MXL2: Solving Polynomial Equations over GF(2) Using an Improved Mutant Strategy

Mohamed Saied Emam Mohamed[1], Wael Said Abd Elmageed Mohamed[1], Jintai Ding[2], and Johannes Buchmann[1]

[1] TU Darmstadt, FB Informatik, Hochschulstrasse 10, 64289 Darmstadt, Germany
{mohamed,wael,buchmann}@cdc.informatik.tu-darmstadt.de
[2] Department of Mathematical Sciences, University of Cincinnati, Cincinnati OH 45220, USA
jintai.ding@uc.edu

Abstract. MutantXL is an algorithm for solving systems of polynomial equations that was proposed at SCC 2008. This paper proposes two substantial improvements to this algorithm over GF(2) that result in significantly reduced memory usage. We present experimental results comparing *MXL2* to the XL algorithm, the MutantXL algorithm and Magma's implementation of F_4. For this comparison we have chosen small, randomly generated instances of the MQ problem and quadratic systems derived from HFE instances. In both cases, the largest matrices produced by *MXL2* are substantially smaller than the ones produced by MutantXL and XL. Moreover, for a significant number of cases we even see a reduction of the size of the largest matrix when we compare *MXL2* against Magma's F_4 implementation.

1 Introduction

Solving systems of multivariate quadratic equations is an important problem in cryptology. The problem of solving such systems over finite fields is called the Multivariate Quadratic (MQ) problem. In the last two decades, several cryptosystems based on the MQ problem have been proposed as in [1,2,3,4,5]. For generic instances it is proven that the MQ problem is NP-complete [6]. However for some cryptographic schemes the problem of solving the corresponding MQ system has been demonstrated to be easier, allowing these schemes to be broken. Therefore it is very important to develop efficient algorithms to solve MQ systems.

Recently, MutantXL [7] and MutantF4 [8] were proposed at SCC 2008, two algorithms based on Ding's mutant concept. Roughly speaking, in algorithms that operate on linearized representations of the polynomial system by increasing degree – such as F_4 and XL – this concept proposes to maximize the effect of lower-degree polynomials occurring during the computation. In this paper, we present MutantXL2 (*MXL2*) – a new algorithm based on MutantXL that oftentimes allows to solve systems with significantly smaller matrix sizes than

J. Buchmann and J. Ding (Eds.): PQCrypto 2008, LNCS 5299, pp. 203–215, 2008.

XL and MutantXL. Moreover, experimental results for both HFE systems and random systems demonstrate that for a significant number of cases we even get a reduction of the size of the largest matrix when comparing *MXL2* against Magma's F_4 implementation.

The paper is organized as follows. In Section 2 the key ideas of the *MXL2* algorithm and the required definitions are presented. A formal description and explanations of the algorithm are in Section 3. Section 4 contains the experimental results. In Section 5 we conclude our paper.

2 Improvements to the Mutant Strategy

In this section we present the key ideas of the *MXL2* algorithm and explain their importance for solving systems of multivariate quadratic polynomial equations more efficiently. Throughout the paper we will use the following notations: Let $X := \{x_1, \ldots, x_n\}$ be a set of variables, upon which we impose the following order: $x_1 < x_2 < \ldots < x_n$. Let

$$R = \mathbb{F}_2[x_1, \ldots, x_n]/(x_1^2 - x_1, ..., x_n^2 - x_n)$$

be the ring of polynomial functions over $\mathbb{F}_2$ in X with the monomials of R ordered by the graded lexicographical order $<_{glex}$. By an abuse of notation, we call the elements of R polynomials throughout this paper. Let $P = (p_1, \ldots, p_m) \in R^m$ be a sequence of m quadratic polynomials in R. Throughout the operation of the algorithms described in this paper, a degree bound D will be used. This degree bound denotes the maximum degree of the polynomials contained in P. Note that the contents of P will be changed throughout the operation of the algorithm.

Some algorithms for solving the system

$$p_j(x_1, \ldots, x_n) = 0, 1 \leq j \leq m \tag{1}$$

such as XL and MutantXL are based on finding new elements in the ideal generated by the polynomials of P that correspond to equations that are easy to solve, i.e. univariate or linear polynomials. The MutantXL algorithm is an application of the mutant concept to the XL algorithm. The following definitions explain the term *mutant*:

Definition 1. *Let $g \in R$ be a polynomial in the ideal generated by the elements of P. Naturally, it can be written as*

$$g = \sum_{p \in P} g_p p \tag{2}$$

where $g_p \in R$, $p \in P$. The level of this representation is defined to be

$$\max\{\deg(g_p p) \ : \ p \in P\}.$$

Note that this level depends on P. The level *of the polynomial g is defined to be the minimum level of all of its representations.*

Definition 2. *Let $g \in R$ be a polynomial in the ideal generated by the elements of P. The polynomial g is called a* mutant *with respect to P if its degree is less than its level.*

Next, we explain the meaning of mutants. When a mutant is written as a linear combination (2), then one of the polynomials $g_p p$ has a degree exceeding the degree of the mutant. This means that a mutant of degree d cannot be found as a linear combination of polynomials of the form mp where m is a monomial, $p \in P$ and the degree of mp is at most d. However, such mutants could help in solving the system (1) if we can find them efficiently.

Given a degree bound D, the MutantXL algorithm extends the system of polynomial equations (1) by multiplying the polynomials on the left-hand side by all monomials up to degree $D - \deg(p_i)$. Then the system is linearized by considering the monomials as new variables and applying Gaussian elimination on the resulting linear system. MutantXL searches for univariate equations, if no such equations exist, it searches for mutants, that are new polynomials of degree $< D$. If mutants are found, they are multiplied by all monomials such that the produced polynomials have degree $\leq D$. Using this strategy, MutantXL achieves to enlarge the system without incrementing D.

In many experiments with MutantXL on some HFE systems and some randomly generated multivariate quadratic systems, we noticed that there are two problems. The first occurs when the number of lower degree mutants is very large, we observed this produces many reductions to zero. A second problem occurs when an iteration does not produce mutants at all or produces only an insufficient number of mutants to solve the system at lower degree D. In this case MutantXL behaves like XL.

Our proposed improvements handle both problems, while using the same linearization strategy as the original MutantXL. This allows us to compute the solution with fewer polynomials. To handle the first problem, we need the following notation.

Let $S_k : \{\ m \in R : deg(m) \leq k\}$ be the set of all monomials of R that have degree less than or equal to k. Combinatorially, the number of elements of this set can be computed as

$$|S_k| = \sum_{\ell=1}^{k} \binom{n}{\ell}, 1 \leq k \leq n \tag{3}$$

where n is the number of variables.

The *MXL2* algorithm as well as MutantXL are based on the mutant concept, however *MXL2* introduces a heuristic strategy of only choosing the minimum number of mutants, which will be called *necessary mutants*. Let k be the degree of the lowest-degree mutant occuring and the number of the linearly independent elements of degree $\leq k{+}1$ in P be $Q(k{+}1)$. Then the smallest number of mutants that are needed to generate $|S_{k+1}|$ linearly independent equations of degree $\leq k+1$ is

$$\lceil(|S_{k+1}| - Q(k+1))/n\rceil, \tag{4}$$

where S_{k+1} is as in (3) and n is the number of variables. Therefore by multiplying only the necessary number of mutants, the system can potentially be solved by a smaller number of polynomials and a minimum number of multiplication. This handles the first problem. In the following we explain how *MXL2* solves the second problem.

Suppose we have a system with not enough mutants. In this case we noticed that in the process of space enlargement, MutantXL multiply all original polynomials by all monomials of degree $D-2$. In most cases only a small number of extended polynomials that are produced are needed to solve the system. Moreover the system will be solved only when some of these elements are reduced to lower degree elements. To be more precise, the degree of the extended polynomials is decreased only if the higher degree terms are eliminated. We have found that by using a partitioned enlargement strategy and a successive multiplication of polynomials with variables method, while excluding redundant products, we can solve the system with a smaller number of equations. To discuss this idea in details we first need to define the following:

Definition 3. *The* leading variable *of a polynomial p in R is x, if x is the smallest variable, according to the order defined on the variables, in the leading term of p. It can be written as*

$$\mathrm{LV}(p) = x \tag{5}$$

Definition 4. *Let $P_k = \{p \in P\colon deg(p) = k\}$ and $x \in X$. We define P_k^x as follows*

$$P_k^x = \{p \in P_k : LV(p) = x\} \tag{6}$$

In the process of space enlargement, *MXL2* deals with the polynomials of P_D differently. Let P_D be divided into a set of subsets depending on the leading variable of each polynomial in it. In other words, $P_D = \bigcup_{x \in X} P_D^x$, where X is the set of variables as defined previously and P_D^x as in (6). *MXL2* enlarges P by increments D and multiplies the elements of P_D as follows: Let x be the largest variable, according to the order defined on the variables, that has $P_D^x \neq \emptyset$. *MXL2* successively multiplies each polynomial of P_D^x by variables such that each variable is multiplied only once. This process is repeated for the next smaller variable x with $P_D^x \neq \emptyset$ until the solution is obtained, otherwise the system enlarges to the next D. Therefore *MXL2* may solve the system by enlarging only subsets of P_D, while MutantXL solves the system by enlarging all the elements of P_D. *MXL2* handles the second problem by using this partitioned enlargement strategy.

In the next section we describe *MXL2*. In section 4 we present examples that show that *MXL2* completely beats the first version of MutantXL and beats in most cases Magma's implementation of F_4 for only the memory efficiency.

3 *MXL2* Algorithm

In this Section we explain the *MXL2* algorithm. We use the notation of the previous section. So P is a finite set of polynomials in R. For simplicity, we assume that the system (1) is quadratic and has a unique solution.

We use a graded lexicographical ordering in the process of linearization and during the Gaussian elimination. *MXL2* creates a *multiplication history* one dimension array to store each previous variable multiplier of each polynomial and for the originals the previous multiplier is 1. The set of solutions of the system is defined as $\{x = b : x \text{ is variable and } b \in \{0, 1\}\}$. The description of the algorithm is as follows.

- *Initialization* Use Gaussian elimination to make P linearly independent. Set the set of *root polynomials* to $\emptyset$, the *total degree bound* D to 2, the *elimination degree* to D, *system extended* to false, *mutants* to $\emptyset$, and *multiplication history* to a one dimension array with number of elements as P and initialize these elements by ones (Algorithm 1 lines 16 – 21).
- *Gauss* Use linearization to transform the set of all polynomials in P of degree $\leq$ *elimination degree* into reduced row echelon form (Algorithm 1 lines 23 and 24).
- *Extract Roots* copy all new polynomials of degree ≤ 2 to the root polynomials set (Algorithm 1 line 25).
- If there are univariate polynomials in the *roots*, then determine the values of the corresponding variables, and remove the solved variables from the variable set. If this solves the system return the solution and terminate. Otherwise, substitute the values for the variables in the *roots*, set P to the *roots*, set *elimination degree* to the maximum degree of the *roots*, reset the *multiplication history* to an array of number of elements as P and initialize these elements to ones, and go back to *Gauss* (Algorithm 1 lines 26 – 32).
- *Extract Mutants* copy all new polynomials of degree < D from $\{P\}$ to *mutants* (Algorithm 1 line 34).
- If there are mutants found, then extend the *multiplication history* by an array of the number of elements of the same length as the new polynomials initialized by ones, multiply the necessary number of mutants having the minimum degree, as stated in Section 2, by all variables, set the *multiplication history* for each new polynomial by its variable multiplier, include the resulting polynomials in P, set the *elimination degree* to that minimum degree + 1, and remove all multiplied mutants from *mutants* (Algorithm 2 lines 9 – 20).
- Otherwise, if *system extended* is false; then increment D by 1, set x to the largest leading variable under the variable order satisfies that $P^x_{D-1} \neq \emptyset$, set *system extended* to true; multiply each polynomial p in P^x_{D-1} by all unsolved variables < the variable stored in the *multiplication history* of p, include the resulting polynomials in P, set x to the next smaller leading variable satisfies that $P^x_{D-1} \neq \emptyset$, if there is no such variable, then set *system extended* to false, *elimination degree* to D, and go back to *Gauss* (Algorithm 2 lines 22 – 39).

To give a more formal description of *MXL2* algorithm and its sub-algorithms, firstly we need to define the following subroutines:

Solve$(Roots, X)$: if there are univariate equations in the *roots*, then solve them and return the solutions.
Substitute$(Solution, roots)$: use all the solutions found to simplify the *roots*.
Reset$(history, n)$: reset history to an array with number of elements equal to n and initialized by ones.
Extend$(history, n)$: append to history an array with number of elements equal to n and initialized by ones.
SelectNecessary(M, D, k, n): compute the necessary number of mutants with degree k as in equation (4), let the mutants be ordered depending on their leading terms, then return the necessary mutants by ascending order.
Xpartition(P, x): return $\{p \in P : LV(p) = x\}$.
LargestLeading(P): return $\max\{y : y = LV(p), p \in P, y \in X\}$.
NextSmallerLeading(P, x): return $\max\{y : y = LV(p), p \in P, y \in X$ and $y < x\}$.

Algorithm1. ***MXL2***

Inputs
  F: set of quadratic polynomials.
  D: highest system degree starts by 2.
  X: set of variables.
Output
  $Solution$: solution of F=0.
Variables
  RP: set of all regular polynomials produced during the process.
  M: set of mutants.
  $roots$: set of all polynomials of degree ≤ 2
  x: variable
  ed: elimination degree
  $history$: array of length $\#RP$ to store previous variable multiplier
  $extended$: a flag to enlarge the system
Begin
$RP \leftarrow F$
$M \leftarrow \varnothing$
$Solution \leftarrow \varnothing$
$ed \leftarrow 2$
$history \leftarrow$ [1,...,1]
$extended \leftarrow$ false
repeat
  Linearize RP using graded lex order
  Gauss(Extract$(RP, ed, \leq)$, $history$)
  $roots \leftarrow roots \cup$ Extract$(RP, 2, \leq)$
  $Solution \leftarrow Solution \cup$ Solve$(roots, X)$
  if there are solutions **then**

```
            roots ← Substitute(Solution, roots)
            RP ← roots
            history ← Reset(history, #roots)
            M ← ∅
            ed ← D ← max{deg(p) : p ∈ roots}
        else
            M ← M ∪ Extract(RP, D − 1, ≤)
            RP ← RP ∪ Enlarge(RP, M, X, D, x, history, extended, ed)
        end if
    until roots = ∅
    End
```

Algorithm2: Enlarge$(RP, M, X, D, x, history, extended, ed)$

```
history, extended, ed: may be changed during the process.
Variable
NP: set of new polynomials.
NM: necessary mutants
Q: set of degree D-1 polynomials have leading variable x
k: minimum degree of the mutants
Begin
NP ← ∅
if M ≠ ∅ then
    k ← min{deg(p) ∈ M}
    NM ← SelectNecessary(M, D, k, #X)
    Extend(history, #X · #NM)
    for all p ∈ NM do
        for all y in X do
            NP ← NP ∪ {y · p}
            history[y · p] = y
        end for
    end for
    M ← M\ SM
    ed ← k + 1
else
    if not extended then
        D ← D + 1
        x ← LargestLeading(Extract(RP, D − 1, =))
        extended ← true
    end if
    Q ← XPartition(Extract(RP, D − 1, =), x)
    Extend(history, #X · #Q)
    for all p ∈ Q do
        for all y ∈ X: y < history[p] do
            NP ← NP ∪ {y · p}
            history[y · p] ← y
```

end for
end for
$x \leftarrow$ NextSmallerLeading(Extract($RP, D-1, =$), x)
if x is undefined **then**
extend $\leftarrow$ false
end if
$ed \leftarrow D$
end if
Return NP
End

Algorithm3: Extract(P, *degree*, *operation*)

P: set of polynomials
SP: set of selected polynomials
operation: conditional operations belongs to $\{<, \leq, >, \geq, =\}$
Begin
for all $p \in P$ **do**
if deg(p) *operation degree* **then**
$SP \leftarrow SP \cup \{p\}$
end if
end for
End

We show that the system is partially enlarged, so *MXL2* leads to the original MutantXL if the system is solved with the last partition enlarged. Whereas *MXL2* outperforms the original MutantXL if it solves the system by earlier partition enlarged. This will be clarified experimentally in the next section.

4 Experimental Results

In this section, we present the experimental results for our implementation of the *MXL2* algorithm. We compare *MXL2* with the original MutantXL, Magma's implementation of F_4, and the XL algorithm for some random systems (5-24 equations in 5-24 variables). The results can be found in Table 1. Moreover, we have another comparison for *MXL2*, original MutantXL, and Magma for some HFE systems (25-55 equations in 25-55 variables) in order to clarify that mutant strategy has the ability to be helpful with different types of systems. See the results in Table 2. For XL and MutantXL, all monomials up to the degree bound D are computed and accounted for as columns in the matrix, even if they did not appear in any polynomial. For *MXL2* on the other hand, we omitted columns that only contained zeros.

Random systems were taken from [9], HFE systems (30-55 equations in 30-55 variables) were generated with code contained in [10], and one HFE system (25 equations in 25 variables) was taken from the Hotaru distribution [11]. The results for F_4 were obtained using Magma version 2.13-10; the parameter

Table 1. Random Comparison

# Var # Eq	XL	MutantXL	Magma	*MXL2*
5	30×26	30×26	30×26	**20×25**
6*	42×42	47×42	46×40	**33×38**
7*	203×99	154×64	154×64	**63×64**
8*	296×163	136×93	131×88	**96×93**
9	414×256	414×256	480×226	**151×149**
10	560×386	560×386	624×3396	**228×281**
11	737×562	737×562	804×503	**408×423**
12	948×794	948×794	1005×704	**519×610**
13	1196×1093	1196×1093	1251×980	**1096×927**
14*	6475×3473	1771×1471	1538×1336	**1191×1185**
15*	8520×4944	2786×2941	2639×1535	**1946×1758**
16	11016×6885	11016×6885	9993×4034	**2840×2861**
17	14025×9402	14025×9402	12382×5784	**3740×4184**
18	17613×12616	17613×12616	15187×8120	**6508×7043**
19	21850×16664	21850×16664	18441×11041	**9185×11212**
20	26810×21700	26810×21700	22441×14979	**14302×12384**
21*	153405×82160	31641×27896	26860×19756	**14365×20945**
22*	194579×110056	92831×35443	63621×21855	**35463×25342**
23*	244145×145499	76558×44552	41866×29010	**39263×36343**
24*	no sol. obtained	298477×190051	207150×78637	**75825×69708**

`HFE:=true` was used to solve HFE systems. The *MXL2* algorithm has been implemented in C/C++ based on the latest version of M4RI package [12]. For each example, we give the number of equations (#Eq), number of variables (#Var), the degree of the hidden univariate high-degree polynomial for HFE (HUD) and the size of the largest linear system to which Gauss is applied. The '*' in the first column for random systems means that, there are some mutants in this system.

In all experiments, the highest degree of the polynomials generated by MutantXL and *MXL2* is equal to the highest degree of the S-polynomial in Magma. In *MXL2* implementation, we use only one matrix from starting to the end of the process by enlarging and extending the initial matrix, the largest matrix is the accumulative of all polynomials that are held in the memory. unfortunately, in Magma we can not know the total accumulative matrices size because it is not an open source.

In Table 1, we see that in practice *MXL2* is an improvement for memory efficiency over the original MutantXL. For systems for which mutants are produced during the computation, MutantXL is better than XL. If no mutants occur, MutantXL behaves identically to XL. Comparing XL, MutantXL, and *MXL2*; *MXL2* is the most efficient even if there are no mutants. In almost all cases *MXL2* has the smallest number of columns as well as a smaller number of rows compared to the F_4 implementation contained in Magma. We can see easily that 70% of the cases *MXL2* is better, 5% is equal, and 25% is worse.

Table 2. HFE Comparison

# Var # Eq	HUD	Magma	MutantXL	*MXL2*
25	96	12495×15276	14219×15276	**11926×15276**
30	64	23832×31931	26922×31931	**19174×31931**
35	48	27644×59536	31255×59536	**30030×59536**
40	33	45210×102091	49620×102091	**46693×102091**
45	24	43575×164221	57734×164221	**45480×164221**
50	40	75012×251176	85025×251176	**67826×251176**
55	48	104068×368831	119515×368831	**60116×368831**

Table 3. Time Comparison

System	MutantXL	*MXL2*
RND5	0.004	0.001
RND6	0.001	0.004
RND7	0.004	0.008
RND8	0.004	0.001
RND9	0.016	0.012
RND10	0.024	0.016
RND11	0.044	0.024
RND12	0.072	0.040
RND13	0.112	0.084
RND14	0.252	0.184
RND15	0.372	0.256
RND16	13.629	1.636
RND17	28.342	2.420
RND18	92.078	9.561
RND19	178.971	20.057
RND20	346.062	70.001
RND21	699.108	126.576
RND22	1182.410	498.839
RND23	1636.000	854.753
RND24	23370.001	12384.700

In Table 2, we also present HFE systems comparison. In all these seven examples for all the three algorithms (Magma's F_4, MutantXL, and *MXL2*), all the monomials up to degree bound D appear in Magma, MutantXL, and *MXL2*. therefore, the number of columns are equal in all the three algorithms. It is clear that *MXL2* has a smaller number of rows in four cases of seven. In all cases *MXL2* outperforms MutantXL.

A time comparison in seconds for random systems between MutantXL and *MXL2* can be found in Table 3. We use in this comparison a Sun Fire X2200 M2 server with 2 dual core Opteron 2218 CPU running at 2.6GHz and 8GB of RAM. We did not make such a comparison between Magma and *MXL2* for HFE instances. This is due to the following reasons: we use a special Magma

Table 4. Strategy Comparison

# Var # Eq	Method1	Method2	Method3	Method4
5	30×26	30×26	25×25	**20×25**
6	47×42	47×42	33×38	**33×38**
7	154×64	63×64	154×64	**63×64**
8	136×93	96×93	136×93	**96×93**
9	414×239	414×239	232×149	**151×149**
10	560×367	560×367	318×281	**228×281**
11	737×541	737×541	408×423	**408×423**
12	948×771	948×771	519×610	**519×610**
13	1196×1068	1196×1068	1616×967	**1096×927**
14	1771×1444	1484×1444	1485×1185	**1191×1185**
15	2786×1921	1946×1921	2681×1807	**1946×1758**
16	11016×5592	10681×5592	6552×2861	**2840×2861**
17	14025×7919	13601×7919	4862×4184	**3740×4184**
18	17613×10930	17086×10930	6508×7043	**6508×7043**
19	21850×14762	21205×14762	9185×11212	**9185×11212**
20	26810×19554	26031×19554	14302×12384	**14302×12384**
21	31641×25447	31641×25447	14428×20945	**14365×20945**
22	92831×34624	38116×32665	56385×28195	**35463×25342**
23	76558×43650	45541×43650	39263×36343	**39263×36343**
24	298477×190051	297810×190051	75825×69708	**75825×69708**

implementation for HFE systems by using the HFE:=true parameter, the *MXL2* implementation is based on M4RI package which is not in its optimal speed as claimed by M4RI contributors and the *MXL2* implementation itself is not optimal at this point. From Table 3, it is clear that the *MXL2* has a good performance for speed compared to MutantXL.

In order to shed light on which strategy (necessary mutants or partitioned enlargement) worked more than the other in which case, we make another comparison for random systems. In this comparison, we have 4 methods that cover all possibilities to use the two strategies. Method1 is for multiplying all lower degree mutants that are extracted at certain level, non of the two strategies are used. Method2 is for multiplying only our claimed necessary number of mutants, necessary mutant strategy. We use Method3 for partitioned enlargement strategy, multiplications are for all lower degree mutants. For both the two strategies which is *MXL2* too, we use Metod4. See Table 4.

In Table 4, comparing Method1 and Method2, we see that practically the necessary mutant strategy sometimes has an effect in the cases which have a large enough number of hidden mutants (cases 7, 8, 14, 15, 22 and 23). In a case that has less mutants (cases 6, 21 and 24) or no mutants at all (cases 5, 9, 10-13, and 16-20), the total number of rows is the same as in Method1. Furthermore, in case 22 because of not all mutants were multiplied, the number of columns is decreased. By comparing Method1 and Method3, most of the cases in the partitioned enlargement strategy have a smaller number of rows except for case

13 which is worst because Method3 extracts mutants earlier than Method1, so it multiplies all these mutants while MutantXL solves and ends before multiplying them. In a case that is solved with the last partition, the two methods are identical (case 7 and 8).

Indeed, using both the two strategies as in Method4 is the best choice. In all cases the number of rows in this method is less than or equal the minimum number of rows for both Method2 and Method3,

$$\#rows\ in\ Method4 \leq min(\#rows\ in\ Method2,\ \#rows\ in\ Method3)$$

In some cases (13, 15 and 22) using both the two strategies leads to a smaller number of columns.

5 Conclusion

Experimentally, we can conclude that the *MXL2* algorithm is an efficient improvement over the original MutantXL in case of GF(2). Not only can *MXL2* solve multivariate systems at a lower degree than the usual XL but also can solve these systems using a smaller number of polynomials than the original MutantXL, since we produce all possible new equations without enlarging the number of the monomials. Therefore the size of the matrix constructed by *MXL2* is much smaller than the matrix constructed by the original MutantXL. We did not claim that we are absolutely better than F_4 but we are going in this direction. We apply the mutant strategy into two different systems, namely random and HFE. We believe that mutant strategy is a general approach that can improve most of multivariate polynomial solving algorithms.

In the future we will study how to build *MXL2* using a sparse matrix representation instead of the dense one to optimize our implementation. We also need to enhance the mutant selection strategy to reduce the number of redundant polynomials, study the theoretical aspects of the algorithm, apply the algorithm to other systems of equations, generalize it to other finite fields and deal with systems of equations that have multiple solutions.

Acknowledgment

We would like to thank Ralf-Philipp Weinmann for several helpful discussions and comments on earlier drafts of this paper.

References

1. Matsumoto, T., Imai, H.: Public Quadratic Polynomial-Tuples for Efficient Signature-Verification and Message-Encryption. In: Günther, C.G. (ed.) EUROCRYPT 1988. LNCS, vol. 330, pp. 419–453. Springer, Heidelberg (1988)
2. Patarin, J.: Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): two new families of Asymmetric Algorithms. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 33–48. Springer, Heidelberg (1996)

3. Patarin, J., Goubin, L., Courtois, N.: C^{*}_{-+} and HM: Variations Around Two Schemes of T. Matsumoto and H. Imai. In: Ohta, K., Pei, D. (eds.) ASIACRYPT 1998. LNCS, vol. 1514, pp. 35–50. Springer, Heidelberg (1998)
4. Moh, T.: A Public Key System With Signature And Master Key Functions. Communications in Algebra 27, 2207–2222 (1999)
5. Ding, J.: A New Variant of the Matsumoto-Imai Cryptosystem through Perturbation. In: Bao, F., Deng, R., Zhou, J. (eds.) PKC 2004. LNCS, vol. 2947, pp. 305–318. Springer, Heidelberg (2004)
6. Courtois, N.T., Klimov, A., Patarin, J., Shamir, A.: Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 392–407. Springer, Heidelberg (2000)
7. Ding, J., Buchmann, J., Mohamed, M.S.E., Moahmed, W.S.A., Weinmann, R.P.: MutantXL. In: Proceedings of the 1st international conference on Symbolic Computation and Cryptography (SCC 2008), Beijing, China, LMIB, pp. 16–22 (2008), `http://www.cdc.informatik.tu-darmstadt.de/reports/reports/MutantXL_Algorithm.pdf`
8. Ding, J., Cabarcas, D., Schmidt, D., Buchmann, J., Tohaneanu, S.: Mutant Gröbner Basis Algorithm. In: Proceedings of the 1st international conference on Symbolic Computation and Cryptography (SCC 2008), Beijing, China, LMIB, pp. 23–32 (2008)
9. Courtois, N.T.: Experimental Algebraic Cryptanalysis of Block Ciphers (2007), `http://www.cryptosystem.net/aes/toyciphers.html`
10. Segers, A.: Algebraic Attacks from a Gröbner Basis Perspective. Master's thesis, Department of Mathematics and Computing Science, TECHNISCHE UNIVERSITEIT EINDHOVEN, Eindhoven (2004)
11. Shigeo, M.: Hotaru (2005), `http://cvs.sourceforge.jp/cgi-bin/viewcvs.cgi/hotaru/hotaru/hfe25-96?view=markup`
12. Albrecht, M., Bard, G.: M4RI – Linear Algebra over GF(2) (2008), `http://m4ri.sagemath.org/index.html`