The Giophantus proposal

Rene Peralta [1]

[1]National Institute of Standards and Technology (Gaithersburg, USA)

NIST-internal PQC meeting
September 11th, 2018 (Gaithersburg, USA)

The Giophantus proposal

Highlights:

- A public-key encryption scheme.
- Security based on the hardness of finding low-weight solutions to certain indeterminate equations.
- Original system proposed around 2006 and broken around 2010, this has a “fix” to cover known attacks.
- Most of the action happens in a quotient ring

$$R_q = F_q[t]/(t^n - 1).$$

- Uses bivariate polynomials (linear and quadratic, as far as I can tell) over R_q.

The public key

$$X(x, y) = a_{00} + a_{01}x + a_{10}y$$

The public key

$$X(x, y) = a_{00} + a_{01}x + a_{10}y$$

$$a_{i,j} = t^{n-1} + \ldots + 431t + 22$$

Parameters

Parameters:

- ℓ is a small integer (proposed value is 4).
- q a prime (around 2^{30}).
- n the number of terms in polynomials in t (around 2000).

Private key

- The private key is a pair of polynomials

$$u_x(t), u_y(t)$$

of degree $n-1$ and coefficients in $\{0,1,2,3\}$;

- These are picked at random, so I think you can just store the seed to a prng.

Connecting the private key and the public key

- In the public key

$$X(x, y) = a_{00}(t) + a_{01}(t)x + a_{10}(t)y$$

  - the polynomials $a_{01}(t)$ and $a_{10}(t)$ are chosen at random in R_q;
  - $$a_{00}(t) = -(a_{01}(t)u_x(t) + a_{10}(t)u_y(t)) \in R_q;$$
  - so

$$X(u_x(t), u_y(t)) = 0.$$

Encryption

- Message (in hex) is the coefficients of a polynomial $m(t)$ of degree $n-1$.
- Pick random polynomials r_{ij} in R_q. Let

$$r(x,y) = r_{00}(t) + r_{01}(t)x + r_{10}(t)y.$$

- Pick random (noise) polynomials e_{ij} of degree $n-1$ and coefficients in $\{0,1,2,3\}$. Let

$$e(x,y) = e_{00}(t) + e_{01}(t)x + e_{10}(t)y + e_{11}(t)xy + e_{02}(t)x^2 + e_{20}(t)y^2.$$

- Ciphertext is

$$c(x,y) = m(t) + X(x,y)r(x,y) + \ell \cdot e(x,y).$$

Decryption

- Evaluate $c(x,y)$ at $(u_x(t), u_y(t))$:

$$c(x,y) = m(t) + X(x,y)r(x,y) + \ell \cdot e(x,y).$$

Decryption

- Evaluate $c(x, y)$ at $(u_x(t), u_y(t))$:

$$c(x, y) = m(t) + X(x, y)r(x, y) + \ell \cdot e(x, y).$$

- $c(u_x, u_y) = m(t) + X(u_x, u_y)r(u_x, u_y) + \ell \cdot e(u_x, u_y).$
- $c(u_x, u_y) = m(t) + \ell \cdot e(u_x, u_y).$

Decryption

- Evaluate $c(x,y)$ at $(u_x(t), u_y(t))$:

$$c(x,y) = m(t) + X(x,y)r(x,y) + \ell \cdot e(x,y).$$

- $c(u_x, u_y) = m(t) + X(u_x, u_y)r(u_x, u_y) + \ell \cdot e(u_x, u_y)$.
- $c(u_x, u_y) = m(t) + \ell \cdot e(u_x, u_y)$.
- The coefficients of both summands are less than q. So you can view this as a sum of polynomials over (Z). Then $m(t)$ is just $c(u_x, u_y)$ mod ℓ.

???

Let $X(x,y) = a + bx + cy$. Much of the time the polynomials b and c will be mutually prime. In this case let u, v be such that $ub + vc = -a$ in R_q. Then

$$X(x+u, y+v) = a + b(x+u) + c(y+v) = bx + cy$$

and therefore (recall $\ell = 4$)

$$c(x+u, y+v) = m(t) + (bx+cy)r(x+u, y+v) + 4 \cdot e(x+u, y+v).$$

Therefore the "constant" term of $c(x,y)$ is $m(t)$ plus the "constant" term of $4 \cdot e(x+u, y+v)$.

???

Thus, having oracle access to encryptions of a chosen message allows you to sample from a distribution $m(t) + 4\alpha$, where $m(t)$ is fixed and α is a random polynomial in R_q.

For any coefficient m_i of $m(t)$, the oracle allows you to sample $(m_i + 4\theta)$ mod q where θ is a random integer modulo q. Since q is not divisible by 4, the distribution of $(m_i + 4\theta)$ mod q is not uniform.

I think each m_i in $\{0, 1, 2, 3\}$ gives you a different distribution.

Therefore you can determine m_i by sampling enough times (some function of q).

Sizes

Sizes in bytes ($\ell = 4$, deg(X(x,y)) = deg(r(x,y)) = 1):

Level	Secret Key	Public Key	Ciphertext
I	600	14412	28824
III	866	20796	41592
V	1133	27204	54408

Performance

Performance (in Megacycles) on Xeon E5-1620 3.6GHz.

Level	keygen	encrypt	decrypt
I	92	178	335
III	160	378	716
V	239	626	1186

Optimized implementations do about 20% better.