Algebraic Cryptanalysis of MQQ Public Key Cryptosystem by MutantXL

Mohamed Saied Emam Mohamed[1], Jintai Ding[2], and Johannes Buchmann[1]

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

Abstract. In this paper, we present an efficient attack to the multivariate Quadratic Quasigroups (MQQ) cryptosystem. Our cryptanalysis breaks MQQ cryptosystems by solving systems of multivariate quadratic polynomial equations using a modified version of the MutantXL algorithm. We present experimental results comparing the behavior of our implementation of MutantXL to Magma's implementation of F_4 on MQQ systems ($\geq$ 135 bit). Based on our results we show that the MutantXL implementation solves with much less memory than Magma's implementation of F_4 algorithm.

1 Introduction

The intractability of solving some mathematical problems is the security basis for many public key cryptosystems. One of the most popular cryptosystems based on this criteria is the multivariate cryptosystem which involves the problem of solving large systems of multivariate polynomial equations over finite fields.

The first multivariate public key cryptosystem introduced by Matsumoto and Imai in [13] was broken by Patarin in [15]. Other systems like The Hidden Field Equation (HFE) by Patarin [16] and the unbalanced Oil and Vineger (UOV) by Kipnis, Patarin and Goubin [12] were attacked by Faugère [8] and Wolf et al. [1],[17] respectively.

In recent years, several algorithms to solve systems of multivariate equations such as XL [3], F4 [6], and F5[7] have been introduced. These algorithms generally outperform the standard Buchberger algorithm [2]. In 2006, Ding [4] presented the mutant strategy, which characterizes the degeneration of a polynomial system. He also suggested to use this new strategy to improve the performance of various algorithms. Combining the mutant strategy with the XL algorithm, we showed in [14] that a MutantXL algorithm outperforms F4 in solving HFE based systems as well as random systems.

A new multivariate scheme referred to as Multivariate Quadratic Quasigroups (MQQ) was presented by Gligoroski et al. in, [10], [11]. This cryptosystem is considered to be of higher potential and expected to be as fast as block cipher

[10], [11]. In this paper, we introduce a new implementation of MutantXL, first published by Ding et al., in [5] and improved by Mohamed et al., in [14], that also outperforms F4 in solving MQQ systems. We show by experimental results that our implementation can solve the MQQ multivariate quadratic equation systems using much less memory than Magma's implementation of F_4.

The paper is organized as follows. In Section 2 we study the structure of MQQ systems . In Section 3 we describe the MutantXL algorithm and it's new implementation. Section 4 contains our experimental results. Finally we conclude our paper in section 5.

2 MQQ Cryptosystem

The multivariate quadratic polynomial equations of the MQQ cryptosystem are generated via quasigroup string transformation performed on a class of quasigroups. The public key consists of n quadratic polynomials with n variables. The only parameter that adjusts the security level of the MQQ public key cryptosystem is the number of variables n, where the authors in [10], [11] suggest the length of $n \geq 140$ for a conjectured security level of $2^{\frac{n}{2}}$. In this section we only present a brief overview of the MQQ cryptosystem. A more detailed explanation is found in [10], [11].

Definition 1. *Let $Q = \{a_1, \ldots, a_n\}$ be a finite set of n elements. A quasigroup $(Q, *)$ is a groupoid satisfying the law*

$$(\forall a, b \in Q)(!\exists x, y \in Q)(a * x = b \quad \& \quad y * a = b) \tag{1}$$

The unique solutions to these equations are written $x = a \backslash_* b$ and $y = b /_* a$ where $\backslash_*$ and $/_*$ are called a left parastrophe and a right parastrophe of $*$ respectively. The basic quasigroup string transformation e-transformation is defined as follows [9]:

Definition 2. *A quasigroup e-transformation of a string $S = (s_0, \ldots, s_{k-1}) \in Q^k$ with a leader $l \in Q$ is the function $e_l : Q \times Q^k \rightarrow Q^k$ defined as $T = e_l(S)$, $T = (t_0, \ldots, t_{k-1})$ such that*

$$t_i = \begin{cases} l * s_0 & i = 0 \\ t_{i-1} * s_i & 1 \leq i \leq k-1 \end{cases} \tag{2}$$

Consider the case when each element $a \in Q$ has a unique d-bit representation $x_1, \ldots, x_d \in \{0, 1\}$ such that $a = x_1 x_2 \ldots x_d$. The binary operation $*$ of the Finite quasigroups $(Q, *)$ is equivalent to a vector valued operation $*_{vv} : \{0, 1\}^{2d} \rightarrow \{0, 1\}^d$ defined as:

$$a * b = c \Leftrightarrow *_{vv}(x_1, \ldots, x_d, y_1, \ldots, y_d) = (z_1, \ldots, z_d)$$

where $x_1 \ldots x_d$, $y_1 \ldots y_d$, and $z_1 \ldots z_d$ are binary representations of a, b, and c respectively.

Lemma 1. *For every quasigroup $(Q, *)$ of order 2^d and for each d-bit representation of Q there are a unique vector valued operation $*_{vv}$ and d uniquely determined $2d - array$ of boolean functions $f_1, \ldots, f_d$ such that $\forall a, b, c \in Q$*

$$a * b = c \Leftrightarrow *_{vv}(X^d, Y^d) = (f_1(X^d, Y^d), \ldots, f_d(X^d, Y^d))$$

where $X^d = x_1, \ldots, x_d$, $Y^d = y_1, \ldots, y_d$.

Each $k - array$ boolean function $f(x_1, \ldots, x_k)$ has the following algebraic normal form (ANF):

$$ANF(f) = c_0 + \sum_{1 \leq i \leq k} c_i x_i + \sum_{1 \leq i \leq j \leq k} c_{i,j} x_i x_j + \ldots, \tag{3}$$

where $c_0, c_i, c_{i,j}, \ldots \in \{0, 1\}$. The degrees of the boolean functions f_i are one of the complexity factors of the quasigroup $(Q, *)$.

Definition 3. *A quasigroup $(Q, *)$ of order 2^d is called multivariate quadratic quasigroup (MQQ) of type $Quad_{d-k}Link_k$ if exactly $d - k$ of the polynomials f_i are quadratic and k of them are linear, where $0 \leq k \leq d$.*

The authors in [10], [11] provided a heuristic algorithm to generate MQQs of order 2^d and of type $Quad_{d-k}Lin_k$. The algorithm randomly generates quasigroups until it finds a quasigroup that satisfies the aforementioned definition of MQQs. Several runs of the algorithm generate several MQQs which are then used to generate the public key [10], [11].

3 MutantXL

In this section we briefly describe the MutantXL algorithm which is a variant of the XL algorithm using the mutant strategy. For detailed explanation please refer to [5], [14].

Let $X := \{x_1, \ldots, x_n\}$ be a set of variables and

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

be the ring of polynomial functions over $\mathbb{F}_2$ in X with the monomials of R ordered by the graded lexicographical order $<_{glex}$. Let $P = (p_1, \ldots, p_m) \in R^m$ be a sequence of m quadratic polynomials in R. Throughout the operation of the algorithm, 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. The MutantXL will perform the following statements:

- *Initialize*: Set D to the maximum degree of P, Set the elimination degree d to D and set the set of mutants M to empty.

- *Eliminate*: Compute row echelon form of the set $P_d = \{p \in P : deg(p) \leq d\}$.
- *Solve*: If there are univariate polynomials in P, then determine the values of the corresponding variables. If this solves the system return the solution and terminate, otherwise substitute the values for the variables in P, set d to $max\{deg(p) : p \in P\}$ and go back to *Eliminate*.
- *ExtractMutants*: Add all the new elements of P_d that have degree $< d$ to M.
- *MultiplyMutants*: If M is not empty, then multiply a necessary number of mutants that have degree k $= min\{deg(p) \colon p \in M\}$ as in [14], remove the multiplied polynomials from M, add the new polynomials obtained to P, set d to $k + 1$ and go back to *Eliminate*.
- *Extend*: Multiply a subset of the Higher degree elements in P as described below by all variables $x \in X$ without redundant, set d to D and go back to *Eliminate*.

To clarify our modification to the *Extend* step, we define the following:

- $P_D = \{p \in P : deg(p) = D\}$
- $A = \{p \in P_D : p = m \cdot q$, m is a monomial and $q \in M\}$
- $B = P_D \setminus A$

First extending by adding all the elements of E to P where

$$E = \{p : \quad p = x \cdot b, \quad x \in X \quad and \quad b \in B\},$$

$D = D + 1$ and go back to eliminate.

In the case of the system loops back to the *Extend* step, if the set A is not empty then we multiply the elements of A partially as the same as the enlargement strategy in [14]. After each round MutantXL tries to solve the system until it finds the solution or all the elements in A were multiplied.

4 Experimental Results and Analysis

In this section we present the results of our attack attempts on MQQ systems. We compare the performance of this particular implementation of MutantXL to the performance of Magma's implementation of F4 with respect to memory resources required to solve the MQQ systems. Throughout all our simulations we used a Sun X4440 server, with 4 "Quad-Core AMD Opteron(tm) Processor 8356" CPUs and 128 GB of main memory, each CPU is running at 2,3 GHz. In our first experiment, we set 15 GB as an upper limit to the memory using the command ulimit -v 15000000 and tried to solve three MQQ systems with 135, 150 and 160 variables (the systems were suggested by the author of the MQQ scheme in [10], [11]). All three systems were solved by MutantXL at a significantly low memory cost wheras Magma's implementation of F4 failed to solve under this memory constraint any of these systems.

In our second experiment, we increased the upper limit of the useable memory to 50GB and tried to solve the same MQQ systems mentioned above. We found that Magma's implementation of F4 successfully solves but at a much higher cost of memory compared to that of MutantXL.

Table 1 shows the results of our experiments, where N denotes the number of equations which equal to the number of variables of the initial system, Memory denotes the maximum memory used for each algorithm in MB and Max Matrix denotes the maximum matrix size used in both algorithms.

Table 1. Performance of MutantXL versus F4

N	MutantXL		F4	
	Memory	Max Matrix	Memory	Max Matrix
135	2341	47721×410176	18666	395550×392262
150	3483	51469×562626	24484	501565×497886
160	12955	157933×682801	44241	556942×551029

Tables 2 and 3 show the details of running MutantXL and Magma's implementation of F4 in the case of $n = 150$. In table 2 for each step we show the elimination degree (ED), the matrix size, the rank of the matrix (Rank), the number of mutants found (NM) and the memory required. In table 3 we show for each step the step degree (SD), the number of pairs (NP), the matrix size, and the step memory in MB. For MutantXL the memory at each step is computed by takes into account the total matrix size after each step, which is the accumulative of all polynomials that are held in the memory.

It is clear from table 2 that MutantXL can easily solve the 150 variables MQQ system. In the first iteration of the algorithm, the *Eliminate* step created 4 mutant polynomial equations of degree 1. In the multiply step, 590 linearly independent quadratic equations were generated from these 4 mutants. The resulting equations were then appended to the 146 quadratic polynomial equations produced by eliminating the original system. In the second iteration, no mutants were found, therefore, MutantXL extended the system by multiplying only the 146 quadratic equations producing 21900 cubic equations. In the third iteration step, MutantXL eliminated the extended system thus generating 116 mutants. Further iteration steps continuously generate mutants as shown in the table until one of these mutants is univariate which finally leads to solving the system.

The previous analysis shows that MQQ is vulnerable to attacks from MutantXL algorithm due to the fact that it generally generates large number of mutants in early iteration steps. On the other hand, MQQ can resist attacks

Table 2. MutantXL Results for MQQ-150

Step	ED	Rank	NM	Matrix Size	Memory
1	2	150	4	150×11326	0.2
2	2	740	0	750×11326	1.2
3	3	22640	116	22640×562626	1518
4	3	40033	6	40040×562626	2685
5	3	40906	11	40933×562626	2745
6	2	2373	5	2539×11326	2843
7	2	3030	6	3123×11326	2893
8	2	3785	4	3930×11326	2947
9	2	4269	4	4385×11326	2978
...	...	...	...	...	...
40	2	11308	3	11452×11326	3483

from F4 algorithm since F4 does not deal with the mutant polynomials produced during the process in a special way as MutantXL does.

Table 3. Magma Results for MQQ-150

Step	SD	NP	Matrix Size	Memory
1	2	141	150×11326	1195
2	2	62	740×11326	1195
3	3	3094	65680×562615	6932
4	3	3658	73074×553196	18795
5	2	231	2429×11260	18815
6	2	109	2916×11146	18815
7	2	123	3064×10540	18815
8	2	86	3468×10464	18815
9	2	86	3198×9726	18815
...	...	...	...	...
40	3	40270	501565×497886	24584

It is important to point out that we did not make a runtime comparison between Magma and MutantXL, since our implementation is based on M4RI package which is not in its optimal form regarding speed as stated by M4RI contributors. Moreover we are still undergoing speed improvements to our current implementation of MutantXL. In the future we plan to parallelize the implementation of MutantXL by performing parallel multiplication of polynomials and using the last version of M4RI package. We expect that this parallel implementation of MutantXL will significantly improve its speed performance.

5 Conclusion

In this article, we performed a very efficient practical cryptanalysis of MQQ Public Key cryptosystem by solving a multivariate quadratic polynomial equations systems based on a quasigroup string transformations and a specific class of quasigroups. To do that, we used MutantXL a variant of XL algorithm. Our attack based on MutantXL implementation which is able to break MQQ systems of size 160 bits. The results presented in this paper show that MutantXL attack is much more efficient than Magma F4 attack to MQQ cryptosystems. We expect that MutantXL can successfully attack more bigger MQQ systems than Faugere's F4, F5 and Magma.

References

1. A. Braeken, C. Wolf, and B. Preneel. A study of the security of unbalanced oil and vinegar signature schemes, 2004.
2. B. Buchberger. *An algorithm for finding a basis for the residue class ring of a zero-dimensional polynomial ring.* Dissertation, University of Innsbruck, 1965.
3. N. Courtois, A. Klimov, J. Patarin, and A. Shamir. Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations. In *Proceedings of International Conference on the Theory and Application of Cryptographic Techniques(EUROCRYPT)*, volume 1807 of *Lecture Notes in Computer Science*, pages 392–407, Bruges, Belgium, May 2000. Springer.
4. J. Ding. Mutants and its impact on polynomial solving strategies and algorithms. Privately distributed research note, University of Cincinnati and Technical University of Darmstadt, 2006.
5. J. Ding, J. Buchmann, M. S. E. Mohamed, W. S. A. Moahmed, and R.-P. Weinmann. MutantXL. In *Proceedings of the 1st international conference on Symbolic Computation and Cryptography (SCC08)*, pages 16 – 22, http://www.cdc.informatik.tu–darmstadt.de/reports/reports/MutantXL_Algorithm.pdf, Beijing, China, April 2008. LMIB.
6. J.-C. Faugére. A new efficient algorithm for computing Gröbner bases (F4). *Pure and Applied Algebra*, 139(1-3):61–88, June 1999.
7. J.-C. Faugére. A new efficient algorithm for computing Gro"bner bases without reduction to zero (F5). In *Proceedings of the 2002 international symposium on Symbolic and algebraic computation (ISSAC)*, pages 75 – 83, Lille, France, July 2002. ACM.
8. J.-C. Faugére and A. Joux. Algebraic Cryptoanalysis of Hidden Field Equation (HFE) Cryptosystems Using Gröbner Bases . In *Proceedings of the International Association for Cryptologic Research 2003*, pages 44 – 60. Springer, 2003.
9. D. Gligoroski. Candidate one-way functions and one-way permutations based on quasigroup string transformations. volume abs/cs/0510018, 2005.
10. D. Gligoroski, S. Markovski, and S. J. Knapskog. Public Key Block Cipher Based on Multivariate Quadratic Quasigroups. In *Cryptology ePrint Archive, Report 2008/320*. http://eprint.iacr.org/.
11. D. Gligoroski, S. Markovski, and S. J. Knapskog. Multivariate Quadratic Trapdoor Functions Based on Multivariate Quadratic Quasigroups. In *Proceedings of*

The AMERICAN CONFERENCE ON APPLIED MATHEMATICS, (MATH08), Cambridge, Massachusetts, USA, March 2008.

12. A. Kipnis, H. S. H. Hotzvim, J. Patarin, and L. Goubin. Unbalanced oil and vinegar signature schemes. In *Advances in Cryptology EUROCRYPT 1999*, pages 206–222. Springer, 1999.
13. T. Matsumoto and H. Imai. Public Quadratic Polynomial-Tuples for Efficient Signature-Verification and Message-Encryption. In *Workshop on the Theory and Application of of Cryptographic Techniques Advances in Cryptology- EUROCRYPT*, volume 330 of *Lecture Notes in Computer Science*, pages 419–453, Davos, Switzerland, May 1988. Springer.
14. M. S. E. Mohamed, W. S. A. E. Mohamed, J. Ding, and J. Buchmann. MXL2: Solving Polynomial Equations over GF(2) using an Improved Mutant Strategy. In *Proceedings of The Second international Workshop on Post-Quantum Cryptography, (PQCrypto08)*, volume 5299 of *Lecture Notes in Computer Science*, pages 203–215, Cincinnati, USA, October 2008. Springer-Verlag, Berlin.
15. J. Patarin. Cryptanalysis of the Matsumoto and Imai Public Key Scheme. In *Procedings of Eurocrypt (Crypto'95)*.
16. J. Patarin. Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): two new families of Asymmetric Algorithms. In *Proceeding of International Conference on the Theory and Application of Cryptographic Techniques Advances in Cryptology- Eurocrypt*, volume 1070 of *Lecture Notes in Computer Science*, pages 33–48, Saragossa, Spain, May 1996. Springer.
17. C. Wolf and B. Preneel. Superfluous keys in multivariate quadratic asymmetric systems. In *Public Key Cryptography PKC 2005*, pages 275–287. Springer, 2005.