MRHS Equation Systems

Håvard Raddum

Selmersenteret, University of Bergen, Norway*

Abstract. We show how to represent a non-linear equation over $GF(2)$ using linear systems with multiple right hand sides. We argue that this representation is particularly useful for constructing equation systems describing ciphers using an S-box as the only means for non-linearity. Several techniques for solving systems of such equations were proposed in earlier work, and are also explained here. Results from experiments with DES are reported. Finally we use our representation to link a particular problem concerning vector spaces to the security of ciphers with S-boxes as the only non-linear operation.

Keywords: cryptanalysis, algebraic attacks, DES, non-linear equation systems.

1 Introduction

For the last years, most of the activity in cryptanalysis has been focused on algebraic attacks and solving non-linear equation systems. Several interesting properties and observations have been found and studied, and techniques for solving equation systems associated with a cipher have been proposed. So far there is no method for solving such systems which stands out as the "best" way to solve non-linear systems, the structure of the system plays a part. Moreover, it may be difficult to implement the ideas on a large system in practice, normal computers run out of memory too fast, see [1].

The traditional way of representing an equation has been by the use of a multivariate polynomial (MP) written in algebraic normal form (ANF). In [2] systems representing the block cipher DES are studied, and the authors propose to convert them to SAT-problems and use SAT-solvers. In this paper we will look at another way of representing non-linear equations, and with it follows new ways for solving systems of these equations. These methods were recently presented in [3], and earlier versions can also be found in [4] and [5].

We will use these techniques on systems representing the DES cipher for various rounds to see how they do in practice. As a side effect of our view on the equations we also discover a simply stated problem which makes a foundation of the security of a specific class of ciphers, in the same way as factoring is a fundamental problem for the security of RSA and finding discrete logarithms is a basis for the security of Diffie-Hellman key exchange.

* This work was done while visiting the Information Security Institute, Queensland Univeristy of Technology, Australia.

C. Adams, A. Miri, and M. Wiener (Eds.): SAC 2007, LNCS 4876, pp. 232–245, 2007.

The paper is organized as follows. In Section 2 we describe the way we represent equations, and show that systems coming from ciphers where the only non-linear part is the use of an S-box are easy to construct and are particularly suited for this representation. In Section 3 we describe some of the methods we have developed for solving our equation systems, and in Section 4 we try them on systems constructed from the DES cipher. In Section 5 we describe a problem, which must be hard to solve in order for the AES (among other ciphers) to be secure. Conclusions are made in Section 6.

2 MRHS Equation Systems

All variables in our equations will be over $GF(2)$. In most of the literature on algebraic cryptanalysis a non-linear equation over $GF(2)$ is represented as a MP $f(x_1, \ldots, x_n) = 0$. The set $V(f) = \{(x_1, \ldots, x_n) | f(x_1, \ldots, x_n) = 0\}$ is the set of satisfying assignments of f, and is what really defines the constraints the equation puts on the solution. Instead of the representation using MP, we will write a linear system with Multiple Right Hand Sides (MRHS) to describe the constraints:

$$A\mathbf{x} = [b_1, \ldots, b_s], \tag{1}$$

where A is a $(k \times n)$-matrix of full rank and the b_i's are vectors of length k over $GF(2)$. A vector $\mathbf{x}$ satisfies (1) if $A\mathbf{x} = b_i$ for some i. For shorter notation we will usually write an equation as $A\mathbf{x} = [B]$, where B is a matrix with the b_i's as columns. We keep square brackets around B to underline that the equation is not to be understood as a normal matrix/vector product where B is the product $A\mathbf{x}$, but rather that $A\mathbf{x}$ can be any column of B.

2.1 MRHS Equations vs. MP Equations

It is rather straight-forward to map between polynomial equations $f(\mathbf{x}) = 0$ and MRHS equations $A\mathbf{x} = [B]$. Given a MRHS equation E we may construct the set V of points in $GF(2)^n$ that satisfy E. This can be done by getting the solutions to the ordinary linear system $A\mathbf{x} = b_i$ and take the union of these solutions for $i = 1, \ldots, s$ as V. Then we may use a method like Lagrange interpolation to construct an f with $V(f) = V$. Both f and E will then give the same constraint on the solution space.

Conversely, given $f(\mathbf{x}) = 0$, we may compute $V(f) = \{b_1, \ldots, b_s\}$, and create the MRHS equation $I_n\mathbf{x} = [b_1, \ldots, b_s]$. As we will see, this way of creating a MRHS equation is not optimal, we should take advantage of any linearity inherent in f.

We show this with a small example. Suppose we are given the polynomial equation

$$f(x_1, x_2, x_3, x_4, x_5) = x_1x_2 + x_1x_5 + x_2x_3 + x_3x_5 + x_4 = 0. \tag{2}$$

Writing the MRHS equation as $I_5\mathbf{x} = [b_1, \ldots, b_s]$ we get $s = 16$ possible right hand sides. However, if we notice that (2) can be factored as $(x_1 + x_3)(x_2 + x_5) + x_4 = 0$ we can set up the MRHS equation

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

which gives the same constraint, but only has four possible right hand sides.

2.2 MRHS Equations from Ciphers

We will now show that MRHS representation of equations is very well suited for algebraic cryptanalysis of ciphers where the only source of non-linearity is the use of S-boxes. We call this class of ciphers *S-box based ciphers*. This class contains the most important block cipher, the AES [7], as well as several other well known ciphers, like DES [8], Serpent [9], Noekeon [10], etc.

Suppose we are looking at the AES and want to construct a system of equations describing the cipher. In order to keep equations small enough to handle we need to introduce variables in each round. Let us say the bits in the cipher block right after one application of SubBytes are $x_1, \ldots, x_{128}$ and that the bits in the cipher block after the next application of SubBytes are $y_1, \ldots, y_{128}$. Look at the first of the S-boxes used between $\mathbf{x}$ and $\mathbf{y}$. The bits input to this S-box will be $l_1(\mathbf{x})+k_1, \ldots, l_8(\mathbf{x})+k_8$, where k_i are the first eight bits of the round key used in this round, and the l_i are linear combinations using 32 of the $\mathbf{x}$-variables coming from ShiftRows and MixColumns. The bits at the output of this S-box will be $y_1, \ldots, y_8$. We can now set up the MRHS equation for this S-box as

$$\begin{bmatrix} l_1(\mathbf{x}) + k_1 \\ \vdots \\ l_8(\mathbf{x}) + k_8 \\ y_1 \\ \vdots \\ y_8 \end{bmatrix} = \begin{bmatrix} 0 & 1 & \ldots & 255 \\ S(0) & S(1) & \ldots & S(255) \end{bmatrix}, \tag{3}$$

where both i and $S(i)$ are written as 8-bit vectors. This equation has the 256 possible input/output combinations of the S-box as right hand sides and is a compact representation of the constraints imposed by the S-box when linking the $\mathbf{x}$ and $\mathbf{y}$ variables.

To construct MRHS equations in a general S-box based cipher, we will introduce variables between applications of S-boxes, so that the input and output bits of each S-box are linear combinations of variables and constants. The matrix A will have these linear combinations as rows and the columns of B will be the possible input/output combinations for the S-box. Setting up an MRHS equation system describing a complete S-box based cipher is then easily done by constructing one equation for each S-box used in the cipher.

Note that apart from linear combinations of variables, there is no need to compute MPs. The MRHS equations are constructed directly from the cipher

specifications. The size of the system depends only on the size and the number of S-boxes, and not on the degree of MPs defining the S-box or on diffusion properties of the linear operations in the cipher.

3 Techniques for Solving MRHS Equation Systems

In this section we will explain some techniques for solving an MRHS equation system. To make the presentation of this simpler, we first make a note on the number of columns in the A-matrices appearing in the MRHS equations in the system.

When setting up an MRHS equation system

$$A_1\mathbf{x} = [B_1], \ldots, A_m\mathbf{x} = [B_m], \tag{4}$$

it is usually the case that the total number of variables in the system is quite large, but that any individual equation only involves a small subset of those variables. When writing an equation $A_i\mathbf{x} = [B_i]$, it is always assumed that $\mathbf{x} = (x_1, \ldots, x_n)^T$, where n is the **total** number of variables in the system. This is done by inserting $\mathbf{0}$-columns in A_i for variables not occuring in the equation, such that the A_i-matrices all have exactly n columns.

A solution to (4) will be an $\mathbf{x}$-vector such that the product $A_i\mathbf{x}$ is a column found in B_i, for all $i = 1, \ldots, m$. One can say that the solution picks out the correct right hand side in each B_i, and that the other columns in B_i are wrong. The main strategy we use for trying to solve a system like (4) is to identify columns in B_i which can not be the correct right hand side for a solution, and delete them. If we are able to delete all wrong columns from each B_i we will be left with an ordinary system of linear equations (with only one right hand side), which can be easily solved.

The methods described below were all presented in [3], but we repeat them here for completeness since this work is quite recent and not well known.

3.1 Agreeing

This is the core method we use for finding right hand sides in equations that can not possibly be correct. This is done by looking for inconsistencies in a pair of equations and is done as follows. Let the equations $A_i\mathbf{x} = [B_i]$ and $A_j\mathbf{x} = [B_j]$ be given. By concatenating A_i and A_j on top of each other and expanding the columns in B_i and B_j with zeros we get the following identity:

$$\begin{bmatrix} A_i \\ A_j \end{bmatrix} \mathbf{x} = \begin{bmatrix} B_i \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ B_j \end{bmatrix}$$

The possible right hand sides for the concatenated equation are made by picking one column from $\begin{bmatrix} B_i \\ 0 \end{bmatrix}$ and one column from $\begin{bmatrix} 0 \\ B_j \end{bmatrix}$ and adding them. Compute U such that $C = U \begin{bmatrix} A_i \\ A_j \end{bmatrix}$ is upper triangular, and let $T_i = U \begin{bmatrix} B_i \\ 0 \end{bmatrix}$ and $T_j =$

$U \begin{bmatrix} 0 \\ B_j \end{bmatrix}$. By multiplying through with U we get the equation $C\mathbf{x} = [T_i] + [T_j]$. If $\begin{bmatrix} A_i \\ A_j \end{bmatrix}$ does not have full rank the last $r > 0$ rows of C will be all-zero. We only proceed if this is the case. Since the last r rows of C are all-zero, only columns from T_i and T_j that are equal in the last r coordinates can be added to make a possible right hand side for $C\mathbf{x}$, other choices would lead to an inconsistent system. Let Pr_i and Pr_j be the projection of T_i and T_j onto the last r coordinates, repectively. Any column in T_i whose projection is not found in Pr_j can not come from the correct column in B_i since adding it to any column of T_j will produce a right hand side inconsistent with $C\mathbf{x}$. The same applies to T_j, so all columns in T_i and T_j whose projections are not in $Pr_i \cap Pr_j$ will always create an inconsistency and the corresponding columns in B_i and B_j are wrong and can be deleted. An example of agreeing two MRHS equations can be found in the Appendix.

Agreeing pairs of equations may cause a domino effect of deletions of right hand sides. For example, say that no deletions occur when agreeing equations E_1 and E_2, but that deletions occur in E_2 when agreeing it with E_3. These deletions may cause E_1 and E_2 to disagree, so deletions will now occur in E_1 when agreeing it with E_2. These deletions may again trigger deletions in other equations, and so on. We run agreeing on every pair of equations in the system until no more deletions occur and all pairs of equations agree. We call this process the *agreeing algorithm*.

3.2 Extracting Linear Equations

The agreeing algorithm itself is normally not strong enough to solve a system of MRHS equations. When all pairs of equations agree but there still are many wrong right hand sides in the equations we may check to see if it is possible to squeeze ordinary linear equations out of them. This method is applied to equations individually and is done as follows.

Consider the equation $A\mathbf{x} = [B]$. Compute U, such that UB is upper triangular and transform the equation to $UA\mathbf{x} = [UB]$. Assume the last $r > 0$ rows of UB are all-zero, and let the r last rows of $UA\mathbf{x}$ be $l_1, \ldots, l_r$. All columns in B have 0 in the last r coordinates, so the correct column in particular have 0 in the last r coordinates. We then know the r linear equations $l_1 = 0, \ldots, l_r = 0$ must be true.

It may also be possible to make one more linear equation from the MRHS equation. We check if the all-one vector is found in the space spanned by the rows of B. This is easy to do, we may add the all-one vector to B and see if the rank of the resulting matrix increases by one, and can be done even faster when we have a basis for the row space of B in triangular form, as in UB. If we find that $\mathbf{v}B = \mathbf{1}$ for some $\mathbf{v}$, we know with certainty that the linear equation $\mathbf{v}A\mathbf{x} = 1$ is true. An example of extracting linear equations is found in the Appendix.

The linear equations produced this way can be used to eliminate variables in the system. As variables are eliminated, the rank of some of the A-matrices in the equations may not be full anymore. When this happens, we can find a vector $\mathbf{v} \neq \mathbf{0}$ such that $\mathbf{v}A = \mathbf{0}$, and we can compute $\mathbf{u} = \mathbf{v}B$. Columns in B found in positions where $\mathbf{u}$ has a 1-bit would lead to an inconsistent system, and can safely be removed as wrong columns. In this way, finding linear equations will also help to identify wrong columns and bring a solution to the system closer to the surface.

3.3 Gluing

When the agreeing algorithm works, it is because the spaces spanned by the rows of the A-matrices in some equations overlap in non-trivial common subspaces. When the agreeing algorithm stops, and no more linear equations can be extracted, we may try merging several equations into one. The spaces spanned by the A-matrices of the resulting equations will be larger, and hopefully have more overlap among them, so some new disagreements can be created. When merging two equations we say we *glue* them together.

When gluing two MRHS equations $A_i\mathbf{x} = [B_i]$ and $A_j\mathbf{x} = [B_j]$ together into a new equation $A\mathbf{x} = [B]$, much of the same steps as with agreeing are taken. The matrices A_i and A_j are concatenated, and the matrix C and the two expanded sets of right hand sides T_i and T_j are computed as explained under agreeing. Assume the last r rows of C are all-zero rows. We now create the columns in B by xoring every pair of one column from T_i and one column from T_j that are equal in the last r coordinates. The last r (all-zero) coordinates of the sum should be removed. The matrix A of the glued equation will then be C with the last r all-zero rows removed. The two equations we started with are now redundant and can be removed since all information contained in them is kept in the glued equation. Gluing reduces the number of equations in the system.

Let us say that the number of right hand sides in the equations B_i and B_j are s_i and s_j, respectively. Let $\mathbf{u} = (u_0, \ldots, u_{2^r-1})$ be a vector of integers where u_k is the number of columns in T_i that have the binary representation of k in the last r coordinates. Let $\mathbf{v} = (v_0, \ldots, v_{2^r-1})$ be the same kind of vector for T_j. The number of right hand sides in B will be the inner product $\mathbf{u} \cdot \mathbf{v} = \sum_{k=0}^{2^r-1} u_k v_k$. In general, the number of right hand sides in B will be much larger than $s_i + s_j$, and in the case $r = 0$ it will simply be $s_i s_j$.

Assume we have three MRHS equations E_1, E_2, E_3 that all pairwise agree. If we glue E_1 and E_2 into E it may be the case that E and E_3 do not agree, and that right hand sides will be removed from E_3 when agreeing it with E. What we are really doing is searching for inconsistencies across all three initial equations, and not only two as with ordinary agreeing. When gluing several equations together we will increase the probability of creating disagreements, which again will reduce the number of right hand sides.

In fact, if we could glue all equations in a system into one big MRHS equation, we would actually solve the system. What prevents us from doing that in practice is the fact that the number of right hand sides in a glued equation is (much)

bigger than the number of right hand sides in the two equations we started with, assuming they agreed. In practice we have to set a limit on the number of right hand sides we are capable of storing in one MRHS equation. For the system to be solvable by gluing it is then necessary that enough disagreements occur after intermediate gluings so this threshold is never passed.

We will return to the scenario of gluing all equations in a system in Section 5.

3.4 A Complete Algorithm for Solving a System of MRHS Equations

When all pairs of equations agree, no linear equations can be extracted and we can not afford to glue any equations together, the last resort is to guess on the value of one variable, or a sum of variables. Guessing on the sum of some variables has the same effect as taking a linear equation and eliminating a variable with it. The sum of variables to be guessed could be one of the vectors occurring in the span of the rows of some A-matrices, in order to make sure some deletions of right hand sides occur. In the case for systems constructed from ciphers, it is a good idea to guess on the value of some of the user-selected key bits since these variables are special. If the value of the key bits are substituted into the system it will collapse; the rest of the system will be solved by simple agreeing alone.

We present here an algorithm where we try to find how few bits of information we need to guess in order to solve a system of MRHS equations. We call this the FewGuess algorithm, and the idea is to only guess one bit of information when all else has been tried. The maximum number of right hand sides we will handle in one equation is S. After each step in the algorithm we check if the system has been solved or become inconsistent, and exit with the number of guesses made if it has.

FewGuess

1. Run the agreeing algorithm.
2. Try to extract linear equations. If any linear equations were extracted, eliminate variables and go back to 1.
3. Glue together any pair of equations where the number of right hand sides in the glued equation is $\leq S$. If any gluings occurred go back to 1.
4. Guess on the value of a linear combination of variables, eliminate one variable and go back to 1.

This algorithm is designed to find how few bits we need to guess in order to determine whether a guess was right or wrong, and is not very efficient in terms of running time. After running this algorithm we know a set of linear combinations of variables to be guessed, and we know in which order equations were glued together. To set up a key recovery attack, we should first make all the guesses at once. Then we should glue together equations in the order given by our algorithm, and possibly run agreeing and extract linear equations first if some gluings will break the limit S.

If we need to guess b bits of information the complexity for solving a system will be of the order 2^b, multiplied with a constant that depends on S. This constant will be the complexity of running the agreeing algorithm, extracting linear equations and gluing together equations. These are not trivial operations, and implementing them in the most efficient way is a difficult task in itself. We will not investigate the complexities and the implementation issues of these operations here, but rather focus on the number of bits of information needed to guess in order to solve a system.

4 Experiments with DES

Inspired by the work in [2], we have constructed the MRHS equation system representing DES for various number of rounds, and tested the methods for solving described in the previous section. We assume the reader is familiar with the basic structure of DES, we repeat here the features that are most important for the construction of the MRHS equation system.

4.1 Constructing Equations

DES is a Feistel network, with a round function that takes a 32-bit input and a 48-bit round key to compute a 32-bit output. The bits of the round keys in DES are selected directly from the 56 user-selected key bits, so we need only 56 variables to represent the round keys. The only non-linear operation in DES is the use of the eight S-boxes in each round. We construct the equations as explained in Section 2.2, so we need that the bits at the input and output of each S-box can be written as a linear combination of variables and constants. We give variable names to the bits going into the round function in each round, except for the first and last rounds. The inputs of the first and last rounds are parts of the plaintext and ciphertext, considered constants in a cryptanalytic attack. The input and output bits of all S-boxes can then be expressed as linear combinations of variables and constants.

The number of variables in a system representing an r-round version of DES will be $56 + 32(r - 2)$. Each S-box gives one MRHS equation, so the system will consist of $8r$ equations. The S-boxes used in DES all take 6-bit inputs and produce 4-bit outputs. The A-matrix in each equation will thus consist of 10 rows, and the B-matrix will have 64 columns, one for each possible S-box input.

4.2 Results

We have tried the FewGuess algorithm on systems representing DES with a various number of rounds to see how few bits that were needed to guess in order to solve a system. The 56 key-variables are special, once these are determined the values of the other variables will be given by the system straight away. It is therefore natural to guess on the key variables when we need to guess since we know that at most 56 guesses are needed.

Table 1. Number of guesses needed to solve DES systems

# of rounds	guessing most used	guessing $k_1, k_2, \ldots$
4	3	3
5	26	17
6	34	28
7	38	38
8	38	38
10	38	38
12	38	38
16	41	38

Table 2. Number of key variables needed to guess for various limits S

S	# of guesses
2^8	48
2^{12}	45
2^{16}	41
2^{20}	36

The order of the variables to be guessed also plays a part. Since some key bits appear in round keys more often than others in DES, we tried the strategy of guessing on the most used key variables first, in the hope that eliminating these variables would create a bigger impact on the system. We also tried guessing the key bits in the order $1, 2, 3, \ldots$ to see if there was a difference. The maximum number of right hand sides we would allow in one equation was set to $S = 2^{18}$. The results were as follows.

As can be seen, the order in which the variables are guessed makes a difference, and it is not the greedy approach of guessing the most used variables first that is most efficient. A reason for this can maybe be found in the key schedule of DES. The key schedule is designed such that the key bits occuring in the inputs of the S-boxes S_1 - S_4 are all taken from $k_1, \ldots, k_{28}$. When these key bits are guessed, the inputs to S_1 - S_4 in the first and last round will be known. This will immediately give the value of 32 of the variables entering the second and the second to last rounds. When guessing on the most used key variables the guesses will be spread out over all 56 key variables and more guesses are needed before the values of all key variables entering one S-box are known.

Of course, the number of guesses increases with the number of rounds and it is necessary to guess 38 key variables to break seven rounds of DES this way. However, the number of guesses to break more rounds does not increase from 38, at least when guessing in the increasing order. It is natural to believe the number 38 is linked to the limit $S = 2^{18}$ since $18 + 38 = 56$, the number of user-selected key variables.

This suggests that 18 bits are guessed "implicitly" when storing up to 2^{18} right hand sides in the equations and that there is a number-of-guesses/memory tradeoff. The hypothesis is that setting the limit to $S = 2^l$ means we do not

have to guess more than $56 - l$ key variables. We tried FewGuess on the system representing 16-round DES using other values for S and guessing the key bits in the order $1, 2, \ldots$ to check this hypothesis.

These few tests show the hypothesis is not exactly true, but almost. The order in which the variables are guessed plays a part.

5 The Security of S-Box Based Ciphers

In this section we will use the MRHS representation of equations to find an easily stated problem about vector spaces. This problem is independent of the structure of a particular S-box based cipher, and it has to be hard to solve if ciphers in this class are to remain secure.

Assume we are given an S-box based cipher and construct its MRHS equation system using a total of n variables. Let the number of input bits to the S-box(es) used be p, let the number of S-boxes used to process one encryption be q and let the number of output bits of the S-box be $k - p$. The MRHS equation system we get will be

$$A_1\mathbf{x} = [B_1], \ldots, A_q\mathbf{x} = [B_q], \tag{5}$$

where each A_i is a $k \times n$-matrix and each B_i is a $k \times 2^p$-matrix. Let us set this system up as if we are going to glue all equations into one big MRHS equation $A\mathbf{x} = [B]$ in one operation:

$$\begin{bmatrix} A_1 \\ \hline A_2 \\ \hline \vdots \\ \hline A_q \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{bmatrix} B_1 \\ \hline \mathbf{0} \\ \hline \vdots \\ \hline \mathbf{0} \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ \hline B_2 \\ \hline \vdots \\ \hline \mathbf{0} \end{bmatrix} + \ldots + \begin{bmatrix} \mathbf{0} \\ \hline \vdots \\ \hline \mathbf{0} \\ \hline B_q \end{bmatrix} \tag{6}$$

Using linear algebra we compute U such that U multiplied by the matrix on the left hand side of (6) is upper triangular. As in Section 3.3, let us call the resulting matrix C and let U multiplied with the bracket containing B_i be T_i. Multiplying through with U gives us

$$C\mathbf{x} = [T_1] + [T_2] + \ldots + [T_q], \tag{7}$$

where we are supposed to select exactly one column from each T_i and add them together to create a possible right hand side for $C\mathbf{x}$.

The matrix C has qk rows and n columns, and there are no linear relations among the variables (if there were, we would eliminate some variables first). Hence the last $qk - n$ rows of C are all-zero rows and put the constraint on the selection of the columns from the T_i's that their sum must be zero in the last $qk - n$ coordinates. If we can find such a selection of columns from the T_i's we get a consistent linear system with a unique right hand side, and solving this we get the solution to (5).

Let $Pr(\cdot)$ be the projection onto the last $qk-n$ coordinates and let $Z_i = (Pr(T_i))^T$. Concatenate the matrices Z_i like in the left hand side of (6) and call the resulting matrix Z.

$$Z = \begin{bmatrix} Z_1 \\ \dots \\ Z_q \end{bmatrix}$$

The number of rows in Z is $2^p q$ and the number of columns is $qk-n$. Our problem has been transformed to picking exactly one row from each Z_i such that they add up to $\mathbf{0}$. Similarly to the computation of U, we can compute a $(2^p q \times 2^p q)$-matrix M such that MZ is upper triangular. Let the matrix formed by the last $2^p q - qk + n$ rows of M be called M_0. Then any $\mathbf{v}$ from the row space of M_0 will give $\mathbf{v}Z = \mathbf{0}$.

Definition 1. *We say that a binary vector* $\mathbf{v} = (\mathbf{v}_1, \dots, \mathbf{v}_q)$ *where* $|\mathbf{v}_i| = 2^p$ *for some* q *and* p *has the q1-property if the Hamming weight of each* $\mathbf{v}_i$ *is one.*

We can now state the problem as follows.

A fundamental problem for S-box based ciphers. *Given a binary matrix* M_0 *with* $2^p q$ *columns for some* p *and* q*. If there are vectors in the row space of* M_0 *with the q1-property, find one.*

If we can solve this problem, the 1-bits in the found vector will indicate exactly which right hand sides that can be added together in (7) to form a consistent linear system. This would solve (5) and break any S-box based cipher, hence it must be a hard problem if these ciphers are to remain secure. On the other hand, if we can break an S-box based cipher (find its key given some plaintext/ciphertext pairs) we find the solution to the corresponding MRHS equation system and the positions of the correct right hand sides in each equation. Setting a 1-bit in these positions will give us a vector with the q1-property which is found in the rowspace of the associated M_0. This shows that solving the fundamental problem corresponding to an S-box based cipher is equivalent to breaking the cipher.

We do not propose any ideas for efficiently solving the fundamental problem here, but instead we take a look at the actual values of p, q, k and n for DES and AES to briefly see what the problem will look like in these specific instances.

For the full 16 round DES, we get $p = 6, q = 128, k = 10$ and $n = 56+14\cdot 32 = 504$. The matrix M_0 will in this instance be a 7416×8192-matrix. For the full AES with 128-bit key we get $p = 8, q = 200, k = 16$ and $n = 1600$. This gives a M_0 with 49600 rows and 51200 columns. In both cases we see that the number of rows in M_0 is so much larger than the number of bits in the user-selected key that any algorithm solving the fundamental problem must be polynomial (or very close to polynomial) in the number of rows of M_0 to give an efficient attack.

6 Conclusions

The purpose of this paper is to show that the MRHS representation of non-linear equations should be taken into consideration when discussing algebraic attacks.

The representation of equations does matter. In [2] the authors do algebraic cryptanalysis of reduced-round DES, comparing their own technique to those implemented in software packages like MAGMA and Singular. The representation used for the equations is MPs, and the results are not as good as when they convert the equation system into CNF form and use a SAT-solver. There are some important differences between the MRHS representation and the MP representation.

First, the size of an MRHS equation is independent of the linear operations taking place between the use of two S-boxes. We may describe an S-box using a set of MPs in $\mathbf{x}$- (input) and $\mathbf{y}$- (output) variables. When the $\mathbf{x}$ and $\mathbf{y}$ are linear combinations of variables the size of these polynomials in ANF form will be very dependent on the number of variables in each linear combination.

Second, the degree of the MPs representing an S-box plays a crucial role for the complexity of solving a MP equation system. For MRHS equations this degree is irrelevant, the complexity of using the techniques described in this paper does not depend on it.

Third, there are fewer equations in an MRHS equation system than in a system of MP equations representing a cipher. One S-box gives rise to one MRHS equation, while there are a number of MP equations associated with one S-box. The strategies taken for solving these systems are also different in nature. When using MP representation we usually want to create *more* equations, to be able to solve by re-lienarization or find a Gröbner basis. This consumes a lot of memory in implementations. When using MRHS representation we want to *reduce* the number of equations by gluing, and to remove right hand sides.

Fourth, we are not aware of any method for finding and extracting all linear equations that might implicitly be hiding in a non-linear MP equation. The method described in Section 3.2 allows us to efficiently do this for an MRHS equation.

Using the MRHS representation also allowed us to derive a problem about finding a vector with a special property in a given vector space and show that solving this problem is equivalent to breaking any S-box based cipher. A lot of effort has been spent on the problems of factoring and discrete logarithms for assessing the security of several primitives in public key cryptography. It is reasonable to look more closely on the fundamental problem of S-box based ciphers stated in this paper, since the security of the AES depends on it.

We think that the MRHS representation of equations is better suited than MPs for systems representing an S-box based cipher. As traditional methods for solving MP equation systems tend to run out of memory, even on rather small systems, MRHS equation systems representing full ciphers can be constructed, and worked with. The MRHS representation of equations should go into the toolbox for algebraic cryptanalysis.

References

1. Cid, C., Murphy, S., Robshaw, M.: Small Scale Variants of the AES. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 145–162. Springer, Heidelberg (2005)
2. Courtois, N., Bard, G.: Algebraic Cryptanalysis of the Data Encryption Standard, Cryptology ePrint Archive, Report 2006/402 (2006), http://eprint.iacr.org/
3. Raddum, H., Semaev, I.: Solving MRHS linear equations. Extended abstract, In: International Workshop on Coding and Cryptography, April 16-20, 2007, Versailles, France (2007)
4. Raddum, H., Semaev, I.: New Technique for Solving Sparse Equation Systems, Cryptology ePrint Archive, Report 2006/475 (2006), http://eprint.iacr.org/
5. Raddum, H.: Cryptanalytic Results on Trivium (2006), available from http://www.ecrypt.eu.org/stream/triviump3.html
6. Courtois, N., Pieprzyk, J.: Cryptanalysis of Block Ciphers with Overdefined Systems of Equations, Cryptology ePrint Archive, Report 2002/044 (2002), http://eprint.iacr.org/
7. Rijmen, V., Daemen, J.: The Block Cipher Rijndael. Springer, Berlin (2002)
8. US National Bureau of Standards. Data Encryption Standard, Federal Information Processing Standards Publications No. 46 (1977)
9. Anderson, R., Biham, E., Knudsen, L.: Serpent: A Proposal for the Advanced Encryption Standard (1998), available from http://www.cl.cam.ac.uk/~rja14/serpent.html
10. Daemen, J., Peeters, M., Van Assche, G., Rijmen, V.: Nessie Proposal: Noekeon (2000), available from http://gro.noekeon.org/

A Example of Agreeing and Extracting Linear Equations

A.1 Agreeing Example

We want to agree the following two equations:

$$\overset{A_1}{\begin{bmatrix}1&1&0&0&0\\0&1&0&1&0\\0&0&1&1&0\end{bmatrix}}\begin{pmatrix}x_1\\x_2\\x_3\\x_4\\x_5\end{pmatrix}=\overset{B_1}{\begin{bmatrix}0&0&1&1\\0&1&0&1\\1&1&1&0\end{bmatrix}} \qquad \overset{A_2}{\begin{bmatrix}0&0&1&0&1\\0&1&0&1&0\\0&1&1&0&0\end{bmatrix}}\begin{pmatrix}x_1\\x_2\\x_3\\x_4\\x_5\end{pmatrix}=\overset{B_2}{\begin{bmatrix}0&0&1&1\\0&1&0&1\\0&0&0&1\end{bmatrix}}$$

We compute U to make $\begin{bmatrix}A_1\\A_2\end{bmatrix}$ triangular and find $T_1 = U\begin{bmatrix}B_1\\0\end{bmatrix}$ and $T_2 = U\begin{bmatrix}0\\B_2\end{bmatrix}$.

$$U=\begin{bmatrix}1&0&0&0&0&0\\0&1&0&0&0&0\\0&0&1&0&0&0\\0&0&1&1&0&0\\0&1&0&0&1&0\\0&1&1&0&0&1\end{bmatrix} \qquad C=U\begin{bmatrix}A_1\\A_2\end{bmatrix}=\begin{bmatrix}1&1&0&0&0\\0&1&0&1&0\\0&0&1&1&0\\0&0&0&1&1\\0&0&0&0&0\\0&0&0&0&0\end{bmatrix},$$

$$T_1 = \begin{bmatrix} 0\,0\,1\,1 \\ 0\,1\,0\,1 \\ 1\,1\,1\,0 \\ 1\,1\,1\,0 \\ \mathbf{0\,1\,0\,1} \\ \mathbf{1\,0\,1\,1} \end{bmatrix} \qquad T_2 = \begin{bmatrix} 0\,0\,0\,0 \\ 0\,0\,0\,0 \\ 0\,0\,0\,0 \\ 0\,0\,1\,1 \\ \mathbf{0\,1\,0\,1} \\ \mathbf{0\,0\,0\,1} \end{bmatrix}.$$

We see that $r = 2$, and that $Pr_1 \cap Pr_2 = \{\binom{1}{0}, \binom{1}{1}\}$. The projections of the first and third columns from both T_1 and T_2 fall outside the intersection, hence the first and third columns from both B_1 and B_2 have been identified as wrong. After agreeing the equations are

$$\overset{A_1}{\begin{bmatrix} 1\,1\,0\,0\,0 \\ 0\,1\,0\,1\,0 \\ 0\,0\,1\,1\,0 \end{bmatrix}} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \overset{B_1}{\begin{bmatrix} 0\,1 \\ 1\,1 \\ 1\,0 \end{bmatrix}} \qquad \overset{A_2}{\begin{bmatrix} 0\,0\,1\,0\,1 \\ 0\,1\,0\,1\,0 \\ 0\,1\,1\,0\,0 \end{bmatrix}} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \overset{B_2}{\begin{bmatrix} 0\,1 \\ 1\,1 \\ 0\,1 \end{bmatrix}}.$$

A.2 Example of Extracting Linear Equations

We take the equation $A_1\mathbf{x} = [B_1]$, and extract linear equations from it. First we compute U to make B upper triangular, and multiply through to arrive at the following MRHS equation

$$\overset{UA_1}{\begin{bmatrix} 0\,0\,1\,1\,0 \\ 1\,1\,0\,0\,0 \\ 1\,0\,1\,0\,0 \end{bmatrix}} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \overset{UB_1}{\begin{bmatrix} 1\,0 \\ 0\,1 \\ 0\,0 \end{bmatrix}}$$

From the bottom row we get the linear equation $x_1 + x_3 = 0$. Adding the two top rows will create the **1**-vector in UB_1, hence by adding the two top rows from UA_1 we also get the linear equation $x_1 + x_2 + x_3 + x_4 = 1$.