Algebraic Attack on the MQQ Public Key Cryptosystem

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

[1] TU Darmstadt, FB Informatik
Hochschulstrasse 10, 64289 Darmstadt, Germany
{mohamed,buchmann}@cdc.informatik.tu-darmstadt.de, fw@cccmz.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 on the multivariate Quadratic Quasigroups (MQQ) public key cryptosystem. Our cryptanalysis breaks the MQQ cryptosystem by solving a system of multivariate quadratic polynomial equations using both the MutantXL algorithm and the F_4 algorithm. We present the experimental results that show that MQQ systems is broken up to size n equal to 300. Based on these results we show also that MutantXL solves MQQ systems with much less memory than the F_4 algorithm implemented in Magma.

Keywords: Algebraic Cryptanalysis, MQQ public key cryptosystem, MutantXL algorithm, F_4 algorithm.

1 Introduction

The intractability of solving mathematical problems is the security basis for many public key cryptosystems. One example are multivariate cryptosystems which are based on the problem of solving large systems of multivariate polynomial equations over finite fields.

The first multivariate public key cryptosystem was the Matsumoto-Imai scheme in [12] which was broken by Patarin in [14]. Other systems like the Hidden Field Equation (HFE) by Patarin [15] and the unbalanced Oil and Vineger (UOV) by Kipnis, Patarin and Goubin in [11] were attacked by Faugère et al. [7] and Wolf et al. [2,16] respectively.

At the American Conference on Applied Mathematics 2008 (MATH08), Gligoroski et al. presented a new multivariate public key encryption scheme referred to as Multivariate Quadratic Quasigroups (MQQ), which, according to the authors, is as fast as highly efficient block ciphers. MQQ is parameterized by the number of variables n in the multivariate polynomials that are used in the public key. The inventors of MQQ claim that for $n \geq 140$ the security level of MQQ is at least $2^{\frac{n}{2}}$, means an attacker must perform at least $2^{\frac{n}{2}}$ elementary operations to discover the plaintext.

J.A. Garay, A. Miyaji, and A. Otsuka (Eds.): CANS 2009, LNCS 5888, pp. 392–401, 2009.
© Springer-Verlag Berlin Heidelberg 2009

In this paper, we present an algebraic attack that breaks the MQQ scheme. We present experiments that show that MQQ is easily broken for n up to 300. For our algebraic attack we use the MutantXL algorithm, which was published by Ding et al. [4], and improved by Mohamed et al. [13]. We slightly adjusted the MutantXL implementation to make it more efficient when attacking MQQ.

We also use Magma's implementation of the F_4 algorithm [6] for this attack. The result is, that F_4 can also successfully attack MQQ, but MutantXL turns out to use significantly less space than Magma's F_4. For example, if $n = 200$ MutantXL requires 6.3 Gigabytes of memory while F_4 needs 17.7 Gigabytes. After the initial publication of our attack on the IACR eprint archive, we were informed that the F_4 attack on MQQ was independently discovered by Ludovic Perret.

The paper is organized as follows. In Section 2 we review the MQQ cryptosystems. In Section 3 we describe the MutantXL algorithm and its adaptation to MQQ. Section 4 contains our experimental results and shows how to cryptanalyze MQQ. Finally we conclude our paper in Section 5.

2 MQQ Cryptosystem

The MQQ public-key cryptosystem is a standard multivariate public key cryptosystem that is constructed using quasigroup string transformation performed on a class of quasigroups. The security parameter is a positive integer n which is the number of variables and polynomials used in the public key. The authors of MQQ proposed the length of $n \geq 140$ for a conjectured security level of $2^{\frac{n}{2}}$. In this Section we present an overview of the MQQ cryptosystem. A more detailed explanation is found in [9,10].

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*

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

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, called e-transformation is defined as follows [8]:

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 $e_l : Q \times Q^k \to Q^k$ 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 where 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 is a unique vector valued operation $*_{vv}$ and d uniquely determined arrays of length $2d$ 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 $k-bit$ boolean function $f(x_1, \ldots, x_k)$ has the following algebraic normal form (ANF):

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

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 $Quad_{d-k}Lin_k$ 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 of [9,10] provide a heuristic algorithm to generate MQQs of order 2^d and of type $Quad_{d-k}Lin_k$. The public and the private keys are constructed as follows.

A system $P' = (p_1, \ldots, p_n)$ of quadratic polynomials over $\mathbb{F}_2$ in n variables is generated using uniformly and randomly selected quasigroups $*_1, \ldots, *_8$ as described in Table 1. That system represents a map $P' : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$. The matrices $T, S \in \mathbb{F}_2^{(n,n)}$ are selected uniformly at random. The public key is the map

$$P = T \circ P' \circ S$$

which can also be represented by n quadratic polynomials in n variables over $\mathbb{F}_2$. The secret key consists of the 10-tuple $(T, S, *_1, \ldots, *_8)$.

The plain text space is $\mathbb{F}_2^n$. A plaintext $x = (x_1, \ldots, x_n) \in \mathbb{F}_2^n$ is encrypted by computing

$$c = P(x)$$

Decrypting means solving the multivariate system $P(x) = c$. If the secret key is known then we can decrypt using the form

$$x = S^{-1} P'^{-1} T^{-1}(c)$$

where P'^{-1} is computed by using the left parastrophes $\backslash_*$ of the quasigroups $*_1, \ldots, *_8$.

The parameter that was suggested for the practical applications of the MQQ scheme is $n (= 140, 160, 180, 200, \ldots)$, where n is the bit length of the encrypted block.

Table 1. Definition of the the nonlinear mapping P'

Input: Integer n, where $n = 5k, k \geq 28$
Output: Eight quasigroups $*_1, \ldots, *_8$ and n multivariate quadratic polynomials P'
1. Randomly generate n Boolean functions $L = (f_1, \ldots, f_n)$ of n variables $x = (x_1, \ldots, x_n)$; 2. Represent a vector L as a string $L = X_1 \ldots X_k$, where X_i are vectors of dimension 5; 3. Generate several MQQs of type $Quad_4Lin1$ and $Quad_5Lin_0$; The algorithm of generating MQQs is described in [9,10] Table 2. 4. Randomly choose $*_1, *_2 \in Quad_4Lin_1$ and $*_3, *_4, *_5, *_6, *_7, *_8 \in Quad_5Lin_0$; 5. Define a $(k-1)-tuple\ I = (i_1, \ldots, i_{k-1})$ where $i_j \in \{1, \ldots, 8\}$ such that, $1, 2$ are repeated 8 times in I, without loss of generality let $i_1, \ldots, i_8 \in \{1, 2\}$. 6. Compute $y = Y_1 \ldots Y_k$ where $Y_1 = X_1, Y_{j+1} = X_j *_{i_j} X_{j+1}$, for $j = 1, 2, \ldots, k-1$; 7. Set a vector $Z = Y_1 \vert\vert Y_{2,1} \vert\vert Y_{3,1} \vert\vert \ldots \vert\vert Y_{8,1}$ that has 13 components as linear Boolean functions, where $Y_{j,1}$ means the first coordinate of the vector Y_j; 8. Transform Z by the bijection Dobbertin: $W = Dob(Z)$; 9. Set $Y_1 = (W_1, W_2, W_3, W_4, W_5), Y_{2,1} = W_6, \ldots, Y_{8,1} = W_13$; 10. Return y as n multivariate quadratic polynomials $P' = \{p'(x_1, \ldots, x_n)), i = 1, \ldots, n\}$ and the eight Quasigroups $*_1, \ldots, *_8$;

3 MutantXL

MutantXL is an efficient algorithm for solving systems of multivariate polynomial equations that have only one solution. It is a variant of the XL algorithm [3] uses mutant strategy [5].

Let F be a finite field and q be its cardinality. We consider the ring

$$R = F[x_1, \ldots, x_n]/(x_1^q - x_1, ..., x_n^q - x_n)$$

of functions over F in the n variables $x_1, \ldots, x_n$. Here $x_i^q - x_i = 0$, $1 \leq i \leq n$ are the so-called field equations. In R, each element is uniquely expressed as a polynomial where each x_i has degree less than q. Let the monomials of R are ordered by the graded lexicographical order $<_{glex}$.

Let P be a finite set of polynomials in R. Given a degree bound D, the XL algorithm is simply based on extending the set of polynomials P by multiplying each polynomial in P by all the possible monomials such that the resulting polynomials have degree less than or equal to D. Then, by using linear algebra, XL computes a row echelon form of the extended set P. XL uses univariate polynomials in this row echelon form of P to solve $P(\underline{x}) = 0$ at least partially. If the system can not be solved, D is increased.

In [5,4], it was pointed out that during the linear algebra step, certain polynomials of degrees lower than expected appear. These polynomials are called mutants. The mutant strategy is to give mutants a predominant role in the process of solving the system. The precise definition of mutants is as follows:

Let I be the ideal generated by the finite set of polynomials P. An element f in I can be written as

$$f = \sum_{p \in P} f_p p \qquad (4)$$

where $f_p \in R$. The maximum degree of $f_p p$, $p \in P$, is the *level* of this representation. The level of f is the minimum level of all of its representations. The polynomial f is called *mutant* with respect to P if deg(p) is less than its level.

We describe the MutantXL algorithm and its adaptation to MQQ. The input of MutantXL is a set P. The output of MutantXL is a vector $x = (x_1, \ldots, x_n) \in \mathbb{F}^n$ such that $p_i(x) = 0$, $1 \leq i \leq m$. The MutantXL algorithm executes the following steps:

- *Initialize*: Set the degree bound D to the maximum degree of the polynomials in P, set the elimination degree d to the minimum degree of the polynomials in P, and set the set of mutants M to the empty set.
- *Eliminate*: Compute the row echelon form of the set $P_d = \{p \in P : deg(p) \leq d\}$. Here polynomials are identified with their coefficient vectors as explained in [6].
- *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 of the variables in P, set D to $max\{deg(p) : p \in P\}$, set d to D, 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 $k = min\{deg(p): p \in M\}$ by all monomials of degree one, remove the multiplied polynomials from M, add the new polynomials obtained to P, set d to $k + 1$, and go back to *Eliminate*. The necessary number of mutants are numerically computed as in [13].
- *Extend*: Extend P by adding all polynomials that are obtained by multiplying the degree D elements in P by all monomials of degree one. Increment D by one, set d to D and go back to *Eliminate*.

We explain why and how we adapted MutantXL to MQQ. When MutantXL is not successful in solving the system $P = 0$ with a certain degree bound, the algorithm increments the degree bound by one and extends P in the *Extend* step. For MQQ it turned out that *Extend* yields more polynomials than required to solve the system. In the *Extend* step we therefore only multiply the degree D polynomials from the original system. We do not use the degree D polynomials that were generated from mutants.

4 Experimental Results and Analysis

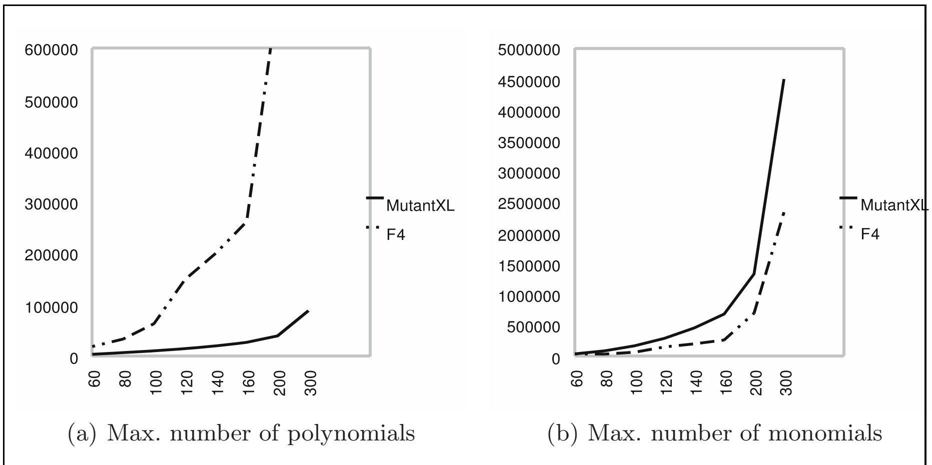
We performed several experiments to attack MQQ systems built using the algorithm described in [9,10]. These systems come from decrypting ciphertext using

the public key but not the secret key. The MQQ inventors supplied us with a few MQQ systems that are not sufficient for the analysis. However, we used them to confirm our implementation of the MQQ cryptosystem. We generated some systems for $n = 60, 80, \ldots, 300$ that were created using MQQs of type $Quad_4 Lin_1$ and $Quad_5 Lin_0$ as in Table 1. According to [9,10], these correspond to $30, 40, \ldots, 150$ bits of security. Our experiment setup has a Sun X4440 server, with four "Quad-Core AMD Opteron^{TM} Processor 8356" CPUs and 128 GB of main memory. Each CPU is running at 2,3 GHz. MutantXL code at the moment uses only one out of the 16 cores. We used both our MutantXL variant and the Magma's implementation of the F_4 algorithm (version V2.13-10).

Table 2 shows the results of our attacks. There we list the number n of variables and equations, the maximum required memory in Megabytes, the maximum matrix size, and the executed time in seconds. It is clear from Table 2 that all systems up to $n = 300$ were successfully attacked by MutantXL as well as Magma's implementation of F_4.

Figure 1(a) compares the maximum number of polynomials used in case of MutantXL and Magma's F_4. We noticed from it that the MutantXL algorithm solves the MQQ systems with smaller number of polynomial equations than Magma's F_4. Conversely, Figure 1(b) shows that the number of monomials of Magma's F_4 is smaller than MutantXL. This is due to the special selection strategy used by the F_4 algorithm, while MutantXL multiplies polynomials of the initial system by all monomials up to certain degree D. For the MQQ systems, all the quadratic monomials appear in the initial system. In this case, all the monomials up to degree D will appear in the enlarged system.

Table 3 and Table 4 show the steps of solving an MQQ system for $n = 200$ using MutantXL and Magma's F_4, respectively. In Table 3, for each step we show the elimination degree (d), the matrix size, the rank of the matrix (Rank), the



(a) Max. number of polynomials (b) Max. number of monomials

Fig. 1. Comparison between MutantXL and F_4 for MQQ

Table 2. Performance of MutantXL versus F_4

n	MutantXL			F_4		
	Memory MB	Max Matrix	Time in in sec.	Memory MB	Max Matrix	Time in in sec.
60	1.6	3714×3605	8	88.8	18835×35918	4
80	70.1	6830×85401	23	217.5	33267×32863	10
100	212.7	10649×166751	76	538	63258×62697	55
120	498.2	14387×288101	283	1819	149077×148234	298
140	1109	20192×457451	556	2909	200397×199391	873
160	2281	26937×682801	1283	4364	262244×261130	1366
200	6437	39497×1333501	16694	18198	699138×697280	17186
300	47952	88647×4500251	237362	111160	2339710×2336171	387754

Table 3. MutantXL: Results for MQQ-200

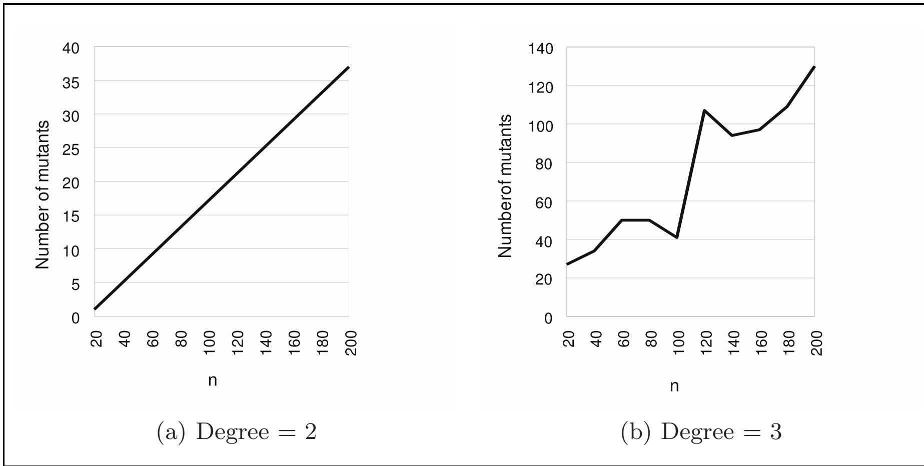
Step	D	Matrix Size	Rank	NM
1	2	200×20101	200	37
2	2	7600×20101	6897	0
3	3	39497×1333501	39497	130
4	2	7427×20101	7347	5
5	2	8347×20101	8125	3
6	2	8725×20101	8584	4
7	2	9384×20101	9183	4
...	...	...	...	...
42	2	20874×20101	20094	4

number of mutants found (NM), and the memory required in Megabyte (MB). In Table 4 we show, for each step, the step degree (SD), the number of pairs (NP), the matrix size, and the step memory in MB.

From Table 3 we see that MutantXL can easily solve the 200 variables MQQ system. In the first iteration of the algorithm, the *Eliminate* step created 37 mutant polynomial equations of degree 1. In the multiply step, 6697 linearly independent quadratic equations were generated from these 37 mutants. The resulting equations were then appended to the 163 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 163 quadratic equations producing 32600 cubic equations. In the

Table 4. Magma-F_4: Results for MQQ-200

Step	SD	NP	Matrix Size
1	2	192	200×20101
2	2	163	6897×20101
3	3	4640	26732×721928
4	2	84	914×13366
5	2	182	1948×13366
6	2	130	2596×13233
7	2	148	3409×13189
...	...	...	...
42	3	58891	699138×697280



(a) Degree = 2

(b) Degree = 3

Fig. 2. Relation between n and the number of mutants obtained

third iteration, MutantXL eliminated the extended system thus generating 128 quadratic mutants and two linear mutants. Further iteration steps continuously generate linear mutants as shown in Table 3 until some of these mutants are univariate which finally leads to solving the system.

It was indeed observed that all MQQ systems offer enough algebraic information (in the sense that it finds enough mutants) to the MutantXL algorithm such that it was always able to solve the system having a critical degree of 3. Figure 2(a) shows the direct linear relation between the size of the initial system n and the number of linear mutants obtained from it. On the other hand Figure 2(b) shows the relation between n and the number of mutants obtained from the extended degree 3 system. Both figures point out two main drawbacks of the MQQ system: First, the initial systems contains linear equations, which is easily discovered in the first step of MutantXL. Second, at degree 3, the system keeps producing mutants until the system is solved. This explains why MQQ systems are solved at degree 3 and therefore can be easily defeated.

We are going to estimate the security level of MQQ against attacks using MutantXL. The MQQ systems that we have broken are solved by MutantXL and F_4 at degree $D = 3$ as explained above. It is reasonable to assume that all the MQQ systems can be defeated at degree $D = 3$. For MutantXL the memory resources are basically measured by the matrix size. From the linear relation explained in Figure 2(a), we can easily estimate the number of linear mutants obtained in the first step. This will enable us to calculate the maximum number of polynomials. For MutantXL, all the monomials appear. We claim that MutantXL can attack MQQ cryptosystems up to $n = 365$ using the same memory resources of the architecture specified above (128 GB). In this case, the expected matrix size is 133590×8104461 which needs $\simeq 126$ Gigabyte of space in less than 10 days.

5 Conclusion

In this article, we have performed an efficient practical cryptanalysis of MQQ Public Key cryptosystem by solving systems of multivariate quadratic polynomial equations. We used an adapted version of MutantXL in this attack. We also compared our attack to the attack using F_4 in Magma. The results showed that both MutantXL and F_4 successfully attack the MQQ cryptosystem. We analyzed the reason why MutantXL could break MQQ efficiently, which is that there are many mutants in the polynomial systems coming from MQQ.

Our implementation of MutantXL uses the M4RI package that is based on the Method of the Four Russians Inversion algorithm published by Bard [1]. M4RI is not optimized for this application, so we claim that further optimization will speed up MutantXL. Also we plan to parallelize MutantXL by performing parallel multiplication of polynomials and using the latest version of the M4RI package. We expect that this parallel implementation of MutantXL will significantly improve its speed performance. Also Magma's implementation of F_4 does not use mutants, so we plan to combine the mutant strategy with the F_4 algorithm.

Acknowledgment

We would like to thank Danilo Gligoroski for supplying us with some MQQ systems generated by his implementation.

References

1. Bard, G.V.: Accelerating cryptanalysis with the Method of Four Russians. Report 251, Cryptology ePrint Archive (2006)
2. Braeken, A., Wolf, C., Preneel, B.: A study of the security of Unbalanced Oil and Vinegar signature schemes. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 29–43. Springer, Heidelberg (2005)
3. Courtois, N., 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)

4. 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, pp. 16–22. LMIB (April 2008)
5. Ding, J., Gower, J.E., Schmidt, D.S.: Zhuang-Zi: A New Algorithm for Solving Multivariate Polynomial Equations over a Finite Field. Technical Report 038, Cryptology ePrint Archive (2006)
6. Faugère, J.-C.: A new efficient algorithm for computing Gröbner bases (F4). Journal of Pure and Applied Algebra 139(1-3), 61–88 (1999)
7. Faugère, J.-C., Joux, A.: Algebraic Cryptoanalysis of Hidden Field Equation (HFE) Cryptosystems Using Gröbner Bases. In: Proceedings of the International Association for Cryptologic Research 2003, pp. 44–60. Springer, Heidelberg (2003)
8. Gligoroski, D.: Candidate One-Way Functions and One-Way Permutations Based on Quasigroup String Transformations. Report 352, Cryptology ePrint Archive (2005)
9. Gligoroski, D., Markovski, S., Knapskog, S.J.: Multivariate Quadratic Trapdoor Functions Based on Multivariate Quadratic Quasigroups. In: Proceedings of The American Conference on Applied Mathematics (MATH 2008), Cambridge, Massachusetts, USA (March 2008)
10. Gligoroski, D., Markovski, S., Knapskog, S.J.: Public Key Block Cipher Based on Multivariate Quadratic Quasigroups. Report 320, Cryptology ePrint Archive (2008)
11. Kipnis, A., Hotzvim, H.S.H., Patarin, J., Goubin, L.: Unbalanced oil and vinegar signature schemes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 206–222. Springer, Heidelberg (1999)
12. 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)
13. Mohamed, M.S.E., Mohamed, W.S.A.E., Ding, J., Buchmann, J.: MXL2: Solving Polynomial Equations over GF(2) using an Improved Mutant Strategy. In: Buchmann, J., Ding, J. (eds.) PQCrypto 2008. LNCS, vol. 5299, pp. 203–215. Springer, Heidelberg (2008)
14. Patarin, J.: Cryptanalysis of the Matsumoto and Imai Public Key Scheme. In: Coppersmith, D. (ed.) CRYPTO 1995. LNCS, vol. 963, pp. 248–261. Springer, Heidelberg (1995)
15. 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)
16. Wolf, C., Preneel, B.: Superfluous keys in multivariate quadratic asymmetric systems. In: Vaudenay, S. (ed.) PKC 2005. LNCS, vol. 3386, pp. 275–287. Springer, Heidelberg (2005)