Computer experiment on key generation for the quantum public key cryptosystem over quadratic fields

Keiichiro Nishimoto Ken Nakamula
Tokyo Metropolitan University

9 March, 2006

Abstract

In CRYPTO2000, Okamoto, Tanaka and Uchiyama [2] first proposed the concept of quantum public key cryptosystems (QPKC) and a concrete practical scheme to realize it. One of it's important ideas is to employ the ring of integers of an algebraic number field. This talk is about the implementation of three processes, namely, key generation, encryption and decryption of the realized QPKC on our system `NZMATH` [1] for number theory. In particular, we experimented about the efficiency of key generation in the case of some quadratic fields. In the process of key generation, since the discrete logarithm problem (DLP) is settled in polynomial time by quantum computers, we only take into account of the efficiency of other parts. As a result of our experiment, we have found that the order of generation key parameters is important, and we propose a different order from the original one so that key generation is much more efficient. We also introduce another new kind of key generation without changing the process of encryption or decryption.

1 Outline of the QPKC

The QPKC uses a quantum turing machine (QTM). Many public key cryptosystems used today are broken in polynomial time if a quantum computer (QC) is realized. So, it is important to study the QPKC assuming the existence of a QC.
Basic points of the QPKC are as follows:

- The DLP is solvable in polynomial time by the QTM using Shor's algorithm [3].
- The QPKC generates key parameters for Knapsack-type cryptosystems by solving the DLP.
- Knapsack-type cryptosystems are based on a NP-hard problem.
- That NP-hard problem is thought to be still hard to solve in probabilistic polynomial time by the QTM.
- The QPKC need the QTM only for key generation, but we can encrypt and decrypt without any QTM.

2 Algorithm for quadratic fields

Let $K = \mathbb{Q}(\sqrt{\theta})$ be a quadratic field with a square-free $\theta \in \mathbb{Z}$ and $\mathcal{O}_K$ be the ring of integers of K. Then

$$\mathcal{O}_K = \mathbb{Z}[\omega] \quad \text{with} \quad \omega = \begin{cases} \sqrt{\theta} & \text{if} \quad \theta \equiv 2, 3 \pmod 4, \\ \frac{1+\sqrt{\theta}}{2} & \text{otherwise.} \end{cases}$$

Let $n, k \in \mathbb{Z}$ with $0 < k \leq n$. We first give the original algorithms.

Algorithm 1 (Key Generation) Given (n, k, θ), this algorithm outputs a private key $[\theta, g, d, p, S]$ and a public key $[n, k, b]$ by the following steps.

1. Choose a random prime ideal (p) of degree 2 from $\mathcal{O}_K$ with $p \in \mathbb{Z}$. (Actually, it is difficult to obtain a proper p satisfying condition (1) in step 4 below.)
2. Choose a subset $S = \{S_1, \cdots, S_n\}$ of n elements of A, where
$$A = \left\{ \alpha + \beta\omega \mid \alpha,\, \beta \in \mathbb{Z},\, -\frac{p}{2} \leq \alpha,\, \beta \leq \frac{p}{2} \right\}.$$
3. If the absolute norms $\mathcal{N}(S_1), \cdots, \mathcal{N}(S_n)$ are not pairwise coprime, then repeat from step 2.
4. If the following condition (1) is not satisfied, then repeat from step 2 for a certain number of times, for example nk times, after that repeat from step 1 again.

For any subset $\{S_{i_1}, S_{i_2}, \cdots, S_{i_k}\}$ of k elements of S,

$$\prod_{j=1}^{k} S_{i_j} \in A. \tag{1}$$

Especially in case K is imaginary, we employ a sufficient condition of (1), namely

$$\prod_{j=1}^{k} \mathcal{N}(S_{i_j}) \; < \; \begin{cases} \frac{p^2}{4} & \text{if} \quad \theta \equiv 2, 3 \pmod 4, \\ \frac{(p-1)^2\theta}{4(\theta-1)} & \text{otherwise.} \end{cases} \tag{2}$$

5. Find a random $g \in \mathcal{O}_K$ such that $\langle g \bmod (p) \rangle = (\mathcal{O}_K/(p))^\times$.

   Choose a random $d \in \mathbb{Z}$ with $1 \le d \le p^2 - 1$.

   For each i from 1 to n, do the following:

   ```
       Compute $a_i$ such that $g^{a_i} \equiv S_i \pmod{(p)}$ using Shor's algorithm.
       Compute $b_i \equiv a_i + d \pmod{p^2 - 1}$.
   ```

   And set $b := [b_1, \cdots, b_n]$.

   Then output the private key $[\theta, g, d, p, S]$ and the public key $[n, k, b]$.

Remark. This algorithm may fail in step 3 or 4 and will not succeed to get proper $[p, S]$. We shall discuss this problem in the next section.

Algorithm 2 (Encryption) Given a public key $[n, k, b]$ and a plaintext M of bit length $\lfloor \log_2 \binom{n}{k} \rfloor$, this algorithm outputs the ciphertext c.

1. Encode M to $m = (m_1, m_2, \cdots, m_n)$ of Hamming weight k as follows:

   (a) Set $l \leftarrow k$.

   (b) For i from 1 to n do the following:

   ```
       If $M \ge \binom{n-i}{l}$, then set $m_i \leftarrow 1$, $M \leftarrow M - \binom{n-i}{l}$, $l \leftarrow l - 1$.
       Otherwise, set $m_i \leftarrow 0$.
   ```

2. Compute and output the ciphertext

$$c = \sum_{i=1}^{n} b_i m_i.$$

Algorithm 3 (Decryption) Givien a ciphertext c, the private key $[\theta, g, d, p, S]$ and the public key $[n, k]$, this algorithm outputs the plaintext M.

```
1. Compute $r \equiv c - kd \pmod{p^2 - 1}$.
2. Compute $u \equiv g^r \pmod{(p)}$.
3. Find encoded message $m$ as follows:
       If $S_i \mid u$, which is verified by $\mathcal{N}(S_i) \mid \mathcal{N}(u)$, then set $m_i \leftarrow 1$.
       Otherwise, set $m_i \leftarrow 0$.
4. Decode $m$ to the plaintext $M$ as follows and output $M$:
   (a) Set $M \leftarrow 0$, $l \leftarrow k$.
   (b) For $i$ from 1 to $n$, do the following:
           If $m_i \leftarrow 1$, then set $M \leftarrow M + \binom{n-i}{l}$ and $l \leftarrow l - 1$.
```

3 Implementation on NZMATH

We have implemented Algorithm 1 as `generate.para` on `NZMATH`. We give two numerical examples for $(n, k, \theta) = (6, 2, -3)$ and $(7, 2, 5)$. The integer θ is displayed by `m` below.

```
>>> generate.para_orig(8,3,-3)
Execution Time(sel)= 0.74
Execution Time(key)= 17.19
Generate Keys(pub) = [[-3, [960, 1456], 648674, 2081,
 [[-8, -1], [-1, -5], [0, 4], [3, -5], [-6, -5], [-2, -7], [1, 6], [-3, -7]]]
Generate Keys(pub) =
 [8, 3, [4019930, 603640, 3416346, 1448597, 2796778, 1041704, 3096802, 1188333]

>>> generate.para_orig(8,3,5)
Execution time(sel)= 0.41
Execution Time(key)= 2.82
Generate Keys(sel) = [[5, [712, 335], 37290, 877,
 [[-5, 4], [-6, 1], [1, -4], [5, -6], [-1, 8], [-2, -7], [2, 1], [0, 3]]]
Generate Keys(pub) =
 [8, 3, [200313, 223704, 593455, 349652, 152156, 102632, 651791, 279080]]
```

We have also implemented Algorithm 2 and Algorithm 3 on `NZMATH`. We give two numerical examples of all processes, namely key generation, encryption and decryption, for $(n, k, \theta, M) = (10, 3, -3, 48)$.

```
>>> ex.QPKC_orig(10,3,-3,48)
Execution time(sel)= 1.33
Execution Time(key)= 67.34
Input Parameter    = [10, 3, -3, 48]
Generate Keys(pri) = [-3, [460, 2655], 4850724, 3671,
 [[8, 4], [3, 8], [-9, -5], [-5, 6], [-3, 0],
 [6, 1], [4, 5], [1, 8], [-3, 4], [-7, -6]]]
Generate Keys(pub) =
 [10, 3, [1472702, 5403077, 2160433, 9909274, 8746716,
 11630355, 9914031, 1770015, 11028303, 267935]]
Ciphertext         = 20821180
Decoded Message    = 48
Execution Time     = 67.34
```

Remark. By the examples above, we can ignore the time of execution spent in the processes of encryption and decryption.

4 Result of experiments

For each (n, k, θ), we experimented key generation ten times in certain quadratic fields, and computed the average of it's execution time. We used a slightly refined version of Algorithm 1 for experiment because it is very hard to get much data by the original one. Hardware and software status is

CPU: Celeron 2GHz,

Memory: 1024MB,

Programming Language: `Python`.

When n, k are large, the time required in steps 1–4 increases, so it is hard to complete the choice of key. We list tables of the cases for small n, k.

Table 1.
Input parameter, p's value and execution time in imaginary quadratic fields.

$[n, k, \theta]$	p's value	steps 1–4	steps 1–5
[4, 2, -1]	37	0.015	0.025
[4, 2, -2]	56	0.021	0.042
[4, 2, -3]	38	0.013	0.025
[4, 2, -5]	139	0.045	0.10
[6, 2, -1]	112	0.050	0.12
[6, 2, -2]	369	0.135	0.68
[6, 2, -3]	154	0.065	0.17
[6, 2, -5]	1359	0.427	7.04
[7, 3, -1]	743	0.254	2.134
[7, 3, -2]	1050	0.342	4.948
[7, 3, -3]	330	0.13	0.516
[7, 3, -5]	4073	1.24	60.11
[8, 3, -1]	1359	0.482	14.514
[8, 3, -2]	11843	2.992	$\times$
[8, 3, -3]	491	0.184	1.324
[8, 3, -5]	99311	20.096	$\times$
[10, 4, -1]	18641	5.382	$\times$
[10, 4, -2]	209942	49.663	$\times$
[10, 4, -3]	8558	2.827	$\times$
[10, 4, -5]	3446752	643.78	$\times$
[12, 4, -1]	727709	164.736	$\times$
[12, 4, -2]	–	$\times$	$\times$
[12, 4, -3]	21174	6.422	$\times$
[12, 4, -5]	–	$\times$	$\times$
[15, 5, -1]	–	$\times$	$\times$
[15, 5, -2]	–	$\times$	$\times$
[15, 5, -3]	893169	245.288	$\times$
[15, 5, -5]	–	$\times$	$\times$

Table 2.
Input parameter, p's value and execution time in real quadratic fields.

$[n,k,\theta]$	p's value	steps 1–4	steps 1–5
[4, 2, 1]	87	0.02	0.02
[4, 2, 2]	46	0.02	0.035
[4, 2, 3]	42	0.025	0.04
[4, 2, 5]	35	0.02	0.03
[6, 2, 1]	541	0.27	0.28
[6, 2, 2]	274	0.10	0.32
[6, 2, 3]	300	0.113	0.435
[6, 2, 5]	91	0.042	0.09
[7, 3, 1]	10633	3.27	3.38
[7, 3, 2]	1042	0.346	3.786
[7, 3, 3]	2043	0.592	14.084
[7, 3, 5]	692	0.304	1.9
[8, 3, 1]	32634	17.89	18.19
[8, 3, 2]	1729	0.598	14.832
[8, 3, 3]	2960	0.948	38.536
[8, 3, 5]	1266	0.52	7.026
[10, 4, 1]	1704882	649.72	$\times$
[10, 4, 2]	6725	17.683	$\times$
[10, 4, 3]	130511	34.37	$\times$
[10, 4, 5]	25934	8.505	$\times$
[12, 4, 1]	–	$\times$	$\times$
[12, 4, 2]	197465	51.940	$\times$
[12, 4, 3]	791499	193.720	$\times$
[12, 4, 5]	818378	22.522	$\times$
[15, 5, 1]	–	$\times$	$\times$
[15, 5, 2]	–	$\times$	$\times$
[15, 5, 3]	–	$\times$	$\times$
[15, 5, 5]	2534949	694.848	$\times$

5 Consideration

5.1 The order of choosing parameters

From the table above, as n is large, the part of choosing key parameters requires much time before solving the DLP in Algorithm 1. There seems to be two reasons about that.

The first reason is in step 3.

Proposition 1 ([4]) *The probability that random n positive integers are pairwise coprime is*

$$A_n = \prod_{p:prime} (1 - \frac{1}{p})^{n-1}(1 + \frac{n-1}{p}).$$

By this proposition, we have

$$A_{10} = 8.1 \times 10^{-5}, \qquad A_{50} = 1.7 \times 10^{-34}.$$

From this, the probability that almost randomly taken n integers $\mathcal{N}(S_i)$ are pairwise coprime is very small as n is large. This means that we may not be able to pass step 3 even after a great deal of trials.

The second reason is in step 4. Since the additional condition (1), or (2) in case K is imaginary, must also hold for pairwise coprime $\mathcal{N}(S_i)$, it is further hard to find out such S_i when p is small. Therefore, we first take an appropriate set S of n elements with pairwise coprime absolute norms, and determine p satisfying condition (1) after that. Then it does not take much time to pass step 4.
We now formula a modification of Algorithm 1 as follows:

Algorithm 4 (Modified Key Generation) Given (n, k, θ), this algorithm outputs a private key $[\theta, g, d, p, S]$ and a public key $[n, k, b]$ by the following steps.

1. Choose random n elements from
$$\left\{\alpha + \beta\omega \mid \alpha,\ \beta \in \mathbb{Z},\ -\frac{\ell}{2} \leq \alpha, \beta \leq \frac{\ell}{2}\right\},$$
where $\ell \in \mathbb{Z}$ is a suitably choosen number for example about $2n$.

2. If $\mathcal{N}(S_1), \cdots, \mathcal{N}(S_n)$ are not pairwise coprime, go to step 1.

3. Choose a rational prime number p so that (p) is a prime ideal of $\mathcal{O}_K$ and satisfing codition (1) or (2).
4. Same as step 5 [DLP part] in Algorithm 1.

Doing like this, we have succeeded in making key generation faster than the original one except the part of the DLP. But, the p's value becomes large, therefore much more time is indeed necessary to solve the DLP.

We now compare Algorithm 4 with Algorithm 1 about key generation except the part of DLP.

Table 3. Input parameter, p's value, execution time of steps 1 – 4 of Algorithm 1 and steps 1 – 3 of Algorithm 4 in some quadratic fields.

Algorithm 1			Algorithm 4		
$[n, k, \theta]$	p's value	Time	$[n, k, \theta]$	p's value	Time
[4, 2, -1]	19	0.008	[4, 2, -1]	39	0.002
[4, 2, -3]	21	0.006	[4, 2, -3]	49	0.00
[4, 2, 3]	51	0.024	[4, 2, 3]	85	0.002
[4, 2, 5]	38	0.02	[4, 2, 5]	39	0.002
[8, 3, - 1]	926	0.312	[8, 3, -1]	1462	0.312
[8, 3, -3]	615	0.232	[8, 3, -3]	2144	0.066
[8, 3, 3]	4163	1.222	[8, 3, 3]	3275	0.204
[8, 3, 5]	1222	0.482	[8, 3, 5]	1836	0.024
[8, 4, -1]	1710	0.598	[8, 4, -1]	9248	0.152
[8, 4, -3]	1704	0.606	[8, 4, -3]	16555	0.03
[8, 4, 3]	18944	5.152	[8, 4, 3]	23610	0.206
[8, 4, 5]	6326	2.498	[8, 4, 5]	18961	0.018
[12, 4, -1]	777395	179.782	[12, 4, -1]	45724	126.518
[12, 4, -3]	53202	14.676	[12, 4, -3]	78710	6.392
[12, 4, 3]	510317	123.484	[12, 4, 3]	279084	99.412
[12, 4, 5]	72517	20.938	[12, 4, 5]	169405	0.912
[12, 5, -1]	908726	224.13	[12, 5, -1]	472096	48.758
[12, 5, -3]	119430	34.232	[12, 5, -3]	1386155	4.132
[12, 5, 3]	3804643	837.79	[12, 5, 3]	5766807	54.058
[12, 5, 5]	631013	155.54	[12, 5, 5]	3411517	1.246

By the table above, we can numerically verify that, except the part of DLP, Algorithm 4 is faster than Algorithm 1 when (n, k) are big. Hence we have solved the second problem.

5.2 Another Key Generation

When n is large, the probability that the norm of random n elements are pairwise coprime is very small as we have seen in Proposition 1. The first problem is not solved and it still takes much time to pass step 1 even by Algorithm 4. The condition of step 2 in Algorithm 4 can be changed choosing n random primes which are not associate. So, we choose primes using the following well-known fact.

Proposition 2 *Let K be an algebraic number field and $\mathcal{O}_K$ be the ring of integers of K . If $p \in \mathcal{O}_K$ and the absolute norm $\mathcal{N}(p)$ is a rational prime, then p is a prime in $\mathcal{O}_K$.*

Algorithm 5 (Another Key Generation) Given (n, k, θ), this algorithm outputs a private key $[\theta, g, d, p, S]$ and a public key $[n, k, b]$ by the following steps.

1. Compute the set of prime elements of degree 1 from

$$\left\{\alpha + \beta\omega \mid -\frac{\ell}{2} \leq \alpha, \beta \leq \frac{\ell}{2}\right\},$$

   where $\ell \in \mathbb{Z}$ is a suitably choosen number for example about $2n$.

2. Choose n random elements from the set above except associate.

3. Same as step 3 in Algorithm 4.

4. Same as step 4 [DLP part] in Algorithm 4.

We now compare Algorithm 5 with Algorithm 4 about key generation except the part of DLP.

Table 4 Input parameter, p's value and execution time in quadratic fields.

Algorithm 4			Algorithm 5		
$[n, k, \theta]$	p's value	Time	$[n, k, \theta]$	p's value	Time
[6, 2, -1]	96	0.012	[6, 2, - 1]	96	0.004
[6, 2, -3]	92	0.01	[6, 2, -3]	92	0.005
[6, 2, 3]	166	0.024	[6, 2, 3]	180	0.086
[6, 2, 5]	111	0.004	[6, 2, 5]	94	0.017
[12, 4, -1]	45724	126.51	[12, 4, -1]	45852	0.018
[12, 4, -3]	78710	6.392	[12, 4, -3]	123176	0.022
[12, 4, 3]	279084	99.412	[12, 4, 3]	189015	0.21
[12, 4, 5]	169405	0.912	[12, 4, 5]	116490	0.066
[18, 6, -1]	–	×	[18, 6, -1]	4919611	0.042
[18, 6, -3]	–	×	[18, 6, -3]	9014287	0.05
[18, 6, 3]	–	×	[18, 6, 3]	54439129	0.724
[18, 6, 5]	–	×	[18, 6, 5]	23400643	0.536
[24, 7, -1]	–	×	[24, 7, -1]	17822592465	0.082
[24, 7, -3]	–	×	[24, 7, - 3]	23398393746	0.097
[24, 7, 3]	–	×	[24, 7, 3]	383543689798	22.808
[24, 7, 5]	–	×	[24, 7, 5]	214851802338	23.274
[30, 8, -1]	–	×	[30, 8, -1]	3233476628170	0.14
[30, 8, -3]	–	×	[30, 8, -3]	4370221432705	0.176
[30, 8, 3]	–	×	[30, 8, 3]	–	×
[30, 8, 5]	–	×	[30, 8, 5]	–	×

By the table above, we can conclude that Algorithm 5 is clearly faster than Algorithm 4 when (n, k) are big. If we observe that the p's value also become smaller, this fact is at least numerically true including the part of DLP.

Remark. It takes, however, much time to pass condition (1) in real quadratic fields even by Algorithm 5.

6 Concluding remark

By our implementaion and experiment, we can summarize our results and future subjects as follows:

- Changing the order of choosing key parameters as above, key genreration is faster than the original one except the part of the DLP.
- Utilizing prime elements of K, key genreration is further faster than the modified one. It is a future problem to compare with the original one including the part of the DLP.
- Although much time is necessary to solve the DLP, if we can once make the key, then we can use Knapsack-type cryptosystem in a classical TM.
- Similar problems can be considerd for general number fields of higher degree.
- Detailed analysis of the practical security is also a future subject.

References

[1] NZMATH Group, `NZMATH`, `http://tnt.math.metro-u.ac.jp/nzmath/`.

[2] T. Okamoto, K. Tanaka and S. Uchiyama, Quantum public-key cryptosystems, CRYPTO 2000, LNCS **1880**, pp. 147–165 (2000).

[3] Peter W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, SIAM J. Comp., Vol. **26**, No. 5, pp. 1484–1509 (1997).

[4] Laszlo Toth, The probabality that k positive integers are pairwise relatively prime, Fibonacci Quart. **40** (2002), 13–18.