows that the size of the matrix in XL is much larger than the matrix in the F5 for different settings ($q = 2$ and large q). That means that the F5 algorithm is faster and that the XL has a lower efficiency than was expected in [12].

3. Algorithms for solving k-SAT problems

3.1. Definitions

Let X be a set of n Boolean variables. Conjunctive normal form(CNF) is a Boolean algebra formula written as a conjunction of clauses:

$$F = \bigwedge_j \left(x^{(d_{j1})} \vee y^{(d_{j2})} \vee \ldots \vee z^{(d_{jk_j})} \right), \tag{2}$$

where one clause is a disjunction of variables X or their negations. Here $x^{(0)} = x$ and $x^{(1)} = \bar{x}$, they are called literals. Generally, the clause length may vary, but we only consider clauses of length $k_j \leq k$. In that case F is called k-CNF. Efficiently computing a $0,1$-assignment to variables X that makes (2) true is k-SAT problem.

Let $f(x_1, \ldots, x_l) = 0$ be any Boolean equation in k Boolean variables and $(a_{11}, \ldots, a_{1l}), \ldots, (a_{s1}, \ldots, a_{sl})$ be all binary vectors such that $f(a_{i1}, \ldots, a_{il}) = 1$. The vector $(b_1, \ldots, b_l)$ is a solution to the equation $f(x_1, \ldots, x_l) = 0$ if and only if it is a satisfying assignment for the conjunctive normal form

$$F_f = \left(x_1^{(a_{11})} \vee \ldots \vee x_l^{(a_{1l})} \right) \wedge \ldots \wedge \left(x_1^{(a_{s1})} \vee \ldots \vee x_l^{(a_{sl})} \right).$$

Given the system of Boolean ($q = 2$) equations (1), one constructs a CNF F which is a conjunction of F_{f_i}. Then $(b_1, \ldots, b_n)$ is a solution to (1) if and only if this vector is a satisfying assignment for F. In non Boolean case($q > 2$), each

variable from F_q is written as $\lceil \log_2 q \rceil$ Boolean variables. The number of Boolean variables in each equation is at most $k = \lceil \log_2 q \rceil l$. Therefore any (1) may be solved with a SAT-solver after writing that as a k-CNF.

3.2. Resolution

Introduced in [16]. If two formulas $x \vee P$ and $\bar{x} \vee Q$ are true, then $P \vee Q$ is true. This observation is used to simplify (2) by eliminating variables. Also one so proves that 2-SAT problem is polynomial time: if N is the number of satisfying assignments for a 2-CNF F, then they are computed in N polynomial time operations. Let x be any relevant variable, and

$$x \vee P_1, \ldots, x \vee P_r, \bar{x} \vee Q_1, \ldots, \bar{x} \vee Q_s$$

are all clauses, where x appears, and $R_1, \ldots, R_t$ are all other clauses in F. Let $s = 0$ first. Then any satisfying assignment to $R_1 \wedge \ldots \wedge R_t$ is extended to one or two satisfying assignments to F. That depends on whether or not there is at least one false clause P_i under that assignment. Then x should be assigned 1 otherwise both $0, 1$ work. Similarly, for $r = 0$. In case $r > 0, s > 0$, one recursively finds a satisfying assignment to $R_1 \wedge \ldots \wedge R_t \wedge \ldots \wedge (P_i \vee Q_j) \wedge \ldots$ The latter does not depend on x. If P_i are all true and at least one Q_j is false under that assignment, then x is assigned 0. If Q_j are all true and at least one P_i is false, then x is assigned 1. Otherwise, when P_i and Q_j are all true, then x is assigned both $0, 1$. As the clause length is always at most 2, we get that N satisfying assignments are constructed in N of polynomial operations.

3.3. DPLL algorithm

Introduced in [16], [17]. This is a guess-and-determine algorithm. One chooses a non assigned variable x, make a decision on which value 0 or 1 it should be assigned. That is expanded over clauses which contain x or $\bar{x}$ by modifying them: remove the clause or remove the literal x or $\bar{x}$ from that clause depending whether the literal is 1 or 0. Force values of variables in one-literal clauses (unit-clause rule) and repeat the above step. If no values of new variables are found, then assign a new variable. Assume a contradiction (the same variable was assigned different values or, equivalently, all literals in a clause are assigned 0) is observed. Then assign x another value or backtrack to previously assigned variable. All solutions (assignments satisfying all clauses) are produced with that algorithm.

A probabilistic version of the DPLL algorithm, enhanced with bounded resolution, was evaluated in [30]. Let

$$\mu_k = \sum_{j=1}^{\infty} \frac{1}{j(j + \frac{1}{k-1})}.$$

It is shown that in case of uniquely satisfiable k-CNF the algorithm running time is bounded by $2^{(1-\frac{\mu_k}{k-1})n+o(n)}$ for each $k \geq 3$ and the error probability $o(1)$ as n tends to ∞. The same was shown true for general k-CNF in case $k \geq 5$.

3.4. Practical improvements

They are mostly in choosing properly variables to assign, accelerating expansion of assigned values with watched literals, and reducing the search space by analyzing conflict clauses; see [28], [47]. The last direction is here presented in some detail. In DPLL algorithm current partial assignment is written as a list of expressions $x = a(n)$, where x is a variable, a its value and n the tree level, where the variable was assigned or forced by satisfying one-literal clauses. After a conflict was observed, one constructs a conflict clause containing information on the assignment which imply the conflict. Conflict clauses are combined with resolution and used to reduce search space. Find below an example from [28] after slightly changing the notation. Assume the clause database consists of

$$\begin{array}{lll} w_1 = \bar{x}_1 \vee x_2, & w_2 = \bar{x}_1 \vee x_3 \vee x_9, & w_3 = \bar{x}_2 \vee \bar{x}_3 \vee x_4, \\ w_4 = \bar{x}_4 \vee x_5 \vee x_{10}, & w_5 = \bar{x}_4 \vee x_6 \vee x_{11}, & w_6 = \bar{x}_5 \vee \bar{x}_6, \\ w_7 = x_1 \vee x_7 \vee x_{12}, & w_8 = x_1 \vee x_8, & w_9 = \bar{x}_7 \vee \bar{x}_8 \vee \bar{x}_{13}, \\ \ldots & & \end{array}$$

Let the current partial assignment be

$$x_9 = 0(1),\ x_{10} = 0(3),\ x_{11} = 0(3),\ x_{12} = 1(2),\ x_{13} = 1(2), \ldots$$

One now assigns $x_1 = 1$ at level 6. Then the algorithm follows the first implication graph in Figure 1, where the edges are labeled with indexes of clauses which produce the implications. For instance, w_1 implies $x_2 = 1(6)$ after the assignment $x_1 = 1(6)$. After forcing $x_5 = 1$, the clause w_6 implies $x_6 = 0$. That is in conflict with forcing $x_6 = 1$ from w_5. One now finds which assignments led to that conflict by walking over the implication graph backwards. That is $x_1 = 1, x_9 = 0, x_{10} = 0, x_{11} = 0$. Therefore, conflict clause $\bar{x}_1 \vee x_9 \vee x_{10} \vee x_{11}$ is added to the clause database. That is to be satisfied as any other database clause. One now assigns $x_1 = 0$ at level 6 and the algorithm follows the second implication graph in Figure 1 and gets a conflict. That produces a new conflict clause $x_1 \vee \bar{x}_{12} \vee \bar{x}_{13}$. New clauses provide with two advantages. First, they increases the probability of producing one-literal clauses and therefore the probability of forcing variable values. Secondly, clauses combined by resolution may result in reducing the search space. For instance, the resolvent of the above conflict clauses is

$$x_9 \vee x_{10} \vee x_{11} \vee \bar{x}_{12} \vee \bar{x}_{13}.$$

All these variables were assigned on the current branch, where the maximal level of assignment was 3. Therefore, after the assignment $x_9 = 0(1)$, $x_{10} = 0(3)$, $x_{12} = 1(2)$, $x_{13} = 1(2)$ the assignment $x_{11} = 0(3)$ was wrong and one

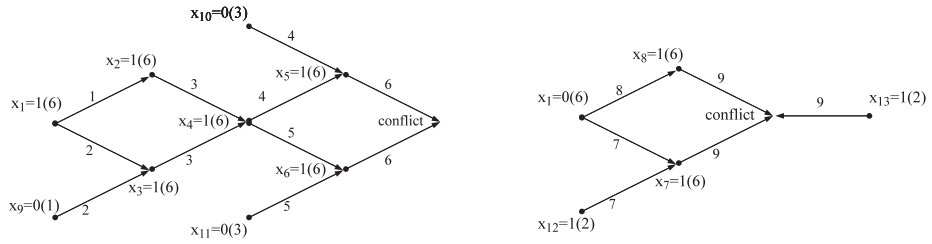

FIGURE 1. Implication graphs.

takes the assignment $x_{11} = 1(3)$ or gets a conflict if the assignment $x_{11} = 0(3)$ was forced. We conclude that after these two conflicts at level 6 one does not backtrack to level 5(chronological backtracking), but directly to level 3(non-chronological backtracking).

3.5. Local search

Introduced in [29], [37]. Given a CNF formula, guess an initial assignment to all variables. Repeat $3n$ times: if the formula is satisfied, then terminate; let R be some clause not being satisfied by the current assignment, then pick one of its literals uniformly at random and flip its value in the current assignment. It is shown in [37] that the probability of finding a satisfying assignment is at least $\frac{2}{3}\left(\frac{k}{2(k-1)}\right)^n$, where each clause has at most k literals. Therefore the expected number of this procedure repetitions before the satisfying assignment is found is proportional to $\left(2-\frac{2}{k}\right)^n$. A deterministic version of local search is presented in [15], its running time is bounded by $\left(2-\frac{2}{k+1}\right)^n$ operations.

3.6. Worst case bounds for k-SAT

A survey of the current worst case bounds is found in [23]. Some of them are shown in the first line in Table 1.

3.7. SAT-solvers competition

See the web-page http://www.satcompetition.org/ for the best current software SAT-solvers.

3.8. Algebraic cryptanalysis of DES with SAT-solvers

Any equations with discrete variables may be written as k-CNF formulas by introducing new variables. If, for instance, polynomials are sparse combinations of low degree monomials, then the number of new variables should be relatively low. In that case an efficient SAT-solver may be faster in solving equations then polynomial based Gröbner bases algorithms, implemented

in MAGMA; [13], [14]. However, in many cases when MAGMA did not crash due to out of memory, it over-performs the method of [13]. The reason for that is probably SAT-solvers are not efficient in solving linear algebra problems.

Typical equation in DES is $S(X+K) = Y + Z$, where S is a 6×4 nonlinear S-box, X is a 6-bit part of a 32-bit state, which is 32-bits right most part of the DES 64-bit states, K is 6-bit part of the 48-bit round key, Y and Z are 4-bit parts of the previous and subsequent 32-bit states.

To write a suitable CNF representing one uses the fact: Let $S(x_1, \ldots, x_n) = (y_1, \ldots, y_m)$, where $n = 6, m = 4$ for each DES S-box. For any subset T of monomials in $x_1, \ldots, x_n, y_1, \ldots, y_m$ there are at least $|T| - 2^n$ linearly independent equations $f_i(x_1, \ldots, x_n, y_1, \ldots, y_m) = 0$ involving only monomials in T; [14]. With $f_i = 0$ the equations are written in CNF for each S-box and therefore for each equation $S(X + K) = Y + Z$. However the best result is produced with using low-gate count representation of DES in [26]. Overall, the authors estimate 6-round DES may be broken in time equivalent to 2^{48} DES encryptions, instead of average 2^{55} by brute force.

4. Agreeing-Gluing methods

Methods studied in this section are mostly guess-and-determine algorithms. In sparse equations the number of guesses on a big enough variable set Y and the time to produce them is much lower than $q^{|Y|}$ due to the Search algorithm which may be treated as an extension to the DPLL algorithm. Let Y be an ordered string of variables and a be an F_q-vector of the same length. We say that a is a vector in variables Y, or Y-vector, if the entries of a may be assigned to the variables Y, for instance, in case of fixation.

4.1. Search algorithm

Given $Y \subseteq X$, the Search algorithm finds all Y-vectors over F_q that do not contradict equations (1), e.g., by running Agreeing algorithm in Section 4.3 or 4.4. In case $Y = X$ they are all solutions to (1). The Search algorithm follows a tree defined by any subset sequence $Y_1 \subseteq Y_2 \subseteq \ldots \subseteq Y_s = Y$. The root is labeled by $\emptyset$, the vertices at level $1 \leq k \leq s$ are labeled by partial assignments $W(k)$ produced by the Agreeing algorithm after the variables Y_k were assigned. The Agreeing algorithm has two outcomes. First, an inconsistence is observed. Then the algorithm backtracks to the previous assignment. Second, no inconsistence is observed and the values of some variables beyond Y_k might be learned. Then new variables get assigned and so on.

Vertices a and b at subsequent levels are connected if a is a sub-vector of b. A sequence of subsets that minimizes the running time may be taken. However,

in practice, it is not necessary to prescribe from the beginning which variables to assign. Asymptotical estimates in [42] suggest $Y = Z_r$, the set of variables that appear in at least r of equations (1) for appropriate r. The subset sequence is then $Z_r(r) \subseteq Z_r(r+1) \subseteq \ldots \subseteq Z_r(m) = Z_r$, where $Z_r(k)$ denote the set of variables that appear in at least r of first k equations.

4.2. Equation representation

We look for the set of all solutions to (1) over F_q, so polynomials f_i of degree at most $q-1$ in each variable are only considered. Obviously, the equation $f_i(X_i) = 0$ is determined by the pair (X_i, V_i), where V_i is the set of X_i-vectors, where f_i is zero. Such representation was first introduced in [46] and independently in [31].

EXAMPLE. Let $1 + x_1 + x_3 + x_1x_3 + x_1x_2x_4 + x_1x_3x_4 + x_2x_3x_4 = 0$. The solution set is the columns of the right hand side matrix in

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}. \tag{3}$$

In case the equation is the part of a system, then right hand side columns are called local solutions.

4.3. Agreeing procedure and Agreeing1 algorithm

Agreeing procedure was called local reduction in [46] and a variation of that was called graph algorithm in [31]. Two equations

$$E_i = (X_i, V_i) \quad \textit{and} \quad E_j = (X_j, V_j)$$

are transformed to (X_i, V_i') and (X_j, V_j'), where each V_i' consists of those vectors in V_i whose projection to $X_i \cap X_j$ occur in the projections of V_j. If $X_i \cap X_j = \emptyset$, then the procedure does not apply. It is shown in [41] that some pairs E_i, E_j can be avoided too even if $X_i \cap X_j \neq \emptyset$. This reduces memory requirement of the Agreeing algorithm. The equations $E_1, \ldots, E_m$ are vertices in an equation graph G. Vertices E_i and E_j are connected by the edge (E_i, E_j) labeled with $X_{i,j} = X_i \cap X_j \neq \emptyset$. The Agreeing procedure, being applied to E_i and E_j, implements a kind of information exchange between them through the edge (E_i, E_j). That is for $Y \subseteq X_{i,j}$ the information $Y \neq a$ is transmitted from E_i to E_j or backwards. Some edges are obsolete in this respect and may be removed, remaining edges are called maximal. The subgraph with minimal number of maximal edges is called minimal and denoted G_m. It is not uniquely defined. The following algorithm constructs G_m.

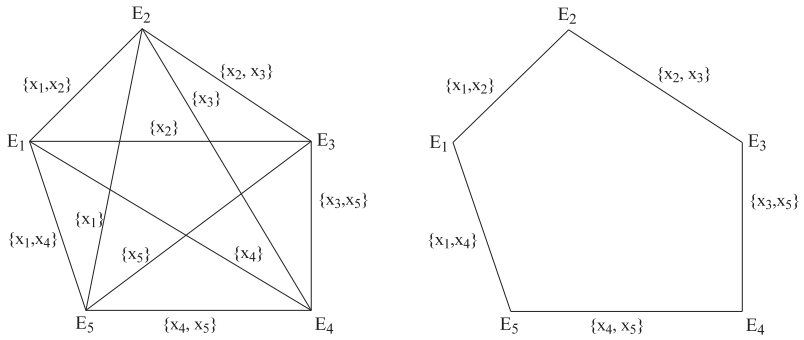

FIGURE 2. Edges removing.

(1) For every label $Y \subseteq X$ find all edges (E_s, E_r) in G such that $Y \subseteq X_{s,r}$. Denote a subgraph of G with all such edges (E_s, E_r) and involved vertices by G_Y. We remark that G_Y is a complete graph.

(2) Find the set V_Y of edges (E_s, E_r) in G_Y, where $X_{s,r} = Y$. Find a subset $W_Y \subseteq V_Y$ such that G_Y is still connected after removing the edges W_Y and W_Y has the largest number of edges. We remark that W_Y is not uniquely defined, and taking different W_Y produces different minimal subgraphs.

(3) Remove the edges W_Y from G for all Y and get G_m.

EXAMPLE. Let $X_1 = \{x_1, x_2, x_4\}$, $X_2 = \{x_1, x_2, x_3\}$, $X_3 = \{x_2, x_3, x_5\}$, $X_4 = \{x_3, x_4, x_5\}$ and $X_5 = \{x_1, x_4, x_5\}$. The equation graph G has 5 vertices and 10 edges: (E_1, E_2) labeled with $X_{1,2} = \{x_1, x_2\}$, (E_2, E_3) labeled with $X_{2,3} = \{x_2, x_3\}$, and so on. Five edges $(E_1, E_3), (E_1, E_4), (E_2, E_4), (E_2, E_5), (E_3, E_5)$ may be removed as they are obsolete for the Agreeing algorithm; see Figure 2.

In Agreeing1 method the procedure is pairwise applied to reduce the size of V_i in the whole system. There is only a very small chance to get the size of each V_i so low, e.g., 1, and to solve the equations.

4.4. Agreeing2 algorithm

This is an asymtotically faster variant of the Agreeing algorithm, see [33].

Precomputation. For each edge(maximal edge) (E_i, E_j), where $X_{i,j} \neq \emptyset$, let $r = |X_{i,j}|$. For each r-string address b an unordered tuple of lists

$$\{V_{i,j}(b); V_{j,i}(b)\} \tag{4}$$

is precomputed. The lists $V_{i,j}(b)$ and $V_{j,i}(b)$ consist of vectors from V_i and respectively V_j whose projection to variables $X_{i,j}$ is b. The list of tuples is sorted

using some linear order. The algorithm marks vectors, which are wrong solutions, in tuples (4). We say list $V_{i,j}(b)$ is empty if it does not contain any entries or all its entries get marked.

Agreeing. The algorithm starts with the first tuple $\{V_{i,j}(b); V_{j,i}(b)\}$, where just one of two lists is empty. If no such tuples are found, then the equations are pairwise agreed, then a guess is generally necessary to start marking.

(1) Let the current tuple be $\{V_{i,j}(b); V_{j,i}(b)\}$, where $V_{i,j}(b)$ is empty, while $V_{j,i}(b)$ is not.

(2) For every unmarked a in $V_{j,i}(b)$ do: mark a in $V_{j,i}(b)$, for every edge (maximal edge or where $X_{j,k} \neq \emptyset$) (E_j, E_k) do: compute the projection d of a to variables $X_{j,k}$, mark a in $V_{j,k}(d)$, the tuple $\{V_{j,k}(d); V_{k,j}(d)\}$ is now current.

(3) If just one of $V_{j,k}(d)$ or $V_{k,j}(d)$ is found empty, then apply step 1. If not, then backtrack to the tuple $\{V_{i,j}(b); V_{j,i}(b)\}$. Take another edge (E_j, E_k) or another unmarked a in $V_{j,i}(b)$. If $V_{j,i}(b)$ is already empty, then backtrack to the tuple last to $\{V_{i,j}(b); V_{j,i}(b)\}$. If the former was the starting tuple, then start a new walk with the next tuple, where just one of the lists is empty or terminate walking.

(4) All vectors that have been earlier marked in the tuples are now deleted from V_i.

We remark that each tuple $\{a_1, \ldots, a_r; b_1, \ldots, b_s\}$ implements two implications. First, marking(with a bar) all $\{a_1, \ldots, a_r\}$ implies marking all $\{b_1, \ldots, b_s\}$, which is denoted $\bar{a}_1, \ldots, \bar{a}_r \Rightarrow \bar{b}_1, \ldots, \bar{b}_s$, and vice versa $\bar{b}_1, \ldots, \bar{b}_s \Rightarrow \bar{a}_1, \ldots, \bar{a}_r$. Agreeing2 algorithm simply expands marking through these implications. Equations (1) are pairwise agreed if and only if in all $\{V_{i,j}(b); V_{j,i}(b)\}$ the lists both are empty or both non-empty. If for at least one edge (E_i, E_j) the lists $V_{i,j}(b)$ are empty for all b, then the system is inconsistent.

EXAMPLE. Let three Boolean equations E_1, E_2, E_3 be given in algebraic normal form:

$$\begin{aligned} 1 + x_3 + x_1x_2 + x_1x_3 + x_1x_2x_3 &= 0, \\ 1 + x_1 + x_4 &= 0, \\ 1 + x_3 + x_2x_4 + x_3x_4 + x_2x_3x_4 &= 0. \end{aligned}$$

Represent them as lists of solutions:

	a_1	a_2	a_3
x_1	0	0	1
x_2	0	1	1
x_3	1	1	0

,

	b_1	b_2
x_1	0	1
x_4	1	0

,

	c_1	c_2	c_3
x_2	0	1	1
x_3	1	0	1
x_4	0	1	0

. (5)

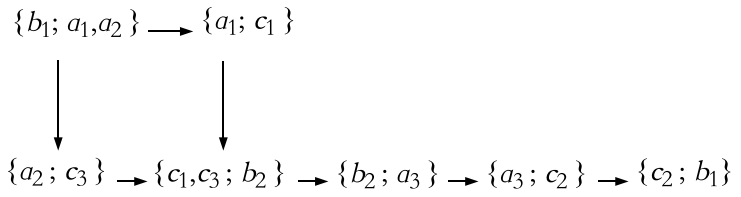

FIGURE 3. The marking expansion.

The list of tuples:

$$P = \{a_1, a_2; b_1\}, \quad Q = \{a_3; b_2\}, \quad R = \{b_1; c_2\}, \quad T = \{b_2; c_1, c_3\},$$
$$U = \{a_1; c_1\}, \qquad V = \{a_2; c_3\}, \quad W = \{a_3; c_2\}.$$

As there are no tuples with just one list empty, a guess is necessary to start marking. Assume $x_4 = 0$. So b_1 should be marked. We now have two tuples, where just one of the lists is empty: $\{\bar{b}_1; a_1, a_2\}$ and $\{\bar{b}_1; c_2\}$. According to the algorithm, take the first of two. Then a_1 get marked in $\{\bar{b}_1; a_1, a_2\}$ and $\{a_1; c_1\}$. Therefore, c_1 get marked in $\{\bar{a}_1; c_1\}$ and then in $\{c_1, c_3; b_2\}$. Now backtrack and mark a_2 in $\{\bar{b}_1; \bar{a}_1, a_2\}$ and $\{a_2; c_3\}$, and so on. The sequence of marking is represented in Fig. 3. Instances in all tuples have been marked. The guess was wrong. We alternatively could add a new tuple $\{b_1; \emptyset\}$ to the tuple list and start marking. Similarly, all tuple lists become empty in case $x_4 = 1$. The system has no solution.

4.5. Gluing

In Gluing procedure the solutions of two equations (X_1, V_1) and (X_2, V_2) are combined. That produces $(X_1 \cup X_2, V)$, where $V = V_1 \circ V_2$ are all common solutions to the equations in common variables $X_1 \cup X_2$. In Boolean case Gluing is viewed an extension to resolution in Section 3.2. Let $x \vee P$ and $\bar{x} \vee Q$ be formulas, where P and Q do not depend on x. Let U_1, U_2 be satisfying assignments to $x \vee P$ and $\bar{x} \vee Q$ in relevant variables. Then $U = U_1 \circ U_2$ are all common satisfying assignments to both the formulas. The projection of U to all variables besides x are all satisfying assignments to $P \vee Q$.

Gluing procedure alone may be used to solve the system (1). Gluing1 method [38] is based on computing solutions U_k to the equation subsystems:

$$f_1(X_1) = 0, \ldots, f_k(X_k) = 0 \qquad \text{for} \quad k = 1, \ldots, m.$$

One extends instances U_k to instances $U_{k+1} = U_k \circ V_{k+1}$ by walking throughout a search tree. In the end, all system solutions are U_m. The running time is determined by the maximal of $|U_k|$, say $|U_{k_0}|$.

Gluing2 is a time-memory trade-off variation [38]. Let $U_{k,t}$ denote the solutions to t equations

$$f_{k+1}(X_{k+1}) = 0, \ldots, f_{k+t}(X_{k+t}) = 0$$

in variables $X_{k+1} \cup \ldots \cup X_{k+t}$. One separately computes U_k, and $U_{k,t}$ and then $U_{k+t} = U_k \circ U_{k,t}$. At least one of $U_k, U_{k,t}$ should be kept in memory to efficiently compute U_{k+t}. Let $k, t < k_0 < k + t$. Then, on the average, the algorithm running time is determined by the maximal of $|U_k|, |U_{k,t}|$ and $|U_{k+t}|$. One may compute k_1 such that $|U_{k_1}|, |U_{k_1,k_1}|$ and $|U_{2k_1}|$ are equal on the average; see Section 5. The algorithm running time is then determined by $|U_{k_1}|$.

4.6. Improved Agreeing-Gluing algorithm

Let $r \geq 1$ and Z_r denote variables that occur in at least r equations. We take the largest r, where Z_r is not empty. Then Z_r-vectors that do not contradict to any of equations (1) are generated by the Search algorithm. We denote them W_r. For each $a \in W_r$ the variables Z_r are substituted by the entries of a. New l-sparse equations in a smaller variable set $X \setminus Z_r$ result. The above step is recursively applied to compute W_{r-1} and so on. It is enough to obtain W_2. The Agreeing-Gluing algorithm [40] is a particular case of the method for $r = 1$.

4.7. Agreeing-gluing versus SAT-solvers

A variant of the Search algorithm was described in [35]. It adapted watching and dynamic learning by conflict analysis, which are important features of modern SAT-solvers; see Section 3.4. The experiments showed that for randomly generated instances of n 5-sparse Boolean equations in $n \leq 170$ variables, the number of guesses while walking over a search tree is significantly lower than with MiniSat.

4.8. Syllogism rule algorithm

Introduced in [46]. Assume a system of Boolean equations. Let x and y be variables which appear in one particular equation $f_i(X_i) = 0$. One may compute all projections to x, y of its local solutions. If the projection set consists of 4 different 2-bit strings, then nothing to do. If there is one projection, then the variables x, y are determined and other system equations, where x or y are relevant, are modified accordingly. In case of two projections, one learns $x = y^{(a)}$ for a constant a or one of the variables is determined. If there are three projections, then the equation $x^{(b)} \vee y^{(c)} = 1$ is deduced, where b, c is a missing projection. A database of such implications is collected. They are combined by resolution, as $x^{(b)} \vee \bar{u} = 1$ and $z^{(d)} \vee u = 1$ imply $x^{(b)} \vee z^{(d)} = 1$; see Section 3.2, or by substitution with $x = y^{(a)}$. New implications may reduce the number of local solutions in other system equations. That in turn may generate new 2-variable implications and so on. The method is combined with initial variable guessing otherwise it is unlikely to have any such implication. Experiments in [46] demonstrated that Syllogism rule algorithm may overcome a combination of guessing with local reduction (called Agreeing1 algorithm here).

5. Average complexity bounds

5.1. Probabilistic model

For numbers q, n, m, and $l_1, \ldots, l_m \leq l$ uniform distribution on instances (1) is assumed. As any particular information on equations is beforehand assumed unknown, this looks the most fair probabilistic model to compute expected complexities. The uniformity means:

(1) the equations in (1) are independently generated. Each equation $f_i(X_i) = 0$ is determined by

(2) the subset X_i of size l_i taken uniformly at random from the set of all possible l_i-subsets of X, that is with the probability $\binom{n}{l_i}^{-1}$,

(3) and the polynomial f_i taken uniformly at random and independently of X_i from the set of all polynomials of degree $\leq q-1$ in each of variables X_i. In other words, with the equal probability $q^{-q^{l_i}}$.

Running time of any deterministic solving algorithm is a random variable under this probabilistic model. We assume that m/n tends to $d \geq 1$, while q and l are fixed.

5.2. Gluing algorithms estimates

These are explicit estimates. For a positive real number α let

$$f(z) = \ln(e^z + q^{-1} - 1) - \alpha \ln(z).$$

By z_α we denote the only positive root of the equation $\frac{\partial f}{\partial z}(z) = 0$. Also let

$$g(\alpha) = f(z_\alpha) - \alpha + \alpha \ln \alpha - \frac{\alpha \ln q}{l}.$$

It was proved in [38] that the expectation of $|U_k|$, the number of solutions to the first k equations in (1), is bounded by $(qe^{g(\alpha)} + \epsilon)^n$, where $\alpha = kl/n$ and ϵ is any positive real number as n tends to infinity. By computing the maximum of $g(\alpha)$ one proves that the Gluing algorithm expected complexity is bounded by $(q^{1-\gamma_{q,l}} + \epsilon)^n + m$ bit operations, where

$$\gamma_{q,l} = \frac{1}{l} + \left(q^{\frac{1}{l}} - 1\right) \log_q \left(\frac{1 - q^{-1}}{1 - q^{-\frac{1}{l}}}\right).$$

One may compute α_1, a positive root to $g(\alpha) = g(2\alpha)$ and put $k_1 = \lceil \alpha_1 n/l \rfloor$. The Gluing2 algorithm expected complexity is then at most $(qe^{g(\alpha_1)} + \epsilon)^n + m$ for any positive real number ϵ as n tends to infinity. Table 1 data provides comparison in running time of different algorithms in the Boolean case $q = 2$ for $m = n$ and a variety of l.

TABLE 1. Algorithms' running time: $q = 2$ and $m = n$.

l	3	4	5	6
the worst case, [23]	1.324^n	1.474^n	1.569^n	1.637^n
Gluing1, expectation, [38]	1.262^n	1.355^n	1.425^n	1.479^n
Gluing2, expectation, [38]	1.238^n	1.326^n	1.393^n	1.446^n
Agr.-Gluing, expectation, [40]	1.113^n	1.205^n	1.276^n	1.334^n
r	2	3	3	4
Weak IAG, expectation, [42]	1.029^n	1.107^n	1.182^n	1.239^n

5.3. Improved Agreeing-Gluing algorithm estimates

A weaker version of the Improved Agreeing-Gluing algorithm was estimated in [42]: let r be a parameter. The vectors W_r are generated by the Search algorithm. The variables Z_r are substituted by the entries of $a \in W_r$. New equations, in case $r \geq 3$, are encoded by a CNF formula and Local search algorithm is applied to find all solutions. In contrast to the Gluing algorithm bounds, the complexity estimates are now not explicit. For any of q, l, d, r one finds the maximal value of a real valued function in at most three variables with MAPLE. One then finds r, where it is minimal. The two last lines in Table 1 show the optimal value of r and the expected complexity of the Weak IAG algorithm.

5.4. Average time complexity conjecture

The problem of solving (1) is NP-hard as it is polynomially equivalent to k-SAT problem for $k = \lceil \log_2 q \rceil \, l$ and $\lceil \log_2 q \rceil \, n$ variables. A drastic improvement over the last few years in average time complexity of solving (1) is observed; see Table 1. That might indicate that *There exists an algorithm whose average time complexity on uniformly random instances* (1) *is sub-exponential in* n *as* q *and* l *are fixed,* $m \geq n$ *while* n *tends to infinity,* [42]. We call this statement *Average time complexity conjecture*, if it is proved that it will have far-reaching consequences in the field of cryptanalysis and in computing in general.

6. Linear algebra variation: Equations with multiple right hand sides

6.1. Definitions

Linear equations with multiple right hand sides were introduced in [33]. Motivation comes from cipher equations. One non-linear equation in stream cipher Trivium: $xy = u + v + w + z$, [8]. Each solution satisfies one of four linear equations: the left hand side is the same, but the right hand side has four variations.

$$\begin{pmatrix} x \\ y \\ u+v+w+z \end{pmatrix} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Generally, let $f(X) = 0$ be any polynomial equation and

$$AX = a_1, \ldots, a_r \tag{6}$$

a set of linear equation systems. Then (6) is called a Multiple Right Hand Side (MRHS) linear equation for $f(X) = 0$ if the set of solutions to $f(X) = 0$ is the union of solutions to particular linear systems $AX = a_i$.

6.2. Constructing MRHS

In practice, there is often no need to construct (6) for f as it is already given in the definition of the problem or very easy to deduce. This is so for the AES and many other modern ciphers; see [33], [34], [36]. Generally, the space $G(f)$ of Boolean n-vectors a satisfying $f(X + a) = f(X)$ is computed. Non Boolean case is treated in [36]. This computation takes at most 2^{2n} n-bit xor's in the worst case. The space $G(f)$ is of rank m for some $0 \leq m \leq n$. There exist n-vectors $b_1, \ldots, b_m$, a basis for $G(f)$. Take any matrix A of size $k \times n$ and of rank $k = n - m$ such that $Ab_i = 0$ for all $i = 1, \ldots m$. The matrix A is computed by solving m homogenous independent linear equations in n variables. Define now the Boolean function g in k variables by the rule $g(b) = f(a)$ for any Boolean k-vector b such that $b = Aa$. The function g is correctly defined as from $Aa = Aa'$ it follows that $f(a) = f(a')$. Then $f(X) = g(AX)$, where $g(Y)$ is a Boolean function in $k \leq n$ variables Y. We call the representation nontrivial if $k < n$. Let now $a_1, a_2, \ldots, a_r$ be all solutions to the equation $g(Y) = 0$. Then the equation $f(X) = 0$ is described by the system of the MRHS linear equations (6). For example,

$$x_1x_2+x_1x_4 + x_2x_4 + x_2 + x_3 + x_4 = y_1y_2 + y_1 + y_2 + y_3,$$
$$y_1 = x_1 + x_2, \quad y_2 = x_1 + x_4, \quad y_3 = x_1 + x_3.$$

In matrix form

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}. \tag{7}$$

Hence $f(X)$ can be written as $g(AX)$, where $g(Y)$ only takes three variables.

6.3. Gluing, agreeing and linear equation extracting

Assume two equations $f_1(X) = 0$ and $f_2(X) = 0$ in MRHS form

$$AX = a_1, \ldots, a_r \tag{8}$$

and

$$BX = b_1, \ldots, b_s. \tag{9}$$

With gluing one produces a MRHS for the system of two equations $f_1(X) = 0$, $f_2(X) = 0$, that is for their common solutions. A common space generated by the rows of the matrices A and B is computed. Two right hand sides a_i and b_j are combined if they have the same projection to the common space. Equivalently, the linear system

$$\begin{pmatrix} A \\ B \end{pmatrix} X = \begin{matrix} a_i \\ b_j \end{matrix} \tag{10}$$

is consistent. The result of the Gluing is $CX = c_1, \ldots, c_t$, where $C = \binom{A}{B}$ and c_i are right hand sides in (10) which are found consistent by triangulation. Inconsistent equations (10) are discarded. In agreeing the equation a_i is removed from (8) if (10) is inconsistent for every $j = 1, \ldots, s$. Equally, b_j is removed from (9) if (10) is inconsistent for every $i = 1, \ldots, r$.

Also one may triangulate the right hand side columns in (6) and extracts linear equations from constant rows. Linear equations are collected as one equation with a single right hand side. They may be used to eliminate some variables.

6.4. Solving strategy

Agreeing, gluing and linear equation extracting are combined with variable guessing to solve a system of MRHS linear equations:

$$A_1X = [L_1], \ldots, A_mX = [L_m]. \tag{11}$$

In practice, there should be a threshold for the number of right hand sides in equations produced with gluing. Extended Agreeing2 method in [33] may be used to simplifying the equations (reducing the number of the right hand sides) after a guess.

6.5. MRHS equations from AES

SR*(n, r, c) are scaled variants of the AES introduced in [10], where n is the number of rounds and $r \times c$ the cipher state size. So that SR*$(10, 8, 8)$ is the full AES128. The equation variables are all bits from the user selected key K_0, $8rc$ variables, and all bits in the leftmost columns of the round keys K_i, $i = 1, \ldots, n$, that is $8nr$ variables, and all bits in the cipher block after application of the S-boxes in each round, except for the last, $8(n-1)rc$ variables. For example, the bytes that are variables of SR*(3,4,4) are shown, marked with 'X', in Figure 4. Any bit in any of the round keys, and any bit at any stage of the encryption can

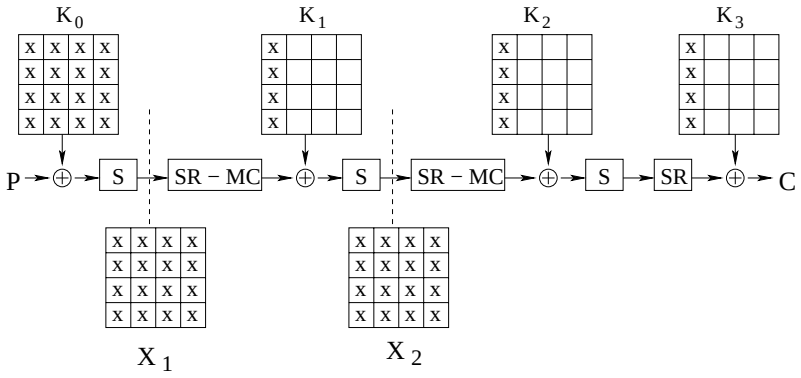

FIGURE 4. SR*(3,4,4) equations variables.

be expressed as a linear combination of in total $8rn(c+1)$ Boolean variables.

Let the eight linear combinations going into one S-box be denoted $l_0, \ldots, l_7$ and the eight linear combinations going out of the same S-box be $l_8, \ldots, l_{15}$. There are 256 possible inputs to the S-box, each one producing a unique output. Hence there are 256 possible values the 16 linear combinations $l_0, \ldots, l_{15}$ can have:

$$\begin{pmatrix} l_0 \\ \ldots \\ l_{15} \end{pmatrix} = \begin{bmatrix} 0 & 1 & \ldots & F \\ 0 & 0 & \ldots & F \\ 3 & C & \ldots & 6 \\ 6 & 7 & \ldots & 1 \end{bmatrix}. \tag{12}$$

There are r S-boxes used when computing K_{i+1} from K_i, and there are rc S-boxes used in each round of SR*(n, r, c). Hence the total number of MRHS linear equations making up the system is $nr(c+1)$. That is 5760 Boolean variables and 720 equations as (12) for the full AES.

6.6. Experiments with cipher equations

In [33] MRHS linear equations from the cipher SR*(n, r, c) for various choice of r, c and n, including those from the full AES, were solved after a number of guesses. When the key size k was above 16 bits, two equations could be glued if the number of right hand sides in the glued equation was at most 2^{16}.

It was found that the number of key bits needed to guess when storing equations with up to 2^l right hand sides is almost always $k - l$. This observation remains true for MRHS linear equations from the DES [34], [36]. This number-of-guesses/memory tradeoff means it is (theoretically) possible to solve the full AES system by guessing 64 key bits when allowing equations to have up to 2^{64} right hand sides.

6.7. Experiments with random equations

Described in [33]. The A-matrices in random equations (11) are $16 \times n$-matrix with random linear independent rows, for various values of n. Each equation puts an 8-bit constraint on the solution space, so for $m = n/8$ we would expect the system to have about one solution. Then $L_1 = \cdots = L_m$ are the right hand sides of the AES equations (12).

MAGMA is a computer algebra program that has an efficient implementation of the F4-algorithm [18] for computing Gröbner bases. This method is considered to be state-of-the-art when it comes to solving general non-linear equation systems.

In the first series of the experiments a system of $m = 6$ random equations with $n = 48$ variables was constructed. Gluing (max 2^{16} right hand sides), agreeing and extracting linear equations solve the system in between one and two seconds. The 24 linearly independent quadratic polynomials listed in [11] together with 16 extra quadratic polynomials from [9] was used to generate the multivariate polynomials describing the same system. Together with the field polynomials that makes 288 quadratic polynomials in 48 variables. MAGMA returned the same solutions after more than twelve and half hours of computing on the same machine with a 450 MHz UltraSparc II processor.

For a system of $m = 7$ random AES equations in $n = 56$ variables the value of seven of the variables was necessary to guess before the system is solved with gluing and agreeing. The total time used for this was five and half minutes. For the corresponding set of quadratic polynomials MAGMA consumed all available memory, more than 3 GB, before it exited with an out-of-memory message.

7. Linear algebra variation: equations with multiple sides

7.1. Definitions

Linear equations with multiple sides were introduced in [40]. This is a more general type of equations than MRHS linear equations. Motivation again comes from cipher equations. One equation in Trivium: $xy = u + v + w + z$. Each solution satisfies one of two linear equations:

$$\begin{aligned} x &= 0, & x &= 1, \\ u + v + w + z &= 0, & y + u + v + w + z &= 0. \end{aligned}$$

We come to the following definition. Let $f(X) = 0$ be any polynomial equation and

$$A_1 X = a_1, \ldots, A_r X = a_r \tag{13}$$

a set of linear equation systems. Then (13) is called a Multiple Side (MS) linear equation for $f(X) = 0$ if the set of solutions to $f(X) = 0$ is the union of solutions to particular linear systems $A_iX = a_i$. In case $A_1 = A_2 = \cdots = A_r$ that is a system of MRHS linear equations.

7.2. Constructing MS linear equations

Assume one equation $f(X) = 0$. Let V be all solutions to $f(X) = 0$. One represents V as a union of its sub-cosets:

$$V = \bigcup_{j=1}^{r} (u_j + U_j). \tag{14}$$

Coset $u_j + U_j$ presents all solutions to a linear equation system $A_jX = a_j$. So (13) follows. In order to get an efficient solution to a system of such equations, the number of the linear equations in (13) may be taken as low as possible.

Example. For equation (3) the cosets are

$$\begin{matrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{matrix}, \quad \begin{matrix} 0 & 1 \\ 1 & 0 \\ 1 & 1 \\ 1 & 1 \end{matrix}, \quad \begin{matrix} 1 & 1 \\ 1 & 1 \\ 0 & 1 \\ 1 & 1 \end{matrix}.$$

So the MS linear representation is

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

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} X = \begin{matrix} 1 \\ 1 \\ 1 \end{matrix}, \quad \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} X = \begin{matrix} 1 \\ 1 \\ 1 \end{matrix},$$

where X is a column vector of variables x_1, x_2, x_3, x_4.

7.3. MS from MRHS

Assume one MRHS equation:

$$BX = b_1, \ldots, b_s, \tag{15}$$

where B is a matrix of size $k \times n$, so that right hand sides vectors $V = \{b_1, \ldots, b_s\}$ are of length k. One constructs a coset cover (14) for V and so a MS system $A_1Y = a_1, \ldots, A_rY = a_r$, where Y consists of k auxiliary variables. Therefore, equation (15) is written as

$$(A_1B)X = a_1, \ldots, (A_rB)X = a_r, \tag{16}$$

where r, the number of sides, is generally lower than s.

7.4. AES equations in MS form

Typical MRHS AES equation (12) has 2^8 right hand sides. The set of the right hand side vectors $(a, S(a))$ is representable as a union of 64 co-sets of size 4 each. That implies (13) with $r = 64$.

7.5. Gluing and agreeing and linear equations extracting

Assume two equations $f_1(X) = 0$ and $f_2(X) = 0$ in MS form

$$A_1X = a_1, \ldots, A_rX = a_r \tag{17}$$

and

$$B_1X = b_1, \ldots, B_sX = b_s. \tag{18}$$

With gluing one produces a MS form for the system of two equations $f_1(X) = 0$, $f_2(X) = 0$, that is for their common solutions. For each pair of indexes i, j the linear system

$$\begin{pmatrix} A_i \\ B_j \end{pmatrix} X = \begin{matrix} a_i \\ b_j \end{matrix} \tag{19}$$

is checked for consistency by triangulation of the left hand side matrix. The common solutions are represented by all consistent (19). In agreeing the equation $A_iX = a_i$ is removed from (17) if (19) is inconsistent for every $j = 1, \ldots, s$. Equally, $B_jX = b_j$ is removed from (18) if (19) is inconsistent for every $i = 1, \ldots, r$.

Also ordinary linear equations may be extracted from any MS equation, e.g., (17). They compose a space of dimension $n + 1$ vectors over F_q. That space is an intersection of spaces generated by linear equations in $A_iX = a_i$ and easy to compute.

7.6. Solving an MS-system by gluing and agreeing

Any combination of variable guessing, gluing and agreeing, and ordinary linear equations extracting is used to produce a MS representation for the solution set of all equations. The solutions themselves are then deduced by solving particular linear equations from that representation.

7.7. Experiments with MS representation

The efficiency of some of ageeing-gluing routines described in Section 4 in case of l-sparse randomly generated Boolean equations was compared in [40], [44] with their MS counterparts. For instance, let $n = m = 75$ and $l = 4$ with random Boolean polynomials, where the number of local solutions is 8 for each of equations. The experiment is characterized by two vectors T and S of length n. For pure gluing the instance t_i of the vector T is the number of solutions to the first i equations. The instance s_i of S is the number of linear systems in (13) representing the solutions to the first i equations. From one instance with MAPLE

linear algebra routines it was found that:

i	1	2	3	4	5	6	7	..58	..64	...
T	8	34	136	1088	2432	19456	38912	**645404416**	227123200	...
S	3	7	8	24	117	351	837	5494938	**6733966**	...

,

where the maximal entry is in bold, [40]. Blue T-curve and red S-curve are shown in Figure 5 in logarithmic scale, where S-curve was generated approximately 4 times faster. That picture appears typical for many other experiments of that sort in [44]. They show that MS solving strategy may be more efficient in sparse Boolean equations.

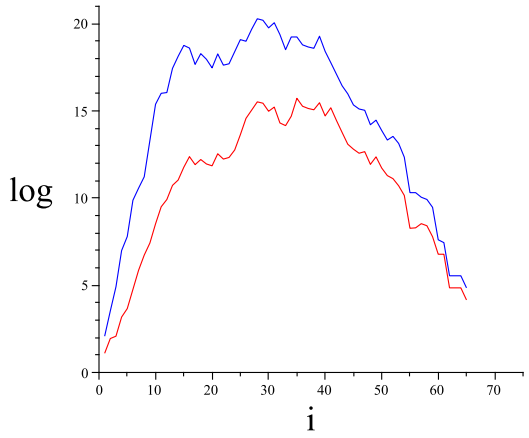

FIGURE 5. Gluing vs MS Gluing.

8. Hardware architectures for Agreeing-Gluing algorithms

Two different hardware architectures were suggested in [40], [41] and in [22]. The first describes a possible hardware implementation of the Agreeing2 algorithm in Section 4.4 as a lattice of switches and wires. As this is not a common computer, the switches are not necessary to be synchronized. A system for the DES may be implemented with 8.5×10^6 switches and 4×10^8 wires(2.7×10^7 and 1.1×10^9 for the TripleDES), where the number of switch turns for rejecting one DES key is 34(98 for the TripleDES) on the average. A transistor on a semi-conductor crystal may work as a switch. One modern Intel chip provides with more than 10^9 transistors. Transistor speed may achieve 1000 GHz. That

probably opens new perspectives in brute force cryptanalysis of modern ciphers. At least, theoretically, this is faster than with COPACOBANA [25].

The second paper [22] describes architecture for hardware implementation of main MRHS routines.

REFERENCES

[1] ARS, G.—FAUGÈRE, J. C.—IMAI, H.—KAWAZOE, M.—SUGITA, M.: *Comparison between XL and Gröbner basis algorithms*, in: Advances in Cryptology—ASIACRYPT '04 (P. J. Lee, ed.), Lecture Notes in Comput. Sci., Vol. 3329, Springer-Verlag, Berlin, 2004, pp. 338–353.

[2] BARDET, M.—FAUGÉRE, J.-C.—SALVY, B.: *Complexity of Gröbner basis computation for semi-regular overdetermined sequences over F_2 with solutions in F_2*, Research report 5049, INRIA, 2003.

[3] BARDET, M.—FAUGÈRE, J. C.—SALVY, B.—YANG, B. Y.: *Asymptotic behaviour of the degree of regularity of semiregular polynomial systems*, in: Proc. of MEGA '05, 8th International Symposium on Effective Methods in Algebraic Geometry Porto Conte, Alghero, Sardinia, Italy, May 27–June 1, 2005.

[4] BUCHBERGER, B.: *An Algorithm for Finding a Basis for the Residue Class Ring of a Zero-Dimensional Polynomial Ideal*. PhD. Thesis, Univ. of Innsbruck, Math. Inst., Innsbruck, 1965.

[5] BUCHBERGER, B.: *A theoretical basis for the reduction of polynomials to canonical forms*, ACM SIGSAM Bull. **10** (1976), no. 3, 19–29.

[6] BUCHBERGER, B.: *Some properties of Gröbner-bases for polynomial ideals*, in: ACM SIGSAM Bull. **10** (1976), no. 4, 19–24.

[7] BUCHBERGER, B.: *A criterion for detecting unnecessary reductions in the construction of Gröbner bases*, in: Proc. of EUROSAM '79, An International Symposiumon Symbolic and Algebraic Computation (W. Ng. Edward, ed.), Lecture Notes in Comput. Sci., Vol. 72, Springer-Verlag, Berlin, 1979, pp. 3–21.

[8] DE CANNIERE, C.—PRENEEL, B.: *Trivium specifications,* `http://www.ecrypt.eu.org/stream/`.

[9] CHEON, J. H.—LEE, D. H.: *Resistance of S-Boxes against Algebraic Attacks*, in: Fast Software Encryption—FSE '04, 11th Internat. Workshop (B. Roy et al., eds.), Lecture Notes in Comput. Sci., Vol. 3017, Springer-Verlag, Berlin, 2004, pp. 83–94.

[10] CID, C.—MURPHY, S.—ROBSHAW, M.: *Small Scale variants of the AES*, in: Fast Software Encryption—FSE '05, 12th Internat. Workshop (H. Glibert et al., eds.), Lecture Notes in Comput. Sci., Vol. 3557, Springer-Verlag, Berlin, 2005, pp. 145–162.

[11] COURTOIS, N.—PIEPRZYK, J. : *Cryptanalysis of block ciphers with overdefined systems of equations*, in: Advances in Cryptology—ASIACRYPT '02, 8th Internat. Conference on the Theory and Application of Cryptology and Information Security (Y. Zheng, ed.), Lecture Notes in Comput. Sci., Vol. 2501, Springer-Verlag, Berlin, 2002, pp. 267–287.

[12] COURTOIS, N.—KLIMOV, A.—PATARIN, J.—SHAMIR, A.: *Efficient algorithms for solving overdefined systems of multivariate equations*, in: Advances in Cryptology––EUROCRYPT '00, Internat. Conference on the Theory and Application of Cryptographic Techniques (B. Preneel, ed.), Lecture Notes in Comput. Sci., Vol. 1807, Springer-Verlag, Berlin, 2000, pp. 392–407.

[13] BARD, G. V.—COURTOIS, N. T.—JEFFERSON, C.: *Efficients methods for conversion and solution of sparse systems of low-degree multivariate polynomials over GF*(2) *via SAT-solvers*, `http://eprint.iacr.org/2007/024`.

[14] BARD, G. V.—COURTOIS, N. T.: *Algebraic cryptanalysis of the data encryption standard*, in: Cryptography and Coding, 11th IMA Internat. Conference (S. Galbraith, (ed.), Lecture Notes in Comput. Sci., Vol. 4887, Springer-Verlag, Berlin, 2007, pp. 152–169, `http://eprint.iacr.org/2006/402`.

[15] DANTSIN, E.—GOERDT, A.—HIRSCH, E. A.—KANNAN, R.— KLEINBERG, J.––PAPADIMITRIOU, C. H.—RAGHAVAN, P.—SCHÖNING, U.: *A deterministic* $(2-\frac{2}{k+1})^n$ *algorithm for k-SAT based on local search,* Theoret. Comput. Sci. **289** (2002), 69–83.

[16] DAVIS, M.—PUTNAM, H.: *A computing procedure for quantification theory*, J. Assoc. Comput. Mach. **7** (1960), 201–215.

[17] DAVIS, M.—LOGEMANN, G.—LOVELAND, D.: *A machine program for theorem proving,* Commun. ACM **5** (1962), 394–397.

[18] FAUGÈRE, J. C.: *A new efficient algorithm for computing Gröbner bases(*F_4*)*, J. Pure Appl. Algebra **139** (1999), 61–88.

[19] FAUGÈRE, J. C.: *A new efficient algorithm for computing Gröbner basis without reduction to zero (*F_5*)*, in: Symbolic and Algebraic Computation—ISSAC '02, Internat. Symposium (T. Mora, ed.), ACM Press, New York, NY, 2002, pp. 75–83.

[20] FAUGÈRE, J. C.—JOUX, A.: *Algebraic cryptanalysis of hidden field equation (HFE) cryptosystems using Gröbner bases*, in: Advances in Cryptology—CRYPTO '03, 23rd Annual Internat. Cryptology Conference (D. Boneh, ed.), Lecture Notes in Comput. Sci., Vol. 2729, Springer-Verlag, Berlin, 2003, pp. 44–60.

[21] GEBAUER, R.—MÖLLER, H. M.: *On an installation of Buchberger's algorithm*, J. Symbolic Comput. **6** (1988), 275–286.

[22] GEISELMANN, W.—MATHEIS, K.—STEINWANDT, R.: *PET SNAKE: A special purpose architecture to implement an algebraic attack in hardware,* `http://eprint.iacr.org/2009/222`.

[23] IWAMA, K.: *Worst-case upper bounds for kSAT,* Bull. EATCS **82** (2004), 61–71.

[24] JOUX, A.—VITSE, V.: *A variant of the F4 algorithm*, `http://eprint.iacr.org/2010/158`.

[25] KUMAR, S.—PAAR, C.—PELZL, J.—PFEIER, G.—SCHIMMLER, M.: *Breaking ciphers with Copacobana-a cost-optimized parallel code breaker,* Lecture Notes in Comput. Sci., Vol. 4249, Springer-Verlag, Berlin, 2006, pp. 101–118.

[26] KWAN, M.: *Reducing the gate count of bitslice DES,* `http://eprint.iacr.org/2000/051`.

[27] LAZARD, D.: *Gröbner bases, Gaussian elimination and resolution of systems of algebraic equations*, in: Computer Algebra—EUROCAL '83, Proc. Conf., London 1983, Lect. Notes Comput. Sci., Vol. 162, Springer-Verlag, Berlin, 1983, pp. 146–156.

[28] MARQUES-SILVA, J. P.—SAKALLAH, K. A.: *GRASP: a search algorithm for propositional satisfiability,* IEEE Trans. Comput. **48** (1999), 506–521.

[29] PAPADIMITRIOU, C. H.: *On selecting a satisfying truth assignment,* in: Proc. of FOCS '91, 32nd Annual Symposium of Foundations of Computer Science, IEEE Comput. Soc., Washington, DC, USA, 1991, pp. 163–169.

[30] PATURI, R.—PUDLÁK, P.—SAKS, M. E.—ZANE, F.: *An improved exponential-time algorithm for k-SAT,* J. ACM **52** (2005), 337–364.

[31] RADDUM, H.: *Solving non-linear sparse equation systems over $GF(2)$ using graphs,* University of Bergen, 2004 (preprint).

[32] RADDUM, H.—SEMAEV, I.: *New technique for solving sparse equation systems*, `http://eprint.iacr.org/2006/475`.

[33] RADDUM, H.—SEMAEV, I.: *Solving multiple right hand sides linear equations*, Des. Codes Cryptogr. **49** (2008), 147–160, extended abstract in: Proc. of WCC '07, Versailles, France, INRIA, 2007.

[34] RADDUM, H.: *MRHS equation systems*, in: Selected Areas in Cryptography—SAC '07, 14th Internat. Workshop (C. Adams et al., eds.), Lecture Notes in Comput. Sci., Vol. 4876, Springer-Verlag, Berlin, 2007, pp. 232–245.

[35] SCHILLING, T. E.—RADDUM, H.: *Solving equation systems by agreeing and learning*, in: WAIFI '10, 2010 (to appear).

[36] SCHOONEN, A. C. C.: *Multiple Right-Hand Side Equations. A New Tool for Cryptanalysis*. Master's Thesis, Eindhoven Univ. of Technology, Eindhoven, 2008.

[37] SCHÖNING, U.: *A probabilistic algorithm for k-Sat based on limited local search and restart*, Algorithmica **32** (2002), 615–623.

[38] SEMAEV, I.: *On solving sparse algebraic equations over finite fields*, Des. Codes Cryptogr. **49** (2008), pp. 47–60.

[39] SEMAEV, I.: *Sparse algebraic equations over finite fields*, SIAM J. Comput. **39** (2009), 388–409.

[40] SEMAEV, I.: *Multiple side linear equations and circuit lattices*, `http://conf.fme.vutbr.cz/cecc09/lectures/semaev.pdf`.

[41] SEMAEV, I.: *Sparse Boolean equations and circuit lattices*, Des. Codes Cryptogr., 2010 (to appear).

[42] SEMAEV, I.: *Improved Agreeing-Gluing algorithm*, `http://eprint.iacr.org/2010/140`.

[43] YANG, B.-Y.—CHEN, J-M.—COURTOIS, N.: *On asymptotic security estimates in XL and Gröbner bases-related algebraic cryptanalysis*, in: Information and Communications Security—ICICS '04, 6th Internat. Conference (J. Lopez et al., eds.), Lecture Notes in Comput. Sci., Vol. 3269, Springer-Verlag, Berlin, 2004, pp. 401–413.

[44] WANGDUE, R.: *Multiple Side Linear Equations: A New Tool for Solving Sparse Algebraic Equations in Finite Field*, Master Thesis, University of Bergen, Bergen, 2010.

[45] WIEDEMANN, D. H.: *Solving sparse linear equations over finite fields,* IEEE Trans. Inform. Theory **32** (1986), 54–62.

[46] ZAKREVSKIJ, A.—VASILKOVA, I.: *Reducing large systems of Boolean equations*, in: 4th Internat. Workshop on Boolean Problems, Freiberg University of Mining and Technology, Freiberg, 2000, pp. 21–28.

[47] ZHANG, L.—MALIK, S.: *The quest for efficient Boolean satisfiability solvers*, Lecture Notes in Comput. Sci., Vol. 2404, Springer-Verlag, Berlin, 2002, pp. 17–36.

Received April 30, 2010

Igor Semaev
Department of Informatics
University of Bergen
N–5020 Bergen
NORWAY
E-mail: igor@ii.uib.no

Michal Mikuš
Department of Applied Informatics
and Computing Technologies
Faculty of Electrical Engineering
Slovak Technical University
Ilkovičova 3
SK–812-19 Bratislava
SLOVAKIA
E-mail: michal.mikus@stuba.sk