Dear all,

There seems to be a mistake in the dme_implementation document. The irreducible polynomial $S^3 + c S^2 + d S + e$ defined at page 3 is in fact not irreducible.

The constants $c$ and $d$ in the document do not agree with the value in the reference implementation (which are likely the correct values because they do define an irreducible polynomial).

Kind regards,

--

You received this message because you are subscribed to the Google Groups "pqc-forum" group. To unsubscribe from this group and stop receiving emails from it, send an email to pqc-forum+unsubscribe@list.nist.gov. Visit this group at https://groups.google.com/a/list.nist.gov/group/pqc-forum/.
Dear all,

I believe that the DME system does not reach the claimed level of security.

The public key is a polynomial map with 6 components in 6 variables over a finite field GF(2^48) of the form \( P = L_3 \circ G_2 \circ L_2 \circ G_1 \circ L_1 \), where the \( L_i \) are invertible linear maps.

If we represent the map over GF(2), we get a map \( P' \) of 6*48 components in 6*48 variables, and the maps \( G_i \) become quadratic maps \( G'_i \).

We can decompose the degree 4 map \( P' \) into the composition of two quadratic maps with the algorithm of Faugère and Perret [1]. The Complexity of this algorithm is \( O(n^9) \), where \( n \) is the number of variables, so we can estimate the bit cost of this step to be roughly \( (6*48)^9 \), which is approximately \( 2^{74} \), much less than the claimed security level of 256 bits.

Now we have \( P' = Q2 \circ Q1 \), where we know that the \( Q1 \) is isomorphic to \( G'_2 \), and \( Q1 \) isomorphic to \( G'_1 \). Generic algorithms that solve the Isomorphism of Polynomials e.g.[2] can recover this isomorphism with a cost that is \( O(2^{n/2}) \), so approximately \( 2^{144} \). Once these isomorphisms are recovered, they can be used to invert the public key and thus break the cryptosystem.

Note that both steps (finding the decomposition, and finding the isomorphisms) are done with generic algorithms. It is conceivable that the structure of the public map can be exploited to do the steps more efficiently. (And this seems very likely for the isomorphism finding step)

Kind regards,

Ward


---

You received this message because you are subscribed to the Google Groups "pqc-forum" group.
To unsubscribe from this group and stop receiving emails from it, send an email to pqc-forum+unsubscribe@list.nist.gov.
Visit this group at https://groups.google.com/a/list.nist.gov/group/pqc-forum/.
Dear Ward,

thanks again for your remarks. I was not aware of the paper [1]. I will check the paper and i will check also your estimates, but I believe that they are OK. That means that we need to change the parameters of DME to double the security bits, we will explore and implement different options:

DME(4,2,64) this scheme will give 64 monomials in each polynomial, double the message and public key size and I estimate that it will multiply by 4 the computation time.
DME(6,2,44) this scheme will produce 144 monomials, double the message size and multiply by 4 (roughly). We do not implemented yet this scheme and we do not have estimate for the time.

A better option will be change the matrix of the first exponential by putting 3 non zero entries instead of to 2. Let’s denote by DME3 this scheme then DME3(4,2,36) will give the same message size (288 bits) but it will produce 256 monomials and multiply by 4 the public key size. In this case when when we represent the map over $F_2$ we get degree 6 polynomial map that is a composition of a degree 3 and a degree 2 polynomial map. We will try to compute the complexity of the IP problem for the cubic polynomials to estimate the complexity of your attack for this scheme.

Once you have the decomposition and the 2 quadratic maps $G_1$ and $G_2$ how do you invert them and what is the complexity?

Best regards,
Ignacio
Dear Ignacio, Dear all,

TL;DR: I the new parameters of the DME scheme are not secure. The following describes an attack that reduces the security in the exponent by more than half.

Recall that the attack on DME that I proposed earlier consisted of 2 steps.

1) Represent the public key map over $F_2$, which will make it a polynomial map of degree 4. Decompose this polynomial map into the composition of two quadratic polynomial maps.

2) We know each of the quadratic components is isomorphic to a known quadratic map (the representation of $G_1$ and $G_2$ over $F_2$) which we know how to invert. The second step is to recover this isomorphism with an IP solver.

The complexity of the first step with the algorithm of [1] is $O(n^{9})$, where $n$ is the number of variables of the decomposed map, or equivalently the number of bits of a ciphertext.

The complexity of the second with the algorithm of [2] is $O(2^{(n/2)})$.

The adaptations proposed by Ignacio increase $n$ enough to make the second step infeasible. But the complexity of the first step remains much lower than the targeted security level.

I did some experiments and it turns out that the degree of regularity of the quadratic components is very low. In fact, in all experiments the highest degree reached during a GB computation was 3. This means that once a decomposition is found, the public map can be efficiently inverted by doing a GB computations on each component. This means that the complexity of the attack is dominated by the complexity of the first step.

Ignacio also proposed to use 3 (instead of 2) nonzero entries in the rows of matrix of the first exponentiation. With this adaptation, if we represent the public key over $F_2$ it becomes a degree 6 polynomial map which is the composition of a degree 2 with a degree 3 map. I believe (correct me if I am wrong) that the complexity of decomposing this map with the algorithm of [1] is $O(n^{15})$.

According to my experiments the degree of regularity of the cubic component map is 4, so preimages for this map can still be found efficiently with a GB computation, and again we can say that the complexity of the attack is dominated by the complexity of decomposing the public key.

For the different parametrisations of Ignacio I estimate the complexity of the attack as:

DME(4,2,64) $\rightarrow (4*2*64)^9 \sim 2^{81}$

DME(6,2,44) $\rightarrow (6*2*44)^9 \sim 2^{81}$

DME3(4,2,36) $\rightarrow (4*2*36)^{15} \sim 2^{122}$
So none of the proposed parameter sets seem to reach the claimed security level of 256 bits.

I'll finish with a note on the experiments I did. I did not implement the algorithm for decomposing the public key. I generated the two components $L_3 \circ G_2 \circ L_2$ and $G_1 \circ L_1$, represented them over $F_2$, composed these quadratic maps with random invertible linear maps over $F_2$. This is what the output of the decomposition algorithm [1] would output. Then I used the PolyBoRi library [3] to find an inverse for a random point in $GF(2)^{6*e}$. The highest degree encountered during the GB computation was 3, except in the DME3 case, where the degree was 4. As expected I always found a unique solution.

Finding an inverse for one of the components of DME(3,2,24) (which was originally proposed to have 128 bits of security) took only 565 seconds.

Kind regards,
Ward


Sage code for the experiments on the first component of DME(3,2,24), the code for the second component and the experiments for DME3 is very similar. (This was my first time working with sage, so I apologize if my code is not very clean.)

```python
def mulK2(A, B):
    t0 = A[0]*B[0];
    t1 = A[1]*B[0] + A[0]*B[1];
```

from time import time

e = 24;
KS= GF(2);
K = GF(2^e);

R = PolynomialRing(K,'X');
R.inject_variables();

#Find irreducible polynomial
while True:
    a = K.random_element();
    b = K.random_element();
    IP = X^2 + a*X + b;
    if IP.is_irreducible():
        break;

#multiplication over a degree 2 extension field of K

```python
def mulK2(A, B):
    t0 = A[0]*B[0];
    t1 = A[1]*B[0] + A[0]*B[1];
```
t2 = A[1]*B[1];
return (t0 + b*t2 , t1 + a*t2)

#squaring an element of a degree 2 extension of K a total of n times
def squareK2(A,n):
    if n==0:
        return A;
    B = mulK2(A,A);
    return squareK2(B,n-1);

#find values of Eij such that the exponentiation map is bijective (for vectors with nonzero entries)
while True:
    E11 = ZZ.random_element(0,2*e);
    E12 = ZZ.random_element(0,2*e);
    E21 = ZZ.random_element(0,2*e);
    E23 = ZZ.random_element(0,2*e);
    E32 = ZZ.random_element(0,2*e);
    E33 = ZZ.random_element(0,2*e);
    det = -2^(E11+E23+E32)-2^(E12+E21+E33);
    if gcd(det, 2^(2*e)-1) == 1 :
        break;

#Find L1
while True:
    L1 = matrix([[K.random_element(),K.random_element(),0,0,0,0],
                  [K.random_element(),K.random_element(),0,0,0,0],
                  [0,0,K.random_element(),K.random_element(),0,0],
                  [0,0,K.random_element(),K.random_element(),0,0],
                  [0,0,0,0,K.random_element(),K.random_element()],
                  [0,0,0,0,K.random_element(),K.random_element()]));
    if L1.is_invertible():
        break;

# matrix M for conversion between GF(2^e) and GF(2)^e
for i in range(0,6):
    MM.set_block(i,e*i,Mrow0);

def G1L1(x):
    x = L1*x;
    y = matrix(K,6,1);
    y[0],y[1] = mulK2(squareK2((x[0],x[1]),E11),squareK2((x[2],x[3]),E12));
    y[2],y[3] = mulK2(squareK2((x[0],x[1]),E21),squareK2((x[4],x[5]),E23));
    y[4],y[5] = mulK2(squareK2((x[2],x[3]),E32),squareK2((x[4],x[5]),E33));
    return y;
def G1L1Decomposed(x):
    return G1L1(MM*x);

R = BooleanPolynomialRing(e*6,'x')
X = R.gens()
polynomials = [0] * 6*e

# calculating the coefficients of representation of G_1 o L_1 over F_2
for i in range(0,6*e):
    for j in range(i,6*e):
        vec = matrix(K,6*e,1)
        vec[i] = 1;
        vec[j] = 1;
        Q = G1L1Decomposed(vec);
        for k in range(0,6):
            Powers = matrix([ [ (Q[k][0])^(2^n) ] for n in range(0,e)]);
            outVec = Minv * Powers;
            for l in range(0,e):
                if outVec[l,0] != 0:
                    polynomials[e*k+l] += X[i]*X[j];

# Pick a random element to forge a signature for
for i in range(0,6*e):
    if (ZZ.random_element(2) == 0):
        polynomials[i] = polynomials[i] + 1

I = R.ideal(polynomials)

start = time()

GB = I.groebner_basis(prot = True);

print("GB computation took",time()-start,"seconds")

for x in GB:
    print(x)
Dear Ward, dear all

you are right on your comment about the posible decomposition of the map in two exponentials. If it is easy to do the decomposition then there is not great advantage in to use two exponentials. Thanks for the practical proof that if one decompose the cuartic map over $F_2$ in two quadratic maps the resulting quadratic map are not difficult to invert for the proposed parameters, in fact the make reason to take to exponential round was to make it safe against this attack over $F_2$ while keeping the message and key size small.

Below I will explain how can we make the decomposition very hard with a small modification of DME3, namely the total map will be

F: $F_q^6$-->${F_q^8}$ instead of F: $F_q^6$-->$F_q^6$. First let me confess that I am not totally convinced that the main algorithm of [1] can be applied over $F_2$ as is. It seems to me that you need to use a big field, in the case of $F_2$ you can work over an extension $F_{(2^e)}$ but in this case resulting the polynomials will be of higher degree. Even if we use the complexity in [1] the following instance of DME3 will be safe:

We can change the initial linear map by adding a first map $L_0 : F_q^6-->$ $F_q^8$ as follows

$L_0(x_1,\ldots,x_6)=(x_1,\ldots,x_6,x_2*x_4*x_6,0)$ and changing the first exponential map

$G_1: (F_q^2)^4-->(F_q^2)^4$ that now is defined by a 4x4 matrix with with 3 non zero entries $A_1:=(a_{ij})$ and the entries of the last column are $a_{i4}=2^u+2^v$. We use also that $x_2*x_4*x_6$ is non zero.

Let's keep calling by DME3 the scheme. The parameter for the DME3(3,2,48) are

$q=2^{48}$, the total map is $F: F_q^6-->$ $F_q^8$,

each component has 64 monomials and the size of the pk is 3072 bytes this gives

for kem ct=48 pk=3072 sk=288 bytes=33 instead of the original

c=36 pk=2304 sk=288 bytes=33

If we represent the composition of the two exponentials over $F_2$ the first one has degree

$2+2*3=8$ because of the exponents of $x_2*x_4*x_6$. Then the complexity given by [1] is

$O(n^{45})=(48)^{3*15}=O(2^367)$.

Notice that by changing the exponents $a_{i4}$ one can increase easily the degree of the first exponential over $F_2$, for instance taking $a_{i4}=2^u+2^v+2^w$ the degree of $G_1$ is $2+2*4=10$ that gives a complexity for [1] of $O(n^{57})=(48)^{3*19}=O(2^465)$. That means that the scheme is practically immune against any algorithm of decomposition whose complexity depends strongly on the total degree of the maps over $F_2$.

We will use the web page http://www.mat.ucm.es/~iluengo/DME/ an updated version of the documentation with the typos corrected and an appendix given more details of the setting of DME3. In a few weeks will put an optimized implementation of DME3(3,2,48) and DME3(5,2,28), because we will revise first the complexity of the different algorithms of decomposition over $F_2$.

Best regards,

Ignacio