Cryptanalysis of the Paeng-Jung-Ha Cryptosystem from PKC 2003

Daewan Han[1], Myung-Hwan Kim[2*], and Yongjin Yeom[1]

[1]National Security Research Institute,
161 Gajeong-dong, Yuseong-gu, Daejeon, 305-350, Korea.
{dwh,yjyeom}@etri.re.kr
[2]Department of Mathematical Science and ISaC-RIM,
Seoul National University, Seoul, 151-747, Korea.
mhkim@math.snu.ac.kr

Abstract. At PKC 2003 Paeng, Jung, and Ha proposed a lattice based public key cryptosystem(PJH). It is originated from GGH, and designed as a hybrid of GGH and NTRUEncrypt in order to reduce the key size. They claimed that PJH is secure against all possible attacks, especially against lattice attacks. However, in this paper, we present a key recovery attack, based on lattice theory, against PJH. The running time of our attack is drastically short. For example, we could recover all secret keys within 10 minutes even for the system with $n = 1001$ on a single PC. Unlike other lattice attacks against NTRUEncrypt and GGH, the attack may be applied well to the system with much larger parameters. We present some clues why we believe so. Based on this belief, we declare that PJH should not be used in practice.

Keywords: Paeng-Jung-Ha cryptosystem, GGH, NTRUEncrypt, Lattice attack

1 Introduction

Since Ajtai's seminal work [1], some lattice-based public-key cryptosystems [2, 4, 5] have been suggested inspired by his work. Among them GGH [4] and NTRU [5] attracted much attention because both systems seemed to be practical with fast encryption/decryption and reasonable key size. GGH is a lattice version of the previously well-known code-based cryptosystems [9]. Though its key size is somewhat large, the system is fast. The proposers claimed that the system with practically usable parameters would be secure. A few years later, however, Nguyen presented a powerful lattice attack against it [12]. In order for GGH to be secure against Nguyen's attack, its key size should be too large to be practical. Thus, GGH has been regarded as a broken system since then. NTRU, more precisely NTRUEncrypt, is another lattice-based system widely reviewed. The system is very efficient and unbroken till now. From the lattice-theoretic point of view, NTRUEncrypt is a special instance of GGH in the sense that the

* The second author was partially supported by KRF(2005-070-C00004).

former uses a circulant matrix for a public key while the latter uses a random square matrix. As a result, the key size of NTRUEncrypt is $O(n)$ while that of GGH is $O(n^2)$, where n is the dimension of the matrix.

A few years after Nguyen's attack, Paeng, Jung, and Ha proposed a variant [18] of GGH, which we will call PJH in this paper. Motivations of developing such a variant are as follows: Firstly, one-way function of GGH has still merits since it is simple and faster than other systems using modular exponentiations. Secondly, at the time that PJH was suggested, it seemed to be easier to design a natural signature scheme based on GGH than on NTRUEncrypt, although both signature schemes turned out to be insecure recently [13]. Thirdly, it seems to be possible to overcome Nguyen's attack by choosing lattices more carefully. With these in mind, they designed PJH as a hybrid type of GGH and NTRUEncrypt: PJH looks similar to GGH except that it takes a partially circulant matrix for a public key. As a result, its key size reduces down to $O(n)$, which is same as that of NTRUEncrypt. Concerning the security of PJH, the proposers claimed that it would be secure against all possible attacks with practical key sizes. Because GGH was broken by a lattice attack, they presented extensive analysis on the security against lattice attacks, and concluded that their system would be secure on the basis of various simulation results.

However, in this paper, we present a key recovery attack against PJH with a lattice technique. In order to recover the secret keys, we induce a linear equation from the public information on key pairs. Then, we construct a lattice from the equation and obtain some of the secret keys by applying lattice reduction algorithms to the lattice. The remaining secret keys can be recovered simply by solving a few linear equations. We could recover secret keys within 10 minutes even for the system with $n = 1001$, where n is a system parameter which will be described in the next section. Unlike other lattice attacks against NTRUEncrypt and GGH, our attack may be applied well to the system of much larger n's. We present some clues why we believe so. Based on this belief, we declare that PJH should not be used in practice.

The rest of this paper is organized as follows. In the next section, we briefly introduce PJH and describe basic principles of general lattice attacks against public key cryptosystems. In Section 3, we present our key recovery attack, simulation results, and applicability of our attack against the system with much larger parameters. Finally, we conclude in Section 4.

2 Preliminaries

2.1 Overview of PJH Cryptosystem

Notations and Parameters Let n be a prime integer, and consider a polynomial ring

$$\mathcal{R} = \mathbb{Z}[x]/\langle x^n - 1\rangle.$$

We identify a polynomial $f(x) = a_0 + a_1x + \cdots + a_{n-1}x^{n-1} \in \mathcal{R}$ with a vector $(a_0, a_1, \cdots, a_{n-1}) \in \mathbb{Z}^n$. We will denote both by f. Note that the multiplication

$f \cdot g$ of f and g is computed by the convolution product of them, that is,

$$h = f \cdot g\ , \qquad c_k = \sum_{i+j=k \bmod n} a_i b_j,$$

where $g = b_0 + b_1 x + \cdots + b_{n-1}x^{n-1}$ and $h = c_0 + c_1 x + \cdots + c_{n-1}x^{n-1}$. For $f = a_0 + a_1 x + \cdots + a_{n-1}x^{n-1} \in \mathcal{R}$ let $\Phi(f)$ be the $n \times n$ circulant matrix

$$\begin{pmatrix} a_0 & a_1 & \cdots & a_{n-1} \\ a_{n-1} & a_0 & \cdots & a_{n-2} \\ \vdots & \vdots & \ddots & \vdots \\ a_1 & a_2 & \cdots & a_0 \end{pmatrix}.$$

Then, it is easy to verify the following:

$$f \cdot g = g \cdot f = f\Phi(g) = g\Phi(f).$$

Finally, we remark that we will use row-oriented notations in matrix representations of lattices while PJH adopts column-oriented rotations in [18].

Key Generation In order to generate a private key, PJH generates 4 polynomials $f_1, f_2, h_1, h_2 \in \mathcal{R}$ which have the following properties:

- $f_1(x) = \alpha_0 + \alpha_1 x^1 + \cdots + \alpha_{n-1}x^{n-1}$ and $f_2(x) = \beta_0 + \beta_1 x^1 + \cdots + \beta_{n-1}x^{n-1}$, where $|\alpha_0|, |\beta_0| \approx \sqrt{2n}$ and the other coefficients are elements in $\{-1, 0, 1\}$.
- The coefficients of h_1 and h_2 are elements in $\{-1, 0, 1\}$.

The private key R is defined by

$$R := \begin{pmatrix} \Phi(f_1) & \Phi(h_2) \\ \Phi(h_1) & \Phi(f_2) \end{pmatrix}.$$

Let p be a positive integer. In [18] p is a 10-bit or 80-bit integer which need not be prime. Proposers recommended that p is kept secret although it does not affect the security. In order to generate public keys, PJH chooses $g \in \mathcal{R}$ with coefficients in $(-p/2, p/2]$ such that g is invertible in $\mathbb{Z}_p[x]/\langle x^n - 1\rangle$. Then there exists g_p with coefficients in $(-p/2, p/2]$ and Q in $\mathcal{R}$ such that

$$g \cdot g_p - 1 = pQ \in \mathcal{R}.$$

Now, four public polynomials $P_1, P_2, P_3, P_4 \in \mathcal{R}$ are defined as follows:

$$\begin{aligned} P_1 &:= f_1 \cdot g + h_1 \cdot Q, \\ P_2 &:= pf_1 + h_1 \cdot g_p, \\ P_3 &:= h_2 \cdot g + f_2 \cdot Q, \\ P_4 &:= ph_2 + f_2 \cdot g_p. \end{aligned} \tag{1}$$

Table 1. Comparison of key sizes(KB) of PJH and GGH

rank of B	PJH(10-bit p)	PJH(80-bit p)	GGH	GGH(HNF)
200	0.85	4.4	330	32
300	1.4	6.6	990	75
400	1.8	8.8	2370	140
500	2.3	11		

The public key B is defined by

$$B := \begin{pmatrix} \Phi(P_1) & \Phi(P_3) \\ \Phi(P_2) & \Phi(P_4) \end{pmatrix}.$$

The pair (R, B) constructed in this way have the same properties as that in GGH. That is, R and B are different bases of a same lattice and have low and high dual-orthogonality defects, respectively. For more details, we refer the readers to [18].

Encryption and Decryption For a message $m = (m_1, m_2) \in \mathcal{R}^2$, the ciphertext c is calculated by

$$c = (c_1, c_2) = mB + e \in (\mathbb{Q}[x]/\langle x^n - 1\rangle)^2$$

for an error vector $e = (e_1, e_2) \in (\mathbb{Q}[x]/\langle x^n - 1\rangle)^2$, where the coefficients of e_i are elements in $\{-1/2, 1/2\}$.

Let T be a matrix defined by

$$T := \begin{pmatrix} \Phi(g) & \Phi(Q) \\ pI & \Phi(g_p) \end{pmatrix}^{-1}.$$

Then, c can be decrypted as follows:

$$m = (m_1, m_2) = \lceil cR^{-1} \rfloor T.$$

The decryption works similarly to that of GGH. Since we don't have to understand thoroughly how the decryption works in order to explain our attack, we omit the details.

Efficiency and Security Since the public matrix B is determined by four polynomials in $\mathcal{R}$, key size of PJH is $O(n)$ while that of GGH is $O(n^2)$. The comparison of key sizes of PJH with GGH in [18] is given in Table 1. The values in the last column are the key size of GGH when it uses Micciancio's HNF expression [10].

Concerning the security of PJH, the proposers claimed that the system is secure against all possible attacks even if the parameter p is disclosed. Because

Table 2. Running times to break PJH estimated by the proposers

n	expected run time
211	$1.46\ \times 10^{9}$ seconds ≈ 46 years
257	$1.45\ \times 10^{11}$ seconds $\approx 4.6 \times 10^{3}$ years
373	$1.58\ \times 10^{16}$ seconds $\approx 5 \times 10^{8}$ years
503	$5.18\ \times 10^{21}$ seconds $\approx 1.6 \times 10^{14}$ years

GGH was broken by a lattice attack, they presented extensive analysis on the security against lattice attacks. We briefly describe their analysis. Because the equations in (1), which are the only public information on secret keys and public keys, are quadratic of unknown variables in $\mathcal{R}$, they expected that no key recovery attack using lattice techniques would be feasible. On the other hand, a message recovery attack seems to be feasible. However, they claimed that the reduction algorithm would work badly because they made *expected-gaps*, which is defined in the next subsection, small. Running times to break PJH estimated by the authors of [18] are given in Table 2.

2.2 Lattice Attacks

A *Lattice* $\mathcal{L}$ is defined to be a discrete subgroup of $\mathbb{R}^n$. It consists of all integral linear combinations of a set of some linearly independent vectors $\mathbf{b}_1, \mathbf{b}_2, \cdots, \mathbf{b}_d \in \mathbb{R}^n$. Such a set of vectors is called a lattice *basis*. All the bases of a lattice have the same number of elements, and this number is called the *dimension* of the lattice, denoted by $\dim(\mathcal{L})$. There are infinitely many bases for $\mathcal{L}$ when $\dim(\mathcal{L}) \geq 2$. Any two bases of a lattice $\mathcal{L}$ are related to each other by some unimodular matrix (integral matrix of determinant ± 1), and therefore all the bases share the same Gramian determinant $\det_{1\leq i,j\leq d}\langle \mathbf{b}_i, \mathbf{b}_j\rangle$, where $\{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_d\}$ is a basis of $\mathcal{L}$. The *volume* $\mathrm{vol}(\mathcal{L})$ of $\mathcal{L}$ is defined by the square root of the Gramian determinant, which is the d-dimensional volume of the parallelepiped spanned by the $\mathbf{b}_i$'s. Since a lattice is discrete, it has a shortest non-zero vector. The Euclidean length of a shortest vector of $\mathcal{L}$ is called the *first minimum*, denoted by $\lambda_1(\mathcal{L})$. More generally, for all $1 \leq i \leq \dim(\mathcal{L})$, the *i-th minimum* $\lambda_i(\mathcal{L})$ is defined by the minimum of $\max_{1\leq j\leq i} \|\mathbf{v}_j\|$, where $\{\mathbf{v}_1, \cdots, \mathbf{v}_i\}$ runs over all possible sets of linearly independent vectors $\mathbf{v}_1, \cdots, \mathbf{v}_i \in \mathcal{L}$. The ratio λ_2/λ_1 is called *lattice gap*, which is useful in estimating the feasibility of lattice attacks.

Given a basis for a lattice $\mathcal{L}$, the problem of finding a shortest vector of $\mathcal{L}$ is called the *shortest vector problem* (SVP). Another famous problem related to lattices is the *closest vector problem* (CVP), the problem of finding a vector $\mathbf{v} \in \mathcal{L}$ which is closest to a given vector $\mathbf{t} \in \mathbb{R}^n$. The CVP in ℓ_p-norm is proved to be NP-hard for all p, and the SVP is also believed to be hard [11]. Indeed, there are no known polynomial time algorithms to solve even the approximated versions of them, if approximation factors are polynomials in $\dim(\mathcal{L})$. However, if the dimension of a lattice is less than a few hundreds, we can solve them in practice using lattice reduction algorithms such as LLL [8] and its variants [19].

For a given lattice basis $\{\mathbf{b}_1, \cdots, \mathbf{b}_n\}$, LLL outputs a reduced basis $\{\mathbf{b}_1^*, \cdots, \mathbf{b}_n^*\}$ satisfying $\|\mathbf{b}_1^*\| \leq 2^{(n-1)/2}\lambda_1(\mathcal{L})$ within $O(n^4 \log B)$ integer arithmetic operations, where B is the maximum of $\|\mathbf{b}_i\|^2$'s. The fastest LLL variant known has bit-complexity essentially $O(n^5 \log^2 B)$ [14]. However, the real performances of LLL and its variants are more better than what are expected from the theory, both in terms of the running time and the output quality [15]. Thus, we can find a genuine shortest vector of a given lattice using the algorithms when the dimension of the lattices are less than a few hundreds.

Lattices have been widely used in attacking public-key cryptosystems. Readers are referred to [16] for well-written summary of the various results about them. Here we briefly describe basic principles to attack systems using lattices. The attacks are accomplished by reducing the problem of finding a secret information in the system to a specific instance of SVP or CVP. We describe the SVP case here. First, one constructs a lattice from public information such as system parameters and public keys. Then, one shows that a relatively short vector $\mathbf{v}$ of the lattice includes the secret information we want to obtain, and finds such $\mathbf{v}$ by solving SVP with lattice reduction algorithms. However, in most cases, one cannot prove that $\mathbf{v}$ is a genuine shortest vector of the lattice. Instead, one can infer that it is a shortest vector by using Gaussian heuristic: Given a random lattice $\mathcal{L}$ of dimension n, the first minimum $\lambda_1(\mathcal{L})$ of $\mathcal{L}$ will be

$$\sqrt{\frac{n}{2\pi e}}\mathrm{vol}(\mathcal{L})^{\frac{1}{n}} \leq \lambda_1(\mathcal{L}) \leq \sqrt{\frac{n}{\pi e}}\mathrm{vol}(\mathcal{L})^{\frac{1}{n}}.$$

One use $\sigma(\mathcal{L})$ as the estimation of $\lambda_1(\mathcal{L})$, where $\sigma(\mathcal{L})$ (briefly σ) is defined by

$$\sigma(\mathcal{L}) := \sqrt{\frac{n}{2\pi e}}\mathrm{vol}(\mathcal{L})^{\frac{1}{n}}.$$

If $\|\mathbf{v}\|$ is less than σ, one may expect that $\mathbf{v}$ is a shortest vector. In practice, the larger the ratio $\sigma/\|\mathbf{v}\|$ is, the easier we can find $\mathbf{v}$. We call this ratio an *expected-gap* of $\mathcal{L}$ with respect to $\mathbf{v}$.

3 Key Recovery Attack against PJH

3.1 Volume of the Lattice Generated by $\Phi(P_2)$

For a polynomial $f \in \mathcal{R}$, let's define $V(f)$ by the volume of the lattice generated by the circulant matrix $\Phi(f)$. In our attack, $V(P_2)$ is used essentially: More precisely, we need to estimate a reasonable lower bound of $\delta_n(P_2)$, which is defined for randomly chosen P_2 by

$$\delta_n(P_2) := p^{-1}\sqrt{\frac{1}{2\pi e}}V(P_2)^{\frac{1}{n+1}}.$$

In this subsection we present a heuristic on the asymptotic estimation of $\delta_n(P_2)$.

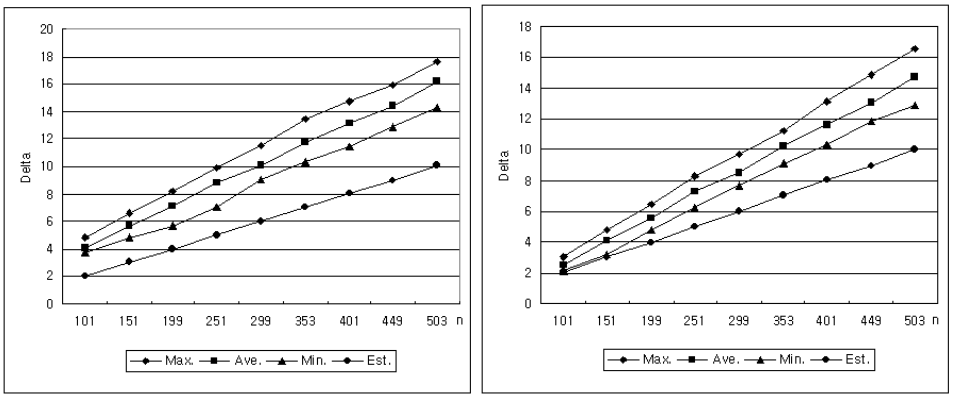

Fig. 1. Experimental estimation of $\delta_n(P_2)$ for 10-bit p(left) and 80-bit p(right)

To our knowledge, it is difficult to understand asymptotic behavior of $V(f)$ theoretically. For simplicity, suppose that the lattice is of full rank. In that case, $V(f)$ is equal to the determinant of the circulant matrix $\Phi(f)$. So, we can infer intuitively

$$V(f) = \det(\Phi(f)) = \frac{\|f\|^n}{\text{orth-defect}(\Phi(f))} \sim O(\|f\|^n).$$

However, if there are no conditions on n and f, there are no known theoretical results on asymptotic properties of $V(f)$. Moreover, we could not find meaningful characteristics even from simulations.

Now, turn to our attention to $\delta_n(P_2)$ in PJH. Because PJH uses prime n and P_2 is obtained by special formulas, $V(P_2)$ and $\delta_n(P_2)$ behave regularly as n increases. We could verify this by simulations. We calculated $\delta_n(P_2)$ in randomly constructed PJH for several n's and for several 10-bit or 80-bit p's. We tested 100 times for each n and p. The experimental results are shown in Figure 1. The upper three curves in the figure show the simulated maximum, average, and minimum values of $\delta_n(P_2)$, respectively. They tend to increase linearly as n increases. If the bit-size of p increases, the value of $\delta_n(P_2)$ decreases a little bit. However, as one can see in Figure 1, the slopes of $\delta_n(P_2)$ do not change much as the bit-size of p increases.

From our simulations, we estimate a lower bound of $\delta_n(P_2)$ for large n and for $p \leq 2^{80}$ very conservatively as follows:

$$\delta_n(P_2) \geq 0.02n \quad \text{ for } n \geq 100. \tag{2}$$

The lowest line in the figure shows the lower bound in (2). In the next subsections, we will use this estimation for theoretical analysis of our attack against PJH.

3.2 A Linear Relation Between Key Pairs

Let's recall some of equations in (1).

$$P_1 = f_1 \cdot g + h_1 \cdot Q \tag{3}$$

$$P_2 = pf_1 + h_1 \cdot g_p \tag{4}$$

By multiplying g to (4), we induce the following equation in $\mathcal{R}$:

$$\begin{aligned} g \cdot P_2 &= pf_1 \cdot g + h_1 \cdot g_p \cdot g \\ &= pf_1 \cdot g + h_1 \cdot (1 + pQ) \\ &= p(f_1 \cdot g + h_1 \cdot Q) + h_1 \\ &= pP_1 + h_1. \end{aligned} \tag{5}$$

In (5), g, p and h_1 are unknown variables. However, note that the equation is linear while those in (1) are quadratic. Suppose we can recover h_1 and p. Then, g can be recovered by solving the linear equation (5). From g and p we can obtain g_p and Q easily. Then, (1) becomes a system of four linear equations of four unknown variables so that the other secret keys can be recovered easily. Thus, if we can find h_1 and p, we can recover all secret keys of PJH. We will recover them using a lattice technique in the following subsections.

Remark 1. PJH can be designed flexibly. In [18], the authors introduced another scheme which uses the polynomial ring $\mathbb{Z}[x]/\langle x^n - x - 1\rangle$ instead of $\mathbb{Z}[x]/\langle x^n - 1\rangle$ in Section 4.4, and one more in the appendix. However, our attack can be applied identically to the one in Section 4.4, and a modified attack, using the equation

$$g \cdot P_{12} = pP_{11} + h_1$$

instead of (5), can be applied to the one in the appendix.

3.3 Finding h_1 with a Lattice Technique

The Case When p is not Secret Consider a lattice $\mathcal{L}_1$ generated by rows of the following $(n+1) \times (n+1)$ matrix L_1:

$$L_1 = \begin{pmatrix} \Phi(P_2) & 0_n \\ pP_1 & 1 \end{pmatrix},$$

where 0_n is a column vector of dimension n whose entries are all 0. Then, a vector

$$\mathbf{v}_1 = (h_1, -1) = (g \cdot P_2 - pP_1, -1) = (g, -1)L_1$$

is contained in $\mathcal{L}_1$ and its length satisfies

$$\|\mathbf{v}_1\| = \sqrt{\|h_1\|^2 + 1} \le \sqrt{n+1}.$$

Table 3. Breaking times (in seconds) of PJH for 80-bit public p.

Dimension of lattice	n =211	n =257	n =373	n =503	$n = 1001$
Time(Seconds)	< 1	< 1	< 10	< 10	< 100

According to Gaussian heuristic we can expect

$$\sigma_1 \sim \sqrt{\frac{n+1}{2\pi e}} \mathrm{vol}(\mathcal{L}_1)^{\frac{1}{n+1}} = \delta_n(P_2)p\sqrt{n+1},$$

where σ_1 is the length of a shortest vector in $\mathcal{L}_1$. Using the approximation of $\delta_n(P_2)$ in (2), we can estimate σ_1 as follows:

$$\sigma_1 \geq 0.02np\sqrt{n+1} \quad \text{for } n \geq 100.$$

Thus, the expected-gap of $\mathcal{L}_1$ with respect to $\mathbf{v}_1$ is bigger than or equal to $0.02np$, i.e.,

$$\frac{\sigma_1}{\|\mathbf{v}_1\|} \geq 0.02np \quad \text{for } n \geq 100. \tag{6}$$

Since this is very large in PJH parameters, where p is a 10-bit or 80-bit prime, $\mathbf{v}_1$ will be a shortest vector of $\mathcal{L}_1$ with high probability. So, we can easily find $\mathbf{v}_1$ (and hence h_1) by using lattice reduction algorithms.

We simulated the above attack on a Pentium IV 3.2 GHz PC using the floating-point variant of LLL algorithm (LLL_FP) with default parameters implemented in NTL package [17]. We tested our attack against PJH with different n's for 10-bit and 80-bit randomly selected p's. For each n and p, 10 different instances were tested against. We could obtain $\mathbf{v}$ for all instances. For 10-bit p's, the running times for lattice reduction were too short to be described. The running times for 80-bit p's are given in Table 3. Even for $n = 1001$, we could find the solutions within 100 seconds.

The Case When p is Secret If p is secret, we cannot construct the lattice $\mathcal{L}_1$. Instead, we consider a lattice $\mathcal{L}_2$ generated by the following matrix L_2:

$$L_2 = \begin{pmatrix} \Phi(P_2) & 0_n \\ P_1 & 1 \end{pmatrix}.$$

Then, a vector $\mathbf{v}_2 = (h_1, -p)$ is contained in $\mathcal{L}_2$ and its length is smaller than or equal to $\sqrt{p^2+n^2}$. The σ_2 corresponding to $\mathcal{L}_2$ is equal to σ_1, and hence $\sigma_2 \geq 0.02np\sqrt{n+1}$. Thus, we get

$$\frac{\sigma_2}{\|\mathbf{v}_2\|} \geq \frac{0.02np\sqrt{n+1}}{\sqrt{p^2+n^2}} \sim 0.02n\sqrt{n}. \tag{7}$$

Although this value is smaller than the expected-gap when p is public, it is still large (see Table 5). So, we can find $\mathbf{v}_2$ (and hence h_1, and p) by using lattice reduction algorithms.

Table 4. Breaking times (in seconds) of PJH for 80-bit secret p.

Dimension of lattice	n =211	n =257	n =373	n =503	$n = 1001$
Times(second)	< 20	< 30	< 60	< 100	< 500

Table 5. Expected-gaps in attacks against NTRUEncrypt, GGH, and PJH.

Dimension of the lattice	200	250	300	350	400	450	500
GGH [12]	9.7	9.4	9.5	9.4	9.6		
NTRUEncrypt [7]	5.7	6.3	6.9	7.5	8.0	8.5	8.9
PJH(secret p) [From (7)]	56.6	79.1	103.9	131.0	160.0	190.9	223.6
PJH(public p) [From (6)]	$4p$	$5p$	$6p$	$7p$	$8p$	$9p$	$10p$

We also simulated the above attack on the same machine using the same algorithm as in the previous case. The results for 80-bit p's are given in Table 4. The running times for lattice reduction in this case were more longer than those in case of public p. This was expected because the expected-gap of the former is smaller than that of the latter. Still, we could find the solutions within 500 seconds even for $n = 1001$.

Remark 2. We can use another lattice in our attack to get larger expected-gap. Let d_p be the bit-length of p, and consider a vector $\mathbf{v}_3 = (2^{d_p}h_1, -p)$ and a lattice $\mathcal{L}_3$ generated by the following matrix L_3:

$$L_3 = \begin{pmatrix} 2^{d_p}\Phi(P_2) & 0_n \\ 2^{d_p}P_1 & 1 \end{pmatrix}.$$

Then, $\mathbf{v}_3$ is a short vector of $\mathcal{L}_3$ and the expected-gap of $\mathcal{L}_3$ with respect to $\mathbf{v}_3$ is about $\delta_n(P_2)p$. However, since the entries of L_3 is larger than those of L_2, it takes more time in lattice reduction for $\mathcal{L}_3$ than for $\mathcal{L}_2$. Thus, using L_2 is more efficient in practice.

3.4 Attack against PJH with Larger Parameters

In attack against GGH, expected-gaps are small and do not increase as the lattice dimension n increases [12], and in attack against NTRUEncrypt, they increase but are bounded by about $0.25\sqrt{n}$ [7]. On the other hand, the efficiency of reduction algorithm becomes worse as n increases. These two facts cause the difficulty in attacking GGH and NTRUEncrypt when n is sufficiently large [3, 6, 12]. The attacks are not possible practically when n is more than $400 \sim 500$.

However, expected-gaps in our attack against PJH are much large and increase very fast as n increases. The comparison of expected-gaps in attacks against GGH, NTRUEncrypt, and PJH are given in Table 5. The large expected-gaps explains why breaking times of PJH are shorter compared to other systems.

Moreover, the fact that the expected-gaps increases fast compensates the inefficiency of reduction algorithms for large n. Thus, we expect that we can break PJH until n grows too large to be used practically.

4 Conclusion

We have shown that Paeng-Jung-Ha cryptosystem proposed at PKC 2003 is not secure against a lattice attack contrary to proposer's expectation. From the relations between public keys and secret keys, we could induce a linear equation useful for a lattice attack. Because the breaking times for suggested parameters are drastically short and the feasibility of our attack against the system with larger parameters is high, we may declare that the system should not be used practically.

It seems to be hard to modify PJH to be secure against our attack without worsening the efficiency. Our result shows that, although lattice-based cryptosystems look attractive, it is difficult to design a practical system other than NTRUEncrypt.

Acknowledgements. The authors would like to thank the anonymous referees for pointing out some errors of this paper and giving valuable comments.

References

1. M. Ajtai. Generating Hard Instances of Lattice Problems. In Proc. of 28th ACM STOC, 99-108, 1996.
2. M. Ajtai and C. Dwork. A Public-key Cryptosystem with Worst-case/Average-case Equivalence. In Proc. of 29th ACM STOC, 284-293, 1997.
3. D. Coppersmith and A. Shamir. Lattice Attacks on NTRU. In Proc. of Eurocrypt'97, LNCS 1233, 52-61, Springer-Verlag, 1997.
4. G. Goldreich, S. Goldwasser, and S. Halevi. Public-key Cryptosystems from Lattice Reduction Problems. In Proc. of Crypto '97, LNCS 1294, 112-131, Springer-Verlag, 1997.
5. J. Hoffstein, J. Pipher, and J. H. Silverman. NTRU: A Ring-Based Public Key Cryptosystem. In Proc. of ANTS III, LNCS 1423, 267-288, Springer-Verlag, 1998.
6. J. Hoffstein, J. H. Silverman, and W. Whyte. Estimated Breaking Times for NTRU Lattices. Technical Report #12(Version 2), NTRU Cryptosystems, 2003.
7. N. Howgrave-Graham, J. H. Silverman, and W. Whyte. Choosing Parameter Sets for NTRUEncrypt with NAEP and SVES-3. In Proc. of CT-RSA 2005, LNCS 3376, 118-135, Springer-Verlag, 2005.
8. A. K. Lenstra, H. W. Lenstra Jr, and L. Lovász. Factoring Polynomials with Rational Coefficients. Mathematische Ann. 261, 513-534, 1982.
9. R. J. McEliece. A public-key Cryptosystem Based on Algebraic Coding Theory. DSN Prog. Rep., Jet Prop. Lab., California Inst. Technol., Pasadena, CA, pages 114–116, January 1978.
10. D. Micciancio. Improving Lattice Based Cryptosystems Using the Hermite Normal Form. In Proc. of CaLC 2001, LNCS 2146, 126-145, Springer-Verlag, 2001.

11. D. Micciancio and S. Goldwasser. *Complexity of lattice problems : A Cryptographic perspective*. Kluwer Academic Publishers, 2002.
12. P. Q. Nguyen. Cryptanalysis of the Goldreich-Goldwasser-Halevi Cryptosystem from Crypto '97. In Proc. of Crypto '99, LNCS 1666, 288-304, Springer-Verlag, 1999.
13. P. Q. Nguyen and O. Regev. Learning a Parallelepiped: Cryptanalysis of GGH and NTRU Signatures. In Proc. of Eurocrypt'06, LNCS 4004, 271-288, Springer-Verlag, 2006.
14. P. Q. Nguyen and D. Stehlé. Floating-point LLL revisited. In Proc. of Eurocrypt 2005, LNCS 3494, 215-233, Springer-Verlag, 2005.
15. P. Q. Nguyen and D. Stehlé. LLL on the Average. In Proc. of ANTS VII, LNCS 4076, 238-256, Springer-Verlag, 2006.
16. P. Q. Nguyen and J. Stern. The Two Faces of Lattices in Cryptology. In Proc. of CaLC 2001, LNCS 2146, 146-180, Springer-Verlag, 2001.
17. NTL - A Number Theory Library. Available at `http://shoup.net/ntl`.
18. S. Paeng, B. E. Jung, and K. Ha. A Lattice Based Public Key Cryptosystem Using Polynomial Representations. In the Proc. of PKC 2003, LNCS 2567, 292-308, Springer-Verlag, 2003.
19. C. P. Schnorr. A Hierarchy of Polynomial Time Lattice Basis Reduction Algorithms. Theoretical Computer Science 53, 201-224, 1987.