Solving MRHS linear equations

Håvard Raddum and Igor Semaev

Department of Informatics, University of Bergen, Norway

Abstract. A new method for solving algebraic equation systems common in cryptanalysis is proposed. Our method differs from the others in that the equations are not represented as multivariate polynomials, but as a system of Multiple Right Hand Sides linear equations. The method was tested on scaled versions of the AES. The results overcome significantly what was previously achieved with Gröbner Basis related algorithms.

Keywords: Multiple Right Hand Sides linear equations, algebraic attacks, AES.

1 Introduction

Most of the cryptanalysis done on symmetric key ciphers in the last few years has been focused on algebraic attacks. One important feature algebraic attacks have is the fact that you only need very few known plain-texts in order to set up an equation system describing the cipher and determining the key uniquely. Strategies for solving non-linear equation systems have been described [7, 8, 11] and some have been developed into cryptanalytic attacks [3–5]. All of them are based on the Gröbner Basis (or XL) related algorithms which make use of multivariate polynomial representation of equations.

In this paper the equations describing encryption are represented as systems of Multiple Right Hand Sides(MRHS) linear equations. This is a more general representation than that independently introduced in [9] and earlier in [12]. Algorithms to solve MRHS linear equations are presented. Linear substitutions is a common tool providing diffusion properties of encryption in modern ciphers. So MRHS linear equations, which take into account such layers, make equations representation quite compact and computations more efficient in comparison with methods based on the Gröbner Basis related algorithms.

Our experiments have been inspired by the work in [2], where the smaller clones of the AES were defined. We adopt the notation from that paper: with SR*(n, r, c) we will mean the variant of the AES that has n rounds, and where the cipher block has r rows and c columns, see [2] for details. Variants using the field $GF(2^4)$ could also be considered, but in our tests we only looked at AES variants where the underlying finite field is $GF(2^8)$.

In [2] standard techniques (F4 and Buchberger's algorithm) for solving non-linear equation systems were applied to some of the systems representing the small versions of the AES, to see which ones that actually could be solved using this approach on a computer with 1GB of RAM. The computer could not solve the system for SR*(5,1,1) this way, even though the key is only 8 bits long. Solving SR*(4,1,1) took 20286.18 seconds. This suggests that using these techniques may be the wrong way to go with algebraic cryptanalysis of the AES. The methods proposed in this paper are general in nature, and we will show that they make a far better approach to solving equations from the different versions of the AES. For instance, SR*(4,1,1) is solved in 0.032 seconds without guessing any variables and more complicated instances are solved on a common computer.

Besides the Introduction, the paper comprises Sections 2, 3, where MRHS linear equations and their systems are presented. Sections 4,5 and 6 describe the main techniques Agreeing, Gluing and Linear equations extracting we use. The Agreeing being the core approach is presented in much detail. It is shown in Section 7 how the above techniques are combined in equations solving. Finally, our experiments with reduced versions of the AES are described in Section 8.

This is the full version of an abstract earlier published in [10].

2 Multiple Right Hand Side linear equations

All matrices and vectors are over $GF(2)$, the field with two elements. A $GF(q)$ variant of the method is easy to deduce. Let X be a set of n Boolean variables represented as a column-vector. An equation

$$AX = a_1, a_2, \ldots, a_s \tag{1}$$

is called a MRHS system of linear equations if A is a matrix of size $k \times n$ and rank k, and $a_1, a_2, \ldots, a_s$ are column-vectors of length k. Solution to (1) is a Boolean n-vector satisfying one of the particular linear equation systems $AX = a_i$. The set of all solutions to (1) is the union of solutions to the linear systems for all a_i. Suppose $f(X)$ is a Boolean function such that

$$f(X) = g(AX), \tag{2}$$

where $g(Y)$ is a Boolean function in $k \leq n$ variables Y and A is a $k \times n$-matrix of rank k. We call the representation (2) nontrivial if $k < n$. Let now $a_1, a_2, \ldots, a_s$ 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 (1) and vice versa (1) implies (2). One also represents the right hand sides of (1) as a $k \times s$ matrix L whose columns are $a_1, a_2, \ldots, a_s$. So that (1) becomes $AX = [L]$ and is called a symbol, we write $[L]$ to stress that is not an ordinary equality with matrices. If the matrix A has just one nonzero entry in each row, we get the old definition of a symbol from [9].

If $f(X)$ is given as a multivariate polynomial or by a truth table and the number of variables n is small we can map it into a symbol in the following way. The space $G(f)$ of Boolean n-vectors a satisfying $f(X + a) = f(X)$ is computed.

This computation takes at most 2^{2n} n-bit xor's. 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 and the following statements are obvious.

Lemma 1. *1. The representation* (2) *holds for the so defined Boolean function g and matrix A.*
2. The representation (2) *is nontrivial if and only if the rank of $G(f)$ is nonzero.*

The running time of the procedure does not exceed the cost of computing $G(f)$ that is 2^{2n} n-bit xor's.

For example, the polynomial

$$x_1x_2+x_1x_4+x_2x_4+x_2+x_3+x_4 = (x_1+x_2)(x_1+x_4)+(x_1+x_2)+(x_1+x_3)+(x_1+x_4)$$

has four variables, but can be written using only three linear combinations. Hence $f(X)$ can be written as $g(AX)$, where $g(Y)$ only takes three variables, and not four, ref. (2).

In practice, there is often no need to construct (1) 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.

3 MRHS equation systems

A system of MRHS linear equations (symbols):

$$S_1 : A_1X = [L_1], \quad \ldots, \quad S_m : A_mX = [L_m] \tag{3}$$

is considered, where A_i and L_i are matrices such that the number k_i of rows in A_i and L_i is bounded by a number k. Some of the variables may be irrelevant to some equations. This means that the related columns in A_i are zero. The maximal number of columns in L_i is 2^{k_i}. So the number k should be relatively small in order to keep all symbols (3) in memory. The solution to (3) is an assignment to variables X satisfying all the equations. The goal is to find all solutions. We will present a generalization of the technique introduced in [9]. The core approach is to remove some of the L_i-columns a if no one solution of $A_iX = a$ can be a solution to (3). This is partially achieved with pairwise agreeing.

4 Agreeing

Let two symbols

$$S_i : A_iX = [L_i] \quad \texttt{and} \quad S_j : A_jX = [L_j] \tag{4}$$

be given. The matrices L_i are of size $k_i \times s_i$. We say the symbols (4) agree if for any $a_1 \in L_i$, there exists an $a_2 \in L_j$ such that the linear system

$$\begin{pmatrix} A_i \\ A_j \end{pmatrix} X = \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} \tag{5}$$

is consistent, and for any $a_2 \in L_j$ there exists an $a_1 \in L_i$ such that (5) is consistent.

When S_i and S_j do not agree one removes columns a_1 from L_i such that $A_iX = a_1$ is inconsistent with $A_jX = [L_j]$. Similarly, one removes columns a_2 from L_j if $A_jX = a_2$ is inconsistent with $A_iX = [L_i]$. The straightforward approach is to remove a_1 if the pair $A_iX = a_1$ and $A_jX = a_2$ composes an inconsistent system of linear equations for any column $a_2 \in L_j$. This requires $O(s_1s_2)$ linear algebra steps to agree S_i and S_j.

We will present a faster algorithm for agreeing. Let $A = \binom{A_i}{A_j}$ be the concatenation of the matrices A_i and A_j found in (5), so that A is a matrix with $t = k_i + k_j$ rows. Similarly, $T_{ij} = \begin{pmatrix} L_i \\ 0 \end{pmatrix}$ and $T_{ji} = \begin{pmatrix} 0 \\ L_j \end{pmatrix}$ are matrices with t rows.

Agreeing Procedure.

1. A nonsingular transform matrix $U = U_{ij}$ of size $t \times t$ such that the product UA is a matrix with zeros in its last $r = r_{ij}$ rows and of rank $t - r$ is produced. If $r = 0$, then the symbols agree.
2. If $r > 0$, then the matrices UT_{ij} and UT_{ij} are computed. Let Pr_{ij} denote the set of UT_{ij}-column projections to the last r coordinates. In case $Pr_{ij} = Pr_{ji}$, the symbols agree.
3. If $Pr_{ij} \neq Pr_{ji}$, then one removes all L_i-columns whose image under the mapping $\varphi_{ij} : L_i \to T_{ij} \to UT_{ij} \to Pr_{ij}$ is not found in $Pr_{ij} \cap Pr_{ji}$. Similarly, one removes all L_j-columns whose image under the mapping $\varphi_{ji} : L_j \to T_{ji} \to UT_{ji} \to Pr_{ji}$ is not found in $Pr_{ij} \cap Pr_{ji}$. This is easily accomplished with sorted sets Pr_{ij}. So the new matrices L'_i and L'_j, consisting of the survived columns, are produced from L_i and L_j. The symbols $A_iX = [L'_i]$ and $A_jX = [L'_j]$ agree.

The algorithm is approved by the following statement.

Lemma 2. *The symbols* (4) *agree if and only if* $r = 0$ *or* $Pr_{ij} = Pr_{ji}$*.*

Proof. Let the symbols agree and $r > 0$. We will prove $Pr_{ij} = Pr_{ji}$. Take any $b \in Pr_{ij}$ and fix $a_1 \in L_i$ such that $b = \varphi_{ij}(a_1)$. In other words, the projection of $U\bar{a}_1$ to the last r coordinates is b, where $\bar{a}_1 = \begin{pmatrix} a_1 \\ 0 \end{pmatrix} \in T_{ij}$. There exists $a_2 \in L_j$ such that the system of linear equations (5) composed by $A_iX = a_1$ and $A_jX = a_2$ is consistent. The latter is true if and only if $AX = \bar{a}_1 + \bar{a}_2$ is consistent, where $\bar{a}_2 = \begin{pmatrix} 0 \\ a_2 \end{pmatrix} \in T_{ji}$. Then $(UA)X = U\bar{a}_1 + U\bar{a}_2$. The last r coordinates of the column vector $(UA)X$ are zeros. This means the projections of $U\bar{a}_1$ and $U\bar{a}_2$ to the last r coordinates coincide. Therefore, $b \in Pr_{ji}$ and so $Pr_{ij} \subseteq Pr_{ji}$. The inclusion $Pr_{ji} \subseteq Pr_{ij}$ is similarly checked. So $Pr_{ji} = Pr_{ij}$.

We will prove the reverse statement now. Let $r > 0$ and $Pr_{ji} = Pr_{ij}$. Then for any $a_1 \in L_i$ we will find $a_2 \in L_j$ such that (5) is consistent. Let $b = \varphi_{ij}(a_1)$ be the image of a_1 under φ_{ij}. As $Pr_{ji} = Pr_{ij}$, there exists $a_2 \in L_j$ whose image under φ_{ji} is also b. So the column $U\binom{a_1}{a_2} = U\bar{a}_1 + U\bar{a}_2$ has r zeros in its last coordinates. Therefore, the system $(UA)X = U\binom{a_1}{a_2}$ is consistent, as the matrix UA has zeros in its last r rows and is of rank $t-r$. So the system (5) is consistent too. Similarly, we prove that for any $a_2 \in L_j$ one finds $a_1 \in L_i$ such that (5) is consistent. Therefore, the symbols agree.

Finally, assume that $r = 0$. Then the matrix $A = \binom{A_i}{A_j}$ is of the full rank t. That means that the system (5) is consistent for every $a_1 \in L_i$ and $a_2 \in L_j$. Therefore, the symbols agree. This finishes the proof of the Lemma.

The running time of the agreeing is defined by sorting and table look-ups. So with neglecting the contribution from t and n, the running time is $O(s_1 \log s_1 + s_2 \log s_2)$ linear algebra steps. In order to agree symbols several times, as in the case of solving (3), the computation of the matrices $U = U_{ij}$, UT_{ij}, and UT_{ji}. may be done once, as in the Agreeing2 Algorithm presented below.

Example. Two equations $A_1X = [L_1]$ and $A_2X = [L_2]$ in variables $X = \{x_1, x_2, x_3, x_4, x_5\}$:

$$\begin{pmatrix} 1\,1\,0\,0\,0 \\ 1\,0\,1\,0\,0 \\ 1\,0\,0\,1\,0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{bmatrix} 1\,0\,0\,1 \\ 0\,1\,0\,0 \\ 0\,0\,1\,1 \end{bmatrix}, \quad \begin{pmatrix} 0\,1\,0\,0\,1 \\ 0\,0\,1\,0\,1 \\ 0\,0\,0\,1\,1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{bmatrix} 0\,1\,0\,0 \\ 1\,1\,0\,0 \\ 1\,1\,0\,1 \end{bmatrix}$$

are given. In the algebraic normal form they are:

$$x_1x_4 + x_1x_2 + x_2x_4 + x_2 + x_3 + x_4 + 1 = 0,$$
$$x_2x_3 + x_2x_5 + x_3x_4 + x_4x_5 + x_2 + x_3 = 0.$$

The first equation can also be represented in the form of (2) as

$$y_1y_3 + y_1 + y_2 + y_3 + 1 = 0, \qquad \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 1\,1\,0\,0 \\ 1\,0\,1\,0 \\ 1\,0\,0\,1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}.$$

The representation (2) for the second equation is

$$y_1y_2 + y_2y_3 + y_1 + y_2 = 0, \qquad \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 1\,0\,0\,1 \\ 0\,1\,0\,1 \\ 0\,0\,1\,1 \end{pmatrix} \begin{pmatrix} x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix}.$$

The matrix A is produced and transformed with the matrix U:

$$A = \begin{pmatrix} 1\,1\,0\,0\,0 \\ 1\,0\,1\,0\,0 \\ 1\,0\,0\,1\,0 \\ 0\,1\,0\,0\,1 \\ 0\,0\,1\,0\,1 \\ 0\,0\,0\,1\,1 \end{pmatrix} \to UA = \begin{pmatrix} 1\,1\,0\,0\,0 \\ 1\,0\,1\,0\,0 \\ 1\,0\,0\,1\,0 \\ 0\,1\,0\,0\,1 \\ 0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0 \end{pmatrix}, \quad U = \begin{pmatrix} 1\,0\,0\,0\,0\,0 \\ 0\,1\,0\,0\,0\,0 \\ 0\,0\,1\,0\,0\,0 \\ 0\,0\,0\,1\,0\,0 \\ 1\,1\,0\,1\,1\,0 \\ 1\,0\,1\,1\,0\,1 \end{pmatrix}.$$

Then $r = 2$. Put now

$$T_{1\,2} = \begin{pmatrix} 1\,0\,0\,1 \\ 0\,1\,0\,0 \\ 0\,0\,1\,1 \\ 0\,0\,0\,0 \\ 0\,0\,0\,0 \\ 0\,0\,0\,0 \end{pmatrix} \quad \texttt{and} \quad T_{2\,1} = \begin{pmatrix} 0\,0\,0\,0 \\ 0\,0\,0\,0 \\ 0\,0\,0\,0 \\ 0\,1\,0\,0 \\ 1\,1\,0\,0 \\ 1\,1\,0\,1 \end{pmatrix},$$

and compute

$$UT_{1\,2} = \begin{pmatrix} 1\,0\,0\,1 \\ 0\,1\,0\,0 \\ 0\,0\,1\,1 \\ 0\,0\,0\,0 \\ 1\,1\,0\,1 \\ 1\,0\,1\,0 \end{pmatrix} \quad \texttt{and} \quad UT_{2\,1} = \begin{pmatrix} 0\,0\,0\,0 \\ 0\,0\,0\,0 \\ 0\,0\,0\,0 \\ 0\,1\,0\,0 \\ 1\,0\,0\,0 \\ 1\,0\,0\,1 \end{pmatrix}.$$

Look at the last two coordinates of columns in the above matrices and define

$$Pr_{1\,2} = \{(1,1),(1,0),(0,1)\}, \quad Pr_{2\,1} = \{(1,1),(0,0),(0,1)\},$$
$$Pr_{1\,2} \cap Pr_{2\,1} = \{(1,1),(0,1)\}.$$

We see that the second and the fourth columns of $UT_{1\,2}$ don't match any columns of $UT_{2\,1}$. So the second and the fourth columns of L_1 should be removed. Similarly, the second and the third columns of L_2 should be removed. The new symbols are:

$$\begin{pmatrix} 1\,1\,0\,0\,0 \\ 1\,0\,1\,0\,0 \\ 1\,0\,0\,1\,0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{bmatrix} 1\,0 \\ 0\,0 \\ 0\,1 \end{bmatrix}, \quad \begin{pmatrix} 0\,1\,0\,0\,1 \\ 0\,0\,1\,0\,1 \\ 0\,0\,0\,1\,1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{bmatrix} 0\,0 \\ 1\,0 \\ 1\,1 \end{bmatrix},$$

and they agree.

In the next two sections we will present algorithms for producing pairwise agreed symbols from (3). Though the algorithms do not necessarily solve the equation system, they are in the core of our methods for finding solutions.

4.1 Agreeing1 Algorithm

The Agreeing1 Algorithm works by repeatedly finding two indices i and j such that S_i and S_j disagree, and apply the agreeing procedure then. When running the Agreeing1 Algorithm we often run into situations where S_i and S_j agree, but S_j and S_l disagree. After deleting some L_j-columns from S_j to make it agree with S_l, it may well be that S_i and S_j disagree. In other words, applying the agreeing procedure to one pair of symbols may cause disagreement in other pairs. We may get a chain-reaction of deletions of columns that will actually remove a lot of wrong columns.

Agreeing1 Algorithm.

```
while symbols (3) do not pairwise agree
    - find $S_i$ and $S_j$ which do not agree
    - agree $S_i, S_j$ with the Agreeing Procedure.
```

Lemma 3. *1. The output of the Agreeing1 Algorithm doesn't depend on the order of pairwise agreeings.*
2. The running time of the Agreeing1 Algorithm is bounded by $O(m^3k^2 2^{2k})$ bit operations. The memory requirement is $O(mnk + mk2^k)$ bits.

Proof. We will prove the first statement. Consider a set of sub-symbols $A_iX = [U_i]$, where $U_i \subseteq L_i$, meaning that columns of U_i are among columns of L_i, for all $1 \leq i \leq m$. The set of sub-symbols is called a maximal agreed set of sub-symbols if $A_iX = [U_i]$ pairwise agree and for any sets U'_i of columns, where $U_i \subseteq U'_i \subseteq L_i$, with $U_i \subset U'_i$ for at least one i, the sub-symbols $A_iX = U'_i$, $1 \leq i \leq m$ do not pairwise agree. We will prove that the maximal agreed set of sub-symbols is unique and is produced with the Agreeing1 Algorithm regardless of the order of pairwise agreeings.

Assume there are two maximal agreed sets of sub-symbols: $A_iX = [U_i]$, $1 \leq i \leq m$ and $A_iX = [U'_i]$, $1 \leq i \leq m$. Then one constructs the new set of sub-symbols $A_iX = [U_i \cup U'_i]$, $1 \leq i \leq m$. They pairwise agree. That is only possible when $U_i = U'_i$, $1 \leq i \leq m$. So the maximal agreed set of sub-symbols is unique.

Let $A_iX = [U_i]$, $1 \leq i \leq m$ be the output of the Agreeing1 Algorithm and $U_i \subseteq U'_i \subseteq L_i$ for some set of symbols $A_iX = [U'_i]$, $1 \leq i \leq m$ which pairwise agree. An intermediate stage of the Agreeing algorithm is the set of sub-symbols $A_iX = [L'_i]$, where $L'_i \subseteq L_i$. Let this be the stage just before deleting any of the columns in $U'_i \setminus U_i$, $1 \leq i \leq m$. Then $U'_i \subseteq L'_i$, $1 \leq i \leq m$. The Agreeing1 Algorithm now deletes some $a_1 \in U'_i \setminus U_i$. This means there is a symbol $A_jX = [L'_j]$ such that $A_iX = a_1$ is inconsistent with $A_jX = a_2$ for any $a_2 \in L'_j$ and, therefore, for any $a_2 \in U'_j$. This means that the symbols $A_iX = [U'_i]$ and $A_jX = [U'_j]$ do not agree. The contradiction implies $U'_i = U_i$ for all $1 \leq i \leq m$. So $A_iX = [U_i]$, $1 \leq i \leq m$ is the maximal agreed set of sub-symbols. That proves the first statement.

We will prove the second statement now. In order to delete at least one column from some matrix L_i one should find at least one pair S_i and S_j of symbols which disagree. To this end the matrices $U = U_{ij}$, UT_{ij}, and UT_{ji} are computed. Then the symbols are checked for agreeing and made agreed with list sorting of size at most 2^k and table look-ups. That is to delete one column costs $O(m^2k^22^k)$ bit operations in the worst case. As one should delete at most $m2^k$ columns, this implies the running time estimate. The memory requirement is obvious as m matrices A_i and m of L_i should be kept. That finishes the proof of the Lemma.

4.2 Agreeing2 Algorithm

In this section another approach to pairwise agreeing of the equations (3) is presented. It is significantly faster than the previous variant but requires some additional memory.

Agreeing2 Algorithm.

(Precomputation.) For each pair of symbols S_i and S_j the matrix $U = U_{ij}$ and the number $r = r_{ij} \leq k$ of zero rows in $U\binom{A_i}{A_j}$ are computed. If $r = 0$, then there is nothing to do. If $r > 0$, then UT_{ij}, and UT_{ji} are computed, but only $U = U_{ij}$ should be kept for the main step of the algorithm. For each r-bit address b the tuple (L_{ijb}, L_{jib}) is computed and kept. The list L_{ijb} consists of the addresses of columns a_1 in L_i whose image under the mapping φ_{ij} is b. Similarly, L_{jib} denotes the list of addresses of columns a_2 in L_j whose image under φ_{ji} is b. The list of tuples for L_i, L_j is denoted by L_{ij} as in Fig.1.

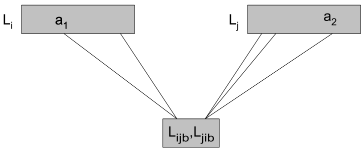

Fig. 1. Precomputation.

(Agreeing.) The Algorithm starts with any tuple (L_{ijb}, L_{jib}), where just one list is empty and follows the rules:

1. Let the list L_{ijb} in the current tuple $(L_{ijb}, L_{jib}) \in L_{ij}$ be empty, while L_{jib} is not. Then all the columns a, whose addresses are in L_{jib}, are deleted from L_j and L_{jib} is made empty. One should actually not delete a in order to not change addresses of other columns. One simply marks a somehow.

2. When the column a is being deleted(marked) from L_j, one computes the addresses $d = \varphi_{jl}(a)$ for $l \neq j$, where $r_{jl} > 0$. Then the address of a in L_j is deleted from L_{jld}, where (L_{jld}, L_{ljd}) is now the current tuple.
3. For each starting tuple the algorithm walks through a search tree with backtracking. If new deletions do not occur in the current tree, then a new tuple, where just one term is empty, is taken and the above steps repeat.
4. The algorithm stops when in all tuples (L_{ijb}, L_{jib}) the terms both are empty or both non-empty.

By Lemma 3 the order of the tuples the algorithm walks through is irrelevant for the result. The complexity of the Agreeing2 Algorithm is estimated with the following Lemma.

Lemma 4. *After $O(m^2k^22^k)$ bit operations of precomputation, the running time of the Agreeing2 Algorithm is bounded by the maximum running time of $O(m^2k^22^k)$ bit operations and $O(m^22^k)$ look-ups in a table comprising at most 2^k binary strings of length at most k. The memory requirement is $O(mnk + m^2k2^k)$ bits.*

Proof. Each deletion of a column $a \in L_i$ results in at most $m-1$ computations of the addresses $b = \varphi_{ij}(a)$, table look-ups, and removals of its address from L_{ijb}. As one should delete at most $m2^k$ columns of the matrices L_i, the algorithm running time is $O(m^2k^22^k)$ bit operations or at most $O(m^22^k)$ look-ups. There should be enough memory locations to keep m matrices A_i of size at most $k \times n$ and m matrices L_i of size at most $k \times 2^k$. Besides, for each pair of symbols S_i, S_j the matrix U_{ij} and the list of tuples L_{ij} comprising at most 2^{k+1} of k-bit addresses each are kept. So the memory requirement is $O(mnk + m^2k2^k)$ bits. This proves the Lemma.

We remark that the Agreeing Algorithms' output may be far away from any solution of the system. What typically happens for the equation systems from ciphers is that all symbols are pairwise agreed already from the beginning, e.g. all $r_{ij} = 0$ or $Pr_{ij} = Pr_{ji}$. So one should do something to start deletions by agreeing. We will describe two approaches to the problem: gluing symbols, which may produce $r_{ij} > 0$ or $Pr_{ij} \neq Pr_{ji}$, and guessing variables which may result in $Pr_{ij} \neq Pr_{ji}$.

5 Gluing

By definition, the gluing of the symbols (4) is a MRHS system of linear equations $BX = [L]$ whose set of solutions is the set of common solutions to $A_iX = [L_i]$ and $A_jX = [L_j]$. The point of this operation is that to represent (find) all common solutions to the above two equations takes at most $O(s_1 \log s_1 + s_2 \log s_2 + s)$ steps of sorting and linear algebra, where s is the number of columns in L. This may be significantly faster than a full search over all strings in relevant variables.

Let B be the sub-matrix of UA in its $t - r$ nonzero rows. The gluing of the symbols is the symbol $BX = [L]$, where L is a matrix of $t - r$ rows and some number of columns. Each column of L is the sum of one column from UT_{ij} and one from UT_{ji} having the same projection to the last r coordinates, and reduced to the first $t-r$ coordinates. One checks that solutions of $BX = [L]$ are exactly all common solutions to the equations (4) in variables X. Then the new symbol $S : BX = [L]$ satisfies all requirements to a MRHS system of linear equations. We denote the result of the gluing as $S = S_1 \circ S_2$.

The straightforward complexity of gluing is at most $O(s_1 s_2)$ linear algebra steps. On the other hand, the bound $O(s_1 \log s_1 + s_2 \log s_2 + s)$ is also true. To demonstrate this one sorts the columns UT_{ij} by their last r-coordinate sub-vectors and glue columns of UT_{ij} and UT_{ji}(compute their sum to produce columns of L) with the same sub-vector by table look-ups. The gluing of the above example symbols is computed as

$$\begin{pmatrix} 1\,1\,0\,0\,0 \\ 1\,0\,1\,0\,0 \\ 1\,0\,0\,1\,0 \\ 0\,1\,0\,0\,1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{bmatrix} 1\,0 \\ 0\,0 \\ 0\,1 \\ 0\,0 \end{bmatrix}. \tag{6}$$

When the symbols (3) reach an agreeing state, we can glue together some of them to create new equations. When gluing S_1 and S_2 together we discard S_1 and S_2, since all information in them are contained in $S_1 \circ S_2$. When we have glued together several pairs of symbols the new set of symbols will in general not be in an agreeing state, so we can start the Agreeing algorithm again. The price to pay when gluing together symbols is longer right hand sides lists. They may be as big as $s_1 s_2$. We may run into cases where we can not afford any symbols to be glued. Then a threshold is set up and we only glue together symbols that produce symbols with the number of the right hand sides below this threshold.

6 From MRHS to URHS

Ordinary (Unique Right Hand Side) linear equations can often be produced from MRHS linear equations. Consider a MRHS system of linear equations: $S : AX = [L]$, where L is a $k \times s$ matrix. We will try to produce linear equations in variables X satisfied by all solutions of $AX = [L]$. To this end one triangulates L with a row transform U to get an upper-triangular matrix with zeros in its last $r_1 \geq 0$ rows. Let A' be the sub-matrix of UA in its last r_1 rows. Then $A'X = 0$ is the system of all independent linear homogenous equations satisfied by the solutions of $AX = [L]$. Non-homogenous equations may occur as well. It is enough to construct one of them. With the above triangulation of L one finds out whether the rows of L generate the all ones vector $\bar{1}$ of length s. If this is so, then the system $bL = \bar{1}$ has a solution b, which is a Boolean k-vector. Then $(bA)X = 1$ is the sought equation. That is, S admits r_1 or $r_1 + 1$ linearly independent linear equations. One sees that if $k \geq s$ there should be at least one linear equation produced from S. E.g. the symbol (6) is transformed to

$$\begin{pmatrix} 1\,1\,0\,0\,0 \\ 0\,1\,0\,1\,0 \\ 1\,0\,1\,0\,0 \\ 0\,1\,0\,0\,1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{bmatrix} 1\,0 \\ 1\,1 \\ 0\,0 \\ 0\,0 \end{bmatrix}$$

with obvious row transformations. This implies three linear equations: $x_2 + x_4 = 1, x_1 + x_3 = 0$ and $x_2 + x_5 = 0$, which are equivalent to the system of the two initial quadratic equations.

7 Solving systems by guessing variables and collecting linear equations

Before starting solving the equations some symbols may be enlarged by gluing up to some threshold. When a variable $x \in X$ is guessed, the linear equation $x = 0$ or $x = 1$ is produced. Several such guesses produce a system of linear equations $A_0 X = a_0$, that is a new symbol S_0. These equations are glued to other symbols (3) during their agreeing in order to enhance deletions. Alternative and equivalent approach is to eliminate variables using current linear equations. After agreeing, the symbols where deletions have occurred are checked for producing new linear equations with the approach from Section 6. Remark that the deletions enhance symbols to imply linear equations as the number of right hand sides in the related MRHS linear systems is reducing. All produced linear equations are being up-glued to the symbol S_0, enlarging the system $A_0 X = a_0$. Then the whole thing repeats.

One concludes that the guess was wrong when all right hand sides of a symbol have been deleted or the system of linear equations in S_0 is inconsistent. Then one backtracts and takes another guess. When neither new linear equations nor new deletions in symbols are found, some guesses of new variables should be made in order to restart the process until there are enough linear equations to produce all solutions to (3) or to reject the current guess.

8 Equation systems for reduced versions of the AES

For readers not familiar with the internal structure of the AES, we refer to [6] or [2]. We consider the general scaled AES, SR*(n, r, c). The variables of the equations we construct will be:

- All bits from the user selected key which is identical to K_0, the round key used to mask the plaintext before the first round, $8rc$ variables.
- All bits in the leftmost columns of the round keys K_i, $i = 1, \ldots, n$, that is $8nr$ variables.
- All bits in the cipher block after application of the S-boxes in each round, except for the last, $8(n-1)rc$ variables. In round i, the block is denoted X_i.

For example, the bytes that are variables of SR*(3,4,4) are shown below, marked with 'X', at Fig. 2. Any bit in any of the round keys, and any bit at any stage of the encryption can be expressed as a linear combination of in total $8rn(c+1)$ Boolean variables.

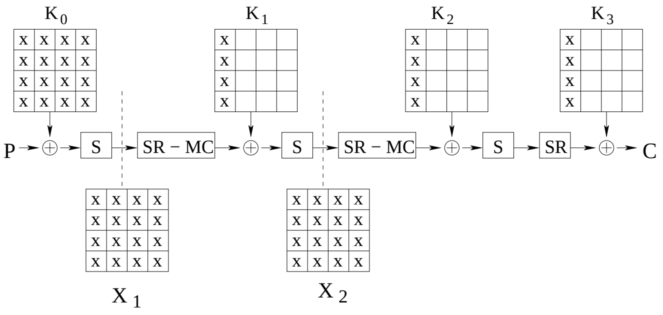

Fig. 2. SR*(3,4,4) equations variables.

Each S-box used in SR*(n, r, c) produces one MRHS linear equation. Each of the eight bits going in or out of one S-box is expressed as a linear combination of the variables, and possibly some plaintext or ciphertext bits (considered known constants), or round constants from the key schedule. 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 $l_8, \ldots, l_{15}$. Let A be the matrix having the coefficients of l_i as row i. There are 256 possible inputs to the S-box, each one producing a unique output. Hence there are 256 possible sets of values the 16 linear combinations $l_0, \ldots, l_{15}$ can have, namely the 16-bit string $(a, S(a))$ for all 8-bit strings a. Collecting this we get the following MRHS equation $AX = [L]$, where the columns of L are written in hexadecimal notation:

$$\begin{pmatrix} l_0 \\ \cdots \\ l_7 \\ l_8 \\ \cdots \\ l_{15} \end{pmatrix} = \begin{bmatrix} 0 & 1 & \ldots & F \\ 0 & 0 & \ldots & F \\ 3 & C & \ldots & 6 \\ 6 & 7 & \ldots & 1 \end{bmatrix}. \tag{7}$$

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 $nr(c+1)$. We expect the system to have a unique solution, so exactly one of the columns in L will be consistent with the solution.

We have implemented SR*(n, r, c) for various choices of r, c and n, constructed the associated MRHS equation systems, and tried to solve them by guessing key bits, running the agreeing algorithm and gluing equations.

For variants where the key is 16 bits or less, the maximum number of columns in L when gluing was set to 2^8. When the key size was above 16 bits, two symbols could be glued if the number of right hand sides in the glued symbol was 2^{16} or less.

We call the exact algorithm used for solving these systems the *G algorithm*. It is designed to only guess variables when the other techniques fail, thus trying to solve the system with as few guesses as possible. It works as follows:

```
while system not solved
    - while something changed since last iteration
        - run agreeing algorithm
        - try to extract linear equations
        - eliminate one variable for each linear equation extracted
        - glue if new number of right hand sides small enough
    - guess one variable from $K_0$
```

The following table gives the number of K_0-bits needed to guess in order to solve the system. For comparison with [2], the time in seconds our computer used for solving SR*$(n, 1, 1)$ is also noted. In the trivial case of SR*$(n, 1, 1)$ no guessing is necessary, the system is always solved by agreeing and gluing alone.

Table 1. The number of guessed key bits before SR*(n, r, c) is solved.

	8-bit key	16-bit key	32-bit key	64-bit key	64-bit key	128-bit key
n	SR*(n,1,1)	SR*(n,2,1)	SR*(n,2,2)	SR*(n,2,4)	SR*(n,4,2)	SR*(n,4,4)
3	0 (0.023)	5	16	48	48	112
4	0 (0.032)	8	16	48	48	112
6	0 (0.076)	8	16	48	48	112
7	0 (0.139)	8	16	48	48	112
10	0 (0.320)	8	16	48	48	112

What we immediately see is that for a k-bit key, the number of key bits needed to guess when storing equations with up to 2^l right hand sides is always $k - l$ (except for the probably degenerate case of three-round SR*$(3, 2, 1)$). This seems to imply that l bits of the key are guessed implicitly when storing equations with 2^l right hand sides. 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} righ hand sides. At this point, this does not indicate a break of the AES, as the complexity of working with equations with 2^{64} right hand sides is of the order of 2^{64}. Each right hand side must be visited at some point, at least when it is deleted, hence the total complexity for solving such a system would still be on the order of 2^{128}. However, if a very fast way of agreeing is found this tradeoff may be important.

It should also be noted that the G algorithm is sensitive to exactly which equations that are chosen to be glued when gluings occur. The above numbers were obtained by always gluing together the two equations that produce the smallest number of right hand sides (assuming it is less than the 2^{16} threshold). We also tried the variant where we glue together the *first* pair we find that produce an equation with less than 2^{16} right hand sides. This gave significantly weaker results in all cases except for SR*$(3, 4, 4)$ which was solved after 105 guesses.

8.1 Random systems comparable to the AES systems

The results from the previous section are from scaled versions of the AES. For comparison, we tried the algorithm on random systems similar to the AES systems.

For real AES systems, we take advantage of the fact that the variables representing the user-selected key are special. If most of these variables are known, the rest of the system can be easily solved. If there is no such subset of special variables, how many variables do we need to guess?

We constructed random equations of size similar to the AES equations (7) as follows. We used the right hand sides of the AES equations as right hand sides for the random equations. The A-matrix in a random equation is a $16 \times n$-matrix with random linear independent rows, for various values of n. Each equation puts an 8-bit constraint on the solution space, so with $n/8$ equations we would expect the system to have about one solution, so we used systems with $n/8$ equations. To make sure there was always at least one solution, we also generated a random n-bit string to act as a solution to the system. The correct right hand side corresponding to this synthetic solution was computed for each equation, and replaced one of the AES right hand sides.

We ran algorithm G on these systems, still with the 2^{16}-limit on the number of right hand sides in one equation, to see how many guesses were needed to solve them. The results of these experiments are in the following table. As we

Table 2. The number of guesses before the system in n variables is solved.

n	48	56	64	80	96	112	128	160	192	224	256
# of guesses	0	7	13	29	45	61	76	108	140	172	204

see, the differences between the number of variables and number of guesses are mostly 51 or 52. It seems that with the limit of number of right hand sides set to 2^{16}, the techniques agreeing, gluing and extracting linear equations are able to solve a random looking non-linear equation system with 52 variables.

Magma is a computer algebra program that has an efficient implementation of the F4-algorithm [7] for computing Gröbner bases. This method is considered to be state-of-the-art when it comes to solving general non-linear equation systems. To compare our methods to F4, we also did the following experiments.

First we constructed a system of 6 random equations with 48 variables, with the AES right hand sides, that is as in (7). This time we did not insert any synthetic solution. We used gluing (max 2^{16} right hand sides), agreeing and extracting linear equations to solve the system. On the first two attempts we had constructed systems with no solutions, but on the third try the system had two solutions. These solutions were found in a few seconds. We then used the 24 linearly independent quadratic polynomials listed in [4] describing the AES S-box to generate the multivariate polynomials describing the same system, and imported them into Magma. Together with the field polynomials $x_i^2 + x_i$ we then had a set of $192 = 24 \times 6 + 48$ quadratic polynomials in 48 variables, and we called the function computing the Gröbner basis for this set. It returned the same two solutions after more than 15 hours of computing on the same machine with a 450MHz UltraSparc II processor.

Next we constructed a system of 7 random AES equations in 56 variables. This time we had to guess on the value of seven of the variables before we could solve the system with gluing and agreeing. We found the unique solution to the system, and the total time used for this was five and a half minutes. We then imported the corresponding set of quadratic polynomials into Magma and called its Gröbner basis function. This time Magma consumed all available memory, more than 3 GB, before it exited with an out-of-memory message.

9 Conclusion

We believe that the methods described in this paper should be put into the toolbox for algebraic cryptanalysis, together with the other algorithms for solving non-linear equation systems.

We also argue that the MRHS representation of equations is much better suited for equations made from S-boxes than multivariate polynomials and CNF formulas as in [1]. There are two reasons for this.

First, the input/output bits of an S-box are normally linear combinations of variables. When expressing the S-box as a set of polynomials, the products of these linear combinations must be expanded before we get the polynomials into ANF form. On the other hand, a lot of new variables should be introduced in order to construct k-SAT formulas with small k suitable for an efficient application of modern SAT solvers. Then these two methods heavily depend on the algebraic degree of the equations. The MRHS representation uses the fact that the input/output bits of the S-box are linear combinations, keeps them intact, and becomes a more compact representation. The method efficiency doesn't depend on the algebraic degree of the equations.

Second, representing an S-box using multivariate polynomials gives a whole *set* of polynomials, but they are all constructed from the same set of linear combinations, and really belong together as a group. This is exactly what is done in the MRHS representation, making one equation from one S-box.

Comparison with the findings in [2], together with our own experiments with Magma show that our methods are generally far stronger than what has been considered the best way to solve non-linear equation systems over $GF(2)$. It was also mentioned in [1] that Magma tends to be faster than SAT-solvers approach if it doesn't crash due to the lack of available memory. This seems to show that our methods overcome [1] too.

The critical reader may argue that the comparison is not fair, since the algorithm for solving equation systems in Magma is designed to handle *any* system, and that the systems that have been tested for both our program and Magma are special and constructed to suit our methods. This really only proves our point: The methods for solving non-linear equation systems presented in this paper are the best known for solving systems describing *ciphers*.

References

1. G. Bard, N. Courtois, C. Jefferson, *Efficient Methods for Conversion and Solution of Sparse Systems of Low-Degree Multivariate Polynomials over* $GF(2)$ *via SAT-Solvers*, Cryptology ePrint Archive, 2007/024, 25 January 2007.
2. C. Cid, S. Murphy, M. Robshaw, *Small Scale Variants of the AES*, in FSE 2005, LNCS 3557, pp. 145 – 162, Springer-Verlag, 2005.
3. N. Courtois, *The Security of Hidden Field Equations (HFE)*, in CT-RSA 2001, LNCS 2020, pp. 266 – 281, Springer-Verlag, 2001.
4. N. Courtois, J. Pieprzyk, *Cryptanalysis of Block Ciphers with Overdefined Systems of Equations* in Asiacrypt 2002, LNCS 2501, pp. 267 – 287, Springer-Verlag, 2002.
5. N. Courtois, W. Meier, *Algebraic Attacks on Stream Ciphers with Linear Feedback*, in Eurocrypt 2003, LNCS 2656, pp. 345 – 359, Springer-Verlag, 2003.
6. J. Daemen, V. Rijmen. *The Design of Rijndael; AES - The Advanced Encryption Standard,* Springer-Verlag, 2002.
7. J.-C. Faugère, *A new efficient algorithm for computing Gröbner bases (F4)*, Journal of Pure and Applied Algebra, vol. 139, pp. 61 – 88, 1999.
8. J.-C. Faugere, *A new efficient algorithm for computing Gröbner bases without reduction to zero (F5)*, Proceedings of ISSAC '02, pp. 75 – 83, ACM Press, 2002.
9. H. Raddum, I. Semaev, *New technique for solving sparse equation systems*, Ecrypt's STVL website, January 16th 2006, see also Cryptology ePrint Archive, 2006/475.
10. H. Raddum, I. Semaev, *Solving MRHS linear equations. Extended abstract* in Proceedings of WCC'07, 16-20 Avril 2007, Versailles, France, INRIA, 323–332.
11. A. Shamir, J. Patarin, N. Courtois, A. Klimov, *Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations*, in Eurocrypt 2000, LNCS 1807, pp. 392 – 407, Springer-Verlag 2000.
12. A. Zakrevskij, I. Vasilkova, *Reducing large systems of Boolean equations,* 4th Int.Workshop on Boolean Problems, Freiberg University, September, 21-22, 2000.