Cryptanalysis of a public key cryptosystem based on Diophantine equations via weighted LLL reduction

Jintai Ding* Momonari Kudo† Shinya Okumura‡
Tsuyoshi Takagi‡ Chengdong Tao§

December 23, 2015

Abstract

In this paper, we give an attack against a public key cryptosystem based on Diophantine equations of degree increasing type (DEC) proposed by the third author ([Oku15]). We show that the security of DEC depends on the difficulty of finding special (relatively) short vectors in some lattices obtained from a public key and a ciphertext. The most important target vector in our attack is not necessarily a shortest vector in a lattice of low rank but only some entries are relatively small. In our attack, the LLL algorithm does not work well for finding such vectors.

The technical point of our method is to change a norm dealt with in the usual LLL algorithm from the Euclidean norm to a special norm called a weighted norm. We call the LLL algorithm with respect to a weighted norm the *"weighted LLL algorithm"* in this paper. Our heuristic analysis suggests that the most important target vector in our attack becomes a shorter vector with respect to a weighted norm for an appropriate weight among the vectors in the lattice of low rank. Our experimental results by a standard PC with Magma suggest that our attack with the weighted LLL algorithm can recover a plaintext without finding a secret key for 128 bit security proposed in [Oku15] with sufficiently high probability.

Key words— Weighted LLL reduction, Public-key cryrtosystem, Post-quantum cryptosystem, Diophantine equation

*Department of Mathematical Sciences, University of Cincinnati
†Graduate School of Mathematics, Kyushu University, E-mail: m-kudo@math.kyushu-u.ac.jp
‡Institute of Mathematics for Industry, Kyushu University, E-mail: s-okumura@imi.kyushu-u.ac.jp
§South China University of Technology

Contents

1 Introduction

The Post-Quantum Cryptography has been studied actively since Shor proposed quantum algorithms for factorizing integers and solving the discrete logarithm problem over finite groups in polynomial time ([Sho97]). So far, various candidates of post-quantum cryptosystems have been proposed, e.g. lattice-based cryptosystems, code-based cryptosystems and multivariate public key cryptosystems (see [BBD08] and [DGS06] for details). In order to construct such cryptosystems, we need computationally-hard problems which are infeasible to solve even with quantum computers. The Diophantine problem is expected to be one of such problems. (The Diophantine problem here means that for given multivariate polynomials with integer coefficients, find common integral or rational zeros of them.) Indeed, a public key cryptosystem ([LCL95]) and key exchange protocols ([BHHKP14], [HP13], [Yos11]) which are based on the difficulty of the Diophantine problem have been already proposed. However, the one-wayness of the cryptosystem in [LCL95] is transformed to solving certain linear congruences, and thus the cryptosystem is broken in polynomial time ([Cus95]). Moreover, as for the key exchange protocols in [BHHKP14], [HP13] and [Yos11], one can get secret keys in these protocols by performing them a few times ([HP13], Proposition 2), and these three protocols are said to be impractical.

The Diophantine problem can be generalized to problems over arbitrary rings (the Diophantine problem mentioned above is the problem over $\mathbb{Z}$). The problems over some rings are proved to be unsolvable ([Vid94], [DMR76], [Eis07], [Phe91]) in general. The Algebraic Surface Cryptosystem (ASC) proposed in [AGM09] is based on the difficulty of the section finding problem, which can be viewed as the Diophantine problem over a global function field. In [AGM09], it is analyzed that ASC may have resistance to all the known attacks ([Iwa08], [UT07], [Vol07]) against the previous versions of ASC ([AG04], [AG06], [AG08]). However, the one-wayness of ASC is broken by the ideal decomposition attack proposed in [FS10].

In [Oku15], Okumura proposed a public key cryptosystem based on the difficulty of solving Diophantine equations of degree increasing type over $\mathbb{Z}$ (see Section 3 in this paper for the definition of a polynomial of degree increasing type). We refer to the cryptosystem as DEC for short. In Remark 3.2 of [Oku15], Okumura showed that Diophantine equations of degree increasing type are generally unsolvable. DEC is proposed as a number field analogue of ASC and a candidate of post-quantum cryptosytem. One of advantages of DEC is that sizes of a public key and a secret key are smaller than other candidates of post-quantum cryptosystem (see Table 4 in [Oku15] and Table 2 in this paper). The main idea of avoiding the analogues of the attacks including the ideal decomposition attack against ASC is to twist a plaintext by using some modular arithmetic and to use some random polynomials with large coefficients. In Section 4 of [Oku15], it is analyzed that by the above idea, the number of possible parameters increases, and thus finding the correct plaintext will become infeasible. In addition, the reason why polynomials of degree increasing type are used in DEC is to decode a plaintext uniquely even if the plaintext is twisted. In Section 3, we give a brief review of DEC and the recommended parameters for DEC.

1.1 Our Contribution

In this paper, we propose an attack against DEC. We show that the one-wayness of DEC can be transformed to a problem of finding special relatively short vectors in lattices obtained from a public key and a ciphertext. Our attack can be divided roughly into three steps. In each step, we have a linear system and need to find its appropriate solution, which is equivalent to a problem of

finding an appropriate vector in the lattice obtained by solving the linear system. We use a solution obtained in the first (resp. second) step to construct a linear system in the second (resp. third) step. After finding appropriate solutions of the linear systems in all the steps, it is possible to recover a plaintext with sufficiently high probability by applying Babai's nearest plane algorithm ([Bab86]) and some modular arithmetic.

As we will see in Section 4, if we find a correct solution in the first step, then we can break DEC with sufficiently high probability. More precisely, after we find a correct solution in the first step, we can solve the linear systems in the second and third steps (note that in the third step, we may use an incorrect solution obtained in the second step). Thus, the key point of our attack is whether the first step succeeds or not.

The lattice obtained in the first step of our attack has low rank, (e.g. 3-rank in many cases), and thus finding a target vector in the lattice seems to be performed by basis reduction algorithms such as the LLL algorithm ([LLL82]). However, in Section 4.3, we show an example that the usual LLL algorithm fails in finding the target vector in the first step. Our heuristic analysis on the failure of the example is as follows: the target vector is not necessarily shortest in the lattice of low rank but only some entries are relatively small, i.e., the target vector is a relatively short vector with entries of unbalanced sizes.

1.2 Weighted LLL

In order to deal with such situations, we apply the *weighted LLL algorithm*, which is the LLL algorithm with respect to a special norm called *weighted norm* for some weight, to our attack. We find heuristically a new weighted norm so that the target vector becomes the (or nearly) shortest in some lattice of low rank with respect to this new norm. By a weighted norm for a vector $\mathbf{X} = (x_1, ..., x_n)$, we means the norm:

$$\|\mathbf{X}\| = \sqrt{\sum \alpha_i^2 x_i^2},$$

where α_i are positive real numbers, which we call the weight factors. The idea of weighted LLL algorithm is known ([Mag]), but not yet widely used in cryptography. For detailed description of the weighted LLL algorithm, see Section 2.

1.3 Experimental Verification of Our Attack

In our attack, we assume that the target vector in the first step is a shortest vector with respect to a weighted norm for some weight chosen appropriately by a heuristic way. Our experimental results in Section 6 show that we can find correct vectors in the first step with probability being about from 70 to 90% for recommended parameters in Section 3 via the weighted LLL algorithm. Moreover, our experimental results also show that we can break the one-wayness of DEC with probability being about from 20 to 30%. From this and the complexity analysis on our attack, we infer that our attack can break DEC in polynomial time for all the parameters with sufficiently high probability.

This paper is organized as follows: In Section 2, we give the definition of a weighted norm and describe the weighted LLL algorithm and a situation that it works well. In Section 3, we give a brief review of DEC. In Section 4, we describe the outline and some assumptions of our attack, and we also give a toy example to illustrate our attack completely. In Section 5, we give the complexity analysis on our attack. In Section 6, we give some experimental results on the attack.

Acknowledgements The authors deeply thank Professor Shun'ichi Yokoyama for many helpful comments, corrections, suggestions on this research, discussions in the implementations on Magma. The authors also thank Professor Masaya Yasuda, the supervisor of the second author, for helpful comments, corrections. This work was supported by CREST, JST.

Notation

Throughout this paper, we denote the polynomial ring with n variables over a ring R by $R[\underline{x}] := R[x_1, \ldots, x_n]$. For every $\underline{i} = (i_1, \ldots, i_n) \in (\mathbb{Z}_{\geq 0})^n$ and $(a_1, \ldots, a_n) \in R^n$, we denote the element $a_1^{i_1} \cdots a_n^{i_n} \in R$, the monomial $x_1^{i_1} \cdots x_n^{i_n} \in R[\underline{x}]$ and the value $\sum_{k=1}^{n} i_k$ by $\underline{a}^{\underline{i}}$, $\underline{x}^{\underline{i}}$ and $\sum \underline{i}$, respectively. We can write any element $f(\underline{x}) = f(x_1, \ldots, x_n) \in R[\underline{x}] \setminus \{0\}$ (sometimes we also write f simply) in a unique way as a sum of terms:

$$f(\underline{x}) = \sum_{\underline{i} \in \Lambda} c_{\underline{i}} \underline{x}^{\underline{i}},$$

where Λ is the finite subset of $(\mathbb{Z}_{\geq 0})^n$ and $c_{\underline{i}} \in R \setminus \{0\}$ for $\underline{i} \in \Lambda$. We then write $c_{\underline{i}}(f) := c_{\underline{i}}$ for $\underline{i} \in \Lambda_f := \Lambda$. We call Λ_f the support of f. We denote the total degree of f by w_f. For every element $\underline{a} = (a_1, \ldots, a_n) \in R^n$ and invertible element $d \in R^{\times}$, we denote the element $(a_1/d, \ldots, a_n/d) \in R^n$ by $\underline{a}/d$. Then we denote the value of $f(\underline{x})$ at $\underline{a}/d$ by $f(a_1/d, \ldots, a_n/d)$ or $f(\underline{a}/d)$. In addition, if $R = \mathbb{Z}$ or $\mathbb{Q}$, then we use the following notation:

$$\begin{aligned} \Gamma_f &:= \{(\underline{i}, b_{\underline{i}}) \in \Lambda_f \times \mathbb{Z}_{>0} \ ; \ 2^{b_{\underline{i}}-1} \leq |c_{\underline{i}}(f)| < 2^{b_{\underline{i}}}\}, \\ H(f) &:= \max\{|c_{\underline{i}}(f)| \ ; \ \underline{i} \in \Lambda_f\}. \end{aligned}$$

We call $H(f)$ the height of f. In addition, if for a polynomial $f \in \mathbb{Z}[\underline{x}]$, the support $\Lambda_f = (\underline{i}_1, \ldots, \underline{i}_q)$ is ordered by the order coming from the lexicographical order on the monomials of f, then we denote the sequence of the ordered coefficients of f by the vector $\mathbf{f} = \left(c_{\underline{i}_1}(f), \ldots, c_{\underline{i}_q}(f)\right)$.

An m-dimensional lattice is defined as a discrete additive subgroup of an m-dimensional vector space over $\mathbb{R}$. It is well-known that for any lattice $\mathcal{L}$, there exist $\mathbb{R}$-lineary independent vectors generating $\mathcal{L}$ as a $\mathbb{Z}$-module. The rank of $\mathcal{L}$ is its rank as a $\mathbb{Z}$-module. For any lattice in $\mathbb{R}^m$ and its basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_r\}$, let U be an $r \times m$ matrix whose i-th row vector coincides with $\mathbf{b}_i$. Then we call U the basis matrix of the lattice. Let $\langle \cdot, \cdot \rangle : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ be the natural inner product for some $n \in \mathbb{Z}_{>0}$. For a vector $\mathbf{v} \in \mathbb{R}^n$, we denote the Euclidean norm of $\mathbf{v}$ by $\|\mathbf{v}\|$. We define the rounding function $\lfloor \cdot \rceil : \mathbb{R} \to \mathbb{Z}$ as $\lfloor c \rceil := \lfloor c + \frac{1}{2} \rfloor$ for any $c \in \mathbb{R}$. Let M be an $m \times n$ matrix over $\mathbb{Z}$ and φ_M the homomorphism as additive groups between $\mathbb{Z}^m \to \mathbb{Z}^n$ defined by $\mathbf{v} \mapsto \mathbf{v}M$. Then the kernel of φ_M is the lattice in $\mathbb{R}^m$, and we call it the kernel lattice of M.

2 Descriptions of Weighted Lattices and Weighted LLL Reduction

In this section, we introduce the notion of a weighted lattice, and give the weighted LLL algorithm. In Section 2.3, through an example we give a heuristic description of a situation in which the weighted LLL algorithm works well to find desired vectors.

2.1 Definition of Weighted Lattice

Basically, any lattice in $\mathbb{R}^m$ is endowed with the Euclidean norm. On the other hand, a weighted lattice is defined as a lattice endowed with a special norm which we call a weighted norm. The formal definitions of a weighted norm and a weighted lattice are as follows:

Definition 2.1.1 For a vector $\mathbf{w} = (w_1, \ldots, w_m) \in (\mathbb{R}_{>0})^m$, we define the map $\|\cdot\|_{\mathbf{w}} : \mathbb{R}^m \to \mathbb{R}$ as follows:

$$\|\mathbf{a}\|_{\mathbf{w}} := \sqrt{(a_1 w_1)^2 + \cdots + (a_m w_m)^2} \quad (\mathbf{a} = (a_1, \ldots, a_m) \in \mathbb{R}) \,. \tag{2.1.1}$$

Then $\|\cdot\|_{\mathbf{w}}$ is a norm on $\mathbb{R}^m$, and we call it the weighted norm for $\mathbf{w}$. We define a *weighted lattice for* $\mathbf{w}$ in $\mathbb{R}^m$ as a lattice endowed with the weighted norm for $\mathbf{w}$. For any lattice $\mathcal{L} \subset \mathbb{R}^m$ and a vector $\mathbf{w} \in (\mathbb{R}_{>0})^m$, we denote $\mathcal{L}$ by $\mathcal{L}^{\mathbf{w}}$ if we consider $\mathcal{L}$ as a lattice endowed with the weighted norm for $\mathbf{w}$.

The following lemma clearly holds.

Lemma 2.1.2 Let $\mathcal{L} \subset \mathbb{R}^m$ be a lattice and $\mathbf{w} = (w_1, \ldots, w_m) \in (\mathbb{R}_{>0})^m$ a vector. We set the following diagonal matrix:

$$W := \begin{pmatrix} w_1 & & & \\ & \ddots & & \\ & & \ddots & \\ & & & w_m \end{pmatrix} . \tag{2.1.2}$$

We define the isomorphism $f_W : \mathbb{R}^m \longrightarrow \mathbb{R}^m$ by $\mathbf{x} \mapsto \mathbf{x}W$. Then the following are equivalent for any $\mathbf{x} \in \mathcal{L}^{\mathbf{w}}$:

(1) The vector $\mathbf{x}$ is a shortest vector in $\mathcal{L}^{\mathbf{w}}$.

(2) The vector $\mathbf{x}W$ is a shortest vector in $f_W(\mathcal{L})$ with respect to the Euclidean norm.

Remark 2.1.3 Lemma 2.1.2 shows that we can find a shortest vector in $\mathcal{L}^{\mathbf{w}}$ if we find a shortest vector in $f_W(\mathcal{L})$ with respect to the Euclidean norm.

2.2 Weighted LLL Reduction

In this subsection, we define a weighted LLL reduced basis and give an algorithm to find such a basis.

Definition 2.2.1 (weighted LLL reduced bases) Let $\mathcal{L}$, W and f_W be as in Lemma 2.1.2. An ordered basis $\mathcal{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ of $\mathcal{L}$ is a *weighted LLL reduced basis* if $f_W(\mathcal{B}) = \{\mathbf{b}_1 W, \ldots, \mathbf{b}_n W\}$ is an LLL reduced basis of $f_W(\mathcal{L})$ with respect to the Euclidean norm.

We give an algorithm to find a weighted LLL reduced basis.

The weighted LLL algorithm
Input: a vector $\mathbf{w}$ and a basis $\mathcal{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ of the weighted lattice $\mathcal{L}$.
Output: a weighted LLL basis $\mathcal{B}' = \{\mathbf{b}_1', \ldots, \mathbf{b}_n'\}$ of the $\mathcal{L}$.

(1) Compute the basis $\{\mathbf{b}_1 W, \ldots, \mathbf{b}_n W\}$ of the lattice $f_W(\mathcal{L})$, where W and f_W are as in Lemma 2.1.2.

(2) Compute an LLL reduced basis $\mathcal{B}^{\mathbf{w}} = \{\mathbf{b}_1^{\mathbf{w}}, \ldots, \mathbf{b}_n^{\mathbf{w}}\}$ of $f_W(\mathcal{L})$ with respect to the Euclidean norm.

(3) Compute $\mathbf{b}_i' := \mathbf{b}_i^{\mathbf{w}} W^{-1}$ $(i = 1, \ldots, n)$, and return $\mathcal{B}' = \{\mathbf{b}_1', \ldots, \mathbf{b}_n'\}$.

It is easy to see that the above algorithm outputs a weighted LLL reduced basis for an input weight $\mathbf{w}$. Note that the weighted LLL algorithm is performed in polynomial time for any weight $\mathbf{w}$ since the LLL algorithm is performed in polynomial time (see Section 4.4 for details).

2.3 Example of A Weighted LLL Reduced Basis

In this subsection, we give an example to describe a situation in which the weighted LLL algorithm works well. We also describe how to choose a weight vector.

Let $\mathcal{L} \subseteq \mathbb{R}^6$ be the lattice spanned by the following 3 vectors $\mathbf{a}_1$, $\mathbf{a}_2$ and $\mathbf{a}_3$:

$$\begin{pmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \\ \mathbf{a}_3 \end{pmatrix} = \begin{pmatrix} 1 & 10765821 & -659296203686 & 786 & -24997119 & 697254279120 \\ 0 & 11058364 & -677197320133 & 0 & -25733721 & 730824644226 \\ 0 & 0 & 0 & 907 & 64235 & -16423065763 \end{pmatrix}.$$

Our goal is to find $\mathbf{a} = (a_1, \ldots, a_6) = (189, 1193, -2675592182, -194, 14633, 2707001228) \in \mathcal{L}$. Assume that we know the approximate bit length of the absolute values of the entries of $\mathbf{a}$ and the vector $\mathbf{a}$ is a relatively short vector in $\mathcal{L}$. First, we apply the usual LLL algorithm to $\mathcal{L}$ since the vector $\mathbf{a}$ is a short vector in $\mathcal{L}$. The result is as follows:

$$\begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{b}_3 \end{pmatrix} = \begin{pmatrix} 86063 & -19350109 & -118524414 & -92870 & 52635911 & -23059636 \\ 280433 & -63058787 & 54435622 & -302647 & 171526468 & 58631305 \\ -172353 & 38757295 & -132463843 & 185997 & -105423855 & 227177667 \end{pmatrix}.$$

The target vector $\mathbf{a}$ does not coincide with $\mathbf{b}_i$ for $i = 1, 2, 3$. We observe that $\|\mathbf{a}\|$ is larger than $\|\mathbf{b}_i\|$ for $i = 1, 2, 3$, and that $\mathbf{a}$ is a vector with entries of unbalanced sizes while $\mathbf{b}_i$ is a vector with entries of balanced sizes for $i = 1, 2, 3$. Thus the usual LLL algorithm does not seem to work well in this case.

However, the above observation and our assumptions on $\mathbf{a}$ seem to be useful to find the target vector $\mathbf{a}$. We know that the 1st, 2nd, 4th and 5th entries of $\mathbf{a}$ are much smaller than those of $\mathbf{b}_i$ for $i = 1, 2, 3$. Thus, for $i = 1, 2, 4, 5$ and sufficiently large $w_i \in \mathbb{Z}$, we have $\|\mathbf{a}\|_{\mathbf{w}} < \|\mathbf{b}_i\|_{\mathbf{w}}$, where $\mathbf{w} := (w_1, w_2, 1, w_4, w_5, 1)$. This means that we can expect $\mathbf{a}$ to become a shortest vector in $\mathcal{L}^{\mathbf{w}}$ for an appropriate weight $\mathbf{w}$. If so, we can expect the weighted LLL algorithm for an appropriate $\mathbf{w}$ to find $\mathbf{a}$ from Lemma 2.1.2 (note that the rank of $\mathcal{L}^{\mathbf{w}}$ is equal to 3).

Next, we describe how to choose such $\mathbf{w}$. Let W and f_W be defined as in Lemma 2.1.2. As we mentioned above, the LLL algorithm outputs vectors with entries of balanced sizes in

this case. We assume that it is also true for the lattice $f_W(\mathcal{L})$. Then we should choose $\mathbf{w}$ so that the vector $f_W(\mathbf{a})$ has entries of balanced sizes. Assume that we know $a_{\max} := \max\{|a_i| \mid i \in \{1,\ldots,6\}\}$ and $\lfloor \log_2(|a_{\max}/a_i|) \rfloor$ for $i = 1,\ldots,6$. Note that this assumption is reasonable because the value of $\lfloor \log_2 a_i \rfloor$ is known for $i = 1,\ldots,6$ from our assumption on $\mathbf{a}$. We take $\mathbf{w} := (2^{\lfloor \log_2(|a_{\max}/a_1|) \rfloor}, \ldots, 2^{\lfloor \log_2(|a_{\max}/a_6|) \rfloor})$. Then we have $f_W(\mathbf{a})$ with entries of balanced sizes. In this example, we have $\mathbf{w} = (2^{23}, 2^{21}, 1, 2^{23}, 2^{17}, 1)$.

Now, we apply the weighted LLL algorithm for the above $\mathbf{w}$ to $\mathcal{L}$. We apply the usual LLL algorithm to the lattice sppaned by $f_W(\mathbf{a}_i)$ for i=1,2,3. Then we have the following LLL reduced basis of $f_W(\mathcal{L})$:

$$\begin{pmatrix} 1585446912 & 2501902336 & -2675592182 & -1627389952 & 1917976576 & 2707001228 \\ 1585446912 & 2501902336 & -2675592182 & 5981077504 & 10337386496 & -13716064535 \\ 17758683136 & -94680121344 & -26386409149 & -10821304320 & 47526445056 & 10465888122 \end{pmatrix}.$$

Finally, we have the following weighted LLL reduced basis of $\mathcal{L}^{\mathbf{w}}$ for $\mathbf{w}$:

$$\begin{pmatrix} 189 & 1193 & -2675592182 & -194 & 14633 & 2707001228 \\ 189 & 1193 & -2675592182 & 713 & 78868 & -13716064535 \\ 2117 & -45147 & -26386409149 & -1290 & 362598 & 10465888122 \end{pmatrix}.$$

Thus, we have found the target vector $\mathbf{a} = (189, 1193, -2675592182, -194, 14633, 2707001228)$.

Remark 2.3.1 As we observed above, the target vector $\mathbf{a}$ is not a shortest vector in $\mathcal{L}$ and the LLL algorithm outputs the basis $\{\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3\}$ of $\mathcal{L}$ such that $\|\mathbf{a}\| > \|\mathbf{b}_i\|$ for $i = 1, 2, 3$. Thus, the weighted LLL algorithm may be useful to find a vector which is not shortest in a lattice of sufficiently low rank such as the lattice $\mathcal{L}$ in the above example.

2.4 Summary of Weighted LLL Algorithm

From the above example, we summarize a situation which the weighted LLL algorithm works well to find a target vector $\mathbf{v} = (v_1, \ldots, v_m)$ in a lattice $\mathcal{H}$.

(1) The rank of $\mathcal{H}$ is sufficiently low so that the LLL algorithm computes a sufficiently short vector.

(2) The vector $\mathbf{v}$ is not a shortest vector in $\mathcal{H}$ but some entries are relatively small. In other words, $\mathbf{v}$ has entries of unbalanced sizes.

(3) The bit length of the absolute values of the entries of $\mathbf{v}$ is known.

(4) The weight vector $\mathbf{w} := (2^{\lfloor \log_2(|v_{\max}/v_1|) \rfloor}, \ldots, 2^{\lfloor \log_2(|v_{\max}/v_m|) \rfloor})$ is computable, where $v_{\max} := \max\{|v_i| \mid i \in \{1,\ldots,m\}\}$.

(5) The vector $\mathbf{v}$ is sufficiently short in $\mathcal{H}^{\mathbf{w}}$ for the above $\mathbf{w}$.

The above situation occurs in our attack (cf. Section 4).

3 Overview of DEC

In this section, we give a brief review of DEC (see Section 3 in [Oku15] for details).

3.1 Polynomials of Degree Increasing Type

Definition 3.1.1 A polynomial $X(\underline{x}) \in \mathbb{Z}[\underline{x}] \setminus \{0\}$ is *of degree increasing type* if the map

$$\Lambda_X \longrightarrow \mathbb{Z}_{\geq 0} \ ; \ \underline{i} \mapsto \sum \underline{i}$$

is injective.

Remark 3.1.2 Let $X(\underline{x})$ be an element in $\mathbb{Z}[\underline{x}] \setminus \{0\}$.

(1) The polynomial $X(\underline{x})$ is of degree increasing type if and only if the total degrees of the monomials of $X(\underline{x})$ are different each other.

(2) If X is of degree increasing type, the support Λ_X is a totally ordered set by the following order $\succ$: for two elements $(i_1, \ldots, i_n)$ and $(j_1, \ldots, j_n)$ of Λ_X, we have $(i_1, \ldots, i_n) \succ (j_1, \ldots, j_n)$ if $i_1 + \cdots + i_n > j_1 + \cdots + j_n$.

Throughout this paper, for a polynomial X of degree increasing type, we endow Λ_X with the total order described in Remark 3.1.2 (2).

Example 3.1.3 The polynomial $X(x, y, z) := 3x^3y^2z - 4x^2y^2 - xyz + 5yz + y + 11 \in \mathbb{Z}[x, y, z]$ is of degree increasing type. In this case, we have $\mathbf{X} = (3, -4, -1, 5, 1, 11)$.

Now, we describe the DEC scheme according to [Oku15]. Note that although in [Oku15] the security parameter is not suggested, here we set the security parameter λ.

3.2 Key Generation

Secret Key :

- A vector $\underline{a} := (a_1, \ldots, a_n) \in \mathbb{Z}^n$.

Public Key :

(1) An integer d such that $\gcd(a_i, d) = 1$ for each $i \in \{1, \ldots, n\}$.

(2) An integer e such that $\gcd(e, \varphi(d)) = 1$, where φ is the Euler function.

(3) An irreducible polynomial $X(\underline{x}) \in \mathbb{Z}[\underline{x}]$ of degree increasing type such that $X(\underline{a}/d) = 0$ and $\sharp\Lambda_X \leq w_X$.

Construction of $X(\underline{x})$:

(1) Choose a finite subset $\Lambda \subset (\mathbb{Z}_{\geq 0})^n$ such that $\sharp\{\sum \underline{i} \ ; \ \underline{i} \in \Lambda\} = \sharp\Lambda \geq 3$ and $\underline{0} \in \Lambda$, where $\underline{0} := (0, \ldots, 0) \in (\mathbb{Z}_{\geq 0})^n$.

(2) Let $\underline{k}$ be the maximal element in Λ (note that Λ is a totally ordered set with respect to the order described in Remark 3.1.2 (2)). Choose a random non-zero integer $c_{\underline{i}}$ for each $\underline{i} \in \Lambda \setminus \{\underline{k}, \underline{0}\}$. For a choice of $c_{\underline{i}}$, see Remark 3.5.1.

(3) Choose random integers $c_{\underline{k}}$ and $c_{\underline{0}}$ such that

$$c_{\underline{k}}\underline{a}^{\underline{k}} + c_{\underline{0}}d^w \quad = \quad - \sum_{\underline{i} \in \Lambda \setminus \{\underline{k}, \underline{0}\}} c_{\underline{i}}\underline{a}^{\underline{i}}d^{w - \sum \underline{i}}, \tag{3.2.1}$$

where $w := \max\{\sum \underline{i} \ ; \ \underline{i} \in \Lambda\}$.

(4) Set $\Lambda_X := \Lambda$ and $X(\underline{x}) := \sum_{\underline{i} \in \Lambda_X} c_{\underline{i}} \underline{x}^{\underline{i}}$.

For a choice of X and sizes of e, d and a_i $(i = 1, \ldots, n)$, see Section 3.5 below.

Remark 3.2.1 There exist integers $c_{\underline{k}}$ and $c_{\underline{0}}$ such that the equality (3.2.1) is satisfied because a_i and d are mutually prime for each $i \in \{1, \ldots, n\}$ from the assumption.

3.3 Encryption

Plaintext : A polynomial $m \in \mathbb{Z}[x_1, \ldots, x_n]$ such that

(a) $\Lambda_m = \Lambda_X$,

(b) $1 < c_{i_1, \ldots, i_n}(m) < d$ for all $(i_1, \ldots, i_n) \in \Lambda_m$,

(c) $\gcd\left(c_{i_1, \ldots, i_n}(m), d\right) = 1$ for all $(i_1, \ldots, i_n) \in \Lambda_m$.

Encryption Process :

(1) Choose an integer $N \in \mathbb{Z}_{>0}$ uniformly such that $Nd > 2^{\lambda} H(X)$. For an upper bound of N, see Section 3.5 below.

(2) Construct the twisted plaintext $\widetilde{m}(\underline{x}) \in \mathbb{Z}[\underline{x}]$ by putting $\Lambda_{\widetilde{m}} := \Lambda_m$ and $c_{\underline{i}}(\widetilde{m}) := c_{\underline{i}}(m)^e \pmod{Nd}$ $(0 < c_{\underline{i}}(\widetilde{m}) < Nd,\ \underline{i} \in \Lambda_{\widetilde{m}})$.

(3) Choose a random $f(\underline{x}) \in \mathbb{Z}[\underline{x}]$ uniformly such that

(a) $\Lambda_f = \Lambda_X$,

(b) $H(\widetilde{m}) < c_{\underline{k}}(f) < Nd$ and $\gcd\left(c_{\underline{k}}(f), d\right) = 1$, where $\underline{k}$ is the maximal element in Λ_f.

(4) Choose random polynomials $s_j(\underline{x}), r_j(\underline{x}) \in \mathbb{Z}[\underline{x}]$ uniformly with $\Gamma_{s_j} = \Gamma_X$ and $\Gamma_{r_j} = \Gamma_f$ $(j \in \{1, 2, 3\})$.

(5) Put cipher polynomials $F_1(\underline{x})$, $F_2(\underline{x})$ and $F_3(\underline{x})$ as follows:

$$F_j(\underline{x}) := \widetilde{m}(\underline{x}) + s_j(\underline{x}) f(\underline{x}) + r_j(\underline{x}) X(\underline{x}) \quad (1 \leq j \leq 3).$$

Finally, send $(F_1(\underline{x}), F_2(\underline{x}), F_3(\underline{x}), N)$.

3.4 Decryption

Decryption Process :

(1) By substituting $\underline{a}/d$, a zero of $X(\underline{x})$, into $F_j(\underline{x})$ $(j \in \{1, 2, 3\})$, we obtain

$$h_j := F_j(\underline{a}/d) = \widetilde{m}(\underline{a}/d) + s_j(\underline{a}/d) f(\underline{a}/d) \quad (1 \leq j \leq 3).$$

Compute

$$\begin{aligned} H_1 := (h_1 - h_2) d^{2w_X} &= (s_1(\underline{a}/d) - s_2(\underline{a}/d)) f(\underline{a}/d) d^{2w_X}, \\ H_2 := (h_1 - h_3) d^{2w_X} &= (s_1(\underline{a}/d) - s_3(\underline{a}/d)) f(\underline{a}/d) d^{2w_X}. \end{aligned}$$

(2) Compute $g := \gcd(H_1, H_2)$. If $\gcd(g, d) > 1$, then let d' be the smallest factor of g satisfying $\gcd(d, g/d') = 1$ and replace g by g/d'.

(3) Compute $H := h_1 d^{2w_X} \pmod{g}$ and $\mu := H d^{-w_X} \pmod{g}$.

(4) Obtain the plaintext polynomial $m(\underline{x})$ from μ or $\mu - g$ by using an algorithm described in Sections 3.4 and 3.5 of [Oku15].

Remark 3.4.1 In the algorithm in Sections 3.4 and 3.5 of [Oku15], we need to compute $\varphi(d)$ efficiently. From this, we should choose a prime number as d.

3.5 Parameter Size

In [Oku15], the sizes of public/secret keys and the ciphertext are estimated so that DEC can be expected to have 128 bit security under some assumptions. In the following, we give their sizes under the same assumptions as [Oku15] to analyze the complexity of our attack (see Section 5 in [Oku15] for details).

(1) The sizes of $\underline{a}$, d, e and N:

$$2^{\frac{\lambda}{2}} \leq d < 2^{\frac{\lambda}{2}+1}, \quad (\lambda+1) + (\frac{\lambda}{2}+1)w_X \leq e < 2\left((\lambda+1) + (\frac{\lambda}{2}+1)w_X\right),$$
$$\frac{2^{\lceil \frac{\lambda}{n-1} \rceil}}{\varphi(d)} d \leq |a_i| < \frac{2^{\lceil \frac{\lambda}{n-1} \rceil + 1}}{\varphi(d)} d \ (i \in \{1, \ldots, n\}), \quad 2^{\lambda + (\frac{\lambda}{2}+1)(w_X - 1)} \leq N < 2^{\lambda + 1 + (\frac{\lambda}{2}+1)(w_X - 1)}.$$

We assume that $|c_{\underline{i}}(X)| < 2^b$ for any $\underline{i} \in \Lambda_X \setminus \{\underline{k}, \underline{0}\}$, where $\underline{k}$ is the maximal element of Λ_X (cf. [Oku15], Section 5).

(2) The size of a secret key is at most

$$\left(\lceil \frac{\lambda}{n-1} \rceil + 1\right) n + \lceil \log_2 d - \log_2 \varphi(d) \rceil$$

bits.

(3) The size of a public key is at most

$$\left(\lceil \frac{\lambda}{n-1} \rceil + (\frac{\lambda}{2} + 2 + b) + \lceil \log_2 d - \log_2 \varphi(d) \rceil\right) w_X + (\lambda + 1) + \lceil \log_2 e \rceil$$

bits.

(4) The size of a ciphertext is at most

$$\frac{3}{2}\left({w_X}^2 + w_X\right)(\lambda + 1 + (\lambda+2)w_X + \lceil \log_2 w_X \rceil) + \lambda + 1 + (\frac{\lambda}{2}+1)(w_X - 1) \qquad (3.5.1)$$

bits. Note that the size of each coefficient of F_i is at most $\lambda + 1 + (\lambda+2)w_X + \lceil \log_2 w_X \rceil$ bits for $i = 1, 2, 3$.

Remark 3.5.1 (1) In Section 4.5 of [Oku15], it is pointed out that we should use a polynomial X satisfying $w_X \geq 5$, $n \geq 3$ and some conditions as a public key in order to avoid finding rational solutions to $X = 0$. However, polynomials of degree increasing type are in a special class of polynomials, and finding rational zeros of such polynomials may be easier than finding those of general polynomials. Moreover, although finding rational zeros of polynomials of higher degree seems to be difficult in general, we should consider sizes of public keys and ciphertexts. Thus we recommend to use X of degree 10 as a public key.

(2) In Section 5 of [Oku15], it is pointed out that for a public key X, we may choose $c_{\underline{i}} \leq 2^{10}$ ($\underline{i} \in \Lambda \setminus \{\underline{k}, \underline{0}\}$), where $\underline{k}$ is the maximal element of Λ_X. However, since solving Diophantine equations of degree increasing type may be easier than solving more general Diophantine equations as we mentioned above, we should also consider using larger b to deal with a wide class of polynomials of degree increasing type. In our experiments of Section 6, we choose $b = 10, 50, 100$.

(3) The above choice of X is independent of the security level suggested in [Oku15] of the DEC. (The above sizes of a_i for $i = 1, \ldots, n$ are determined so that the number of possible choices of secret keys are at most 2^{128}.)

3.6 Toy Example of DEC

In the following, we give a toy example of DEC in the case of $n = 2$.

Secret Key :

- $\underline{a} = (a, b) = (47, 49) \in \mathbb{Z}^2$.

Public Key :
$(d, e, X) = (5, 17, 125x^3 + 675y - 110438)$
$(\Lambda_X = \{(3, 0), (0, 1), (0, 0)\}$, $\underline{k} = (3, 0)$, $H(X) = 110438$.)

Plaintext : $m(\underline{x}) = m(x, y) = 3x^3 + 3y + 2$.

Objects for Encryption :

(1) $N = 353408$ ($Nd = 1767040$).

(2) $\widetilde{m}(\underline{x}) = \widetilde{m}(x, y) = 146243x^3 + 146243y + 131072$ ($H(\widetilde{m}) = 146243$).

(3) $f(\underline{x}) = f(x, y) = 949843x^3 + 1324952y + 1109775$.
($c_{\underline{k}}(f) = 949843$, $H(\widetilde{m}) = 146243 < c_{\underline{k}}(f) = 949843 < 1767040 = Nd$.)

(4) s_j and r_j ($j = 1, 2, 3$) :

$$\begin{aligned} s_1 &= 115x^3 + 924y + 126337, \\ s_2 &= 82x^3 + 962y + 89939, \\ s_3 &= 67x^3 + 977y + 121816, \\ r_1 &= 691019x^3 + 1363650y + 1329029, \\ r_2 &= 852655x^3 + 1584164y + 2007688, \\ r_3 &= 940020x^3 + 2016302y + 1144882. \end{aligned}$$

(5) *Cipher Polynomials* : $F_j(\underline{x}) := \widetilde{m}(\underline{x}) + s_j(\underline{x}) f(\underline{x}) + r_j(\underline{x}) X(\underline{x}) \quad (1 \leq j \leq 3)$.

$$\begin{aligned}
F_1(\underline{x}) &= F_1(x,y) \\
&= 195609320x^6 + 1666918487x^3y + 43979457762x^3 + 2144719398y^2 + 18714355042y - 6569529455, \\
F_2(\underline{x}) &= F_2(x,y) \\
&= 184469001x^6 + 1795957655x^3y - 8395474520x^3 + 2343914524y^2 - 53364106711y - 121912862547, \\
F_3(\underline{x}) &= F_3(x,y) \\
&= 181141981x^6 + 1903319645x^3y + 12109757546x^3 + 2655481954y^2 - 59418815676y + 8750004156.
\end{aligned}$$

4 Attack against DEC in Polynomial Time via Weighted LLL Reduction

In this section, we present an attack against DEC via the weighted LLL algorithm. Under the conditions of Section 3, it is sufficient for breaking the one-wayness of DEC to recover $\widetilde{m}(\underline{x})$. Recall that $\Lambda_X = \Lambda_m = \Lambda_{\widetilde{m}} = \Lambda_f = \Lambda_{s_j} = \Lambda_{r_j}$ $(j = 1, 2, 3)$. Suppose that $\Lambda_{F_1} = \Lambda_{F_2} = \Lambda_{F_3}$. Throughout this section, put $q := \sharp\Lambda_X$ and $\Lambda_X = \{\underline{i}_1, \ldots, \underline{i}_q\}$.

4.1 Idea of Our Attack

Before we give an algorithm of our attack, we describe the idea of our attack. Recall from Section 3 that in DEC, we use the cipher polynomials of the forms

$$F_j(\underline{x}) := \widetilde{m}(\underline{x}) + s_j(\underline{x}) f(\underline{x}) + r_j(\underline{x}) X(\underline{x}) \quad (j = 1, 2, 3).$$

We reduce recovering $\widetilde{m}$ to finding special solutions of linear systems obtained from the public key X and the ciphertext (F_1, F_2, F_3, N) by linearization techniques described below.

We have the following equality from the way to construct the cipher polynomials for $j = 1, 2$:

$$F_j(\underline{x}) - F_{j+1}(\underline{x}) = (s_j(\underline{x}) - s_{j+1}(\underline{x})) f(\underline{x}) + (r_j(\underline{x}) - r_{j+1}(\underline{x})) X(\underline{x}). \tag{4.1.1}$$

Since the cipher polynomials $F_1(\underline{x})$, $F_2(\underline{x})$, $F_3(\underline{x})$ and the public key $X(\underline{x})$ are known, we may obtain $f(\underline{x})$ if we can determine $s_1(\underline{x}) - s_2(\underline{x})$ and $s_2(\underline{x}) - s_3(\underline{x})$. We set

$$\begin{aligned}
{s_j}'(\underline{x}) &:= s_j(\underline{x}) - s_{j+1}(\underline{x}), \\
{r_j}'(\underline{x}) &:= r_j(\underline{x}) - r_{j+1}(\underline{x}), \\
{F_j}'(\underline{x}) &:= F_j(\underline{x}) - F_{j+1}(\underline{x}) \\
&= {s_j}'(\underline{x}) f(\underline{x}) + {r_j}'(\underline{x}) X(\underline{x}) \quad (j = 1, 2), \\
g(\underline{x}) &:= {s_2}'(\underline{x}) {r_1}'(\underline{x}) - {s_1}'(\underline{x}) {r_2}'(\underline{x}).
\end{aligned}$$

We then have the following equalities:

$$\begin{aligned}
{F_1}'(\underline{x}) &= {s_1}'(\underline{x}) f(\underline{x}) + {r_1}'(\underline{x}) X(\underline{x}), &(4.1.2)\\
{F_2}'(\underline{x}) &= {s_2}'(\underline{x}) f(\underline{x}) + {r_2}'(\underline{x}) X(\underline{x}), &(4.1.3)\\
g(\underline{x}) X(\underline{x}) &= {s_2}'(\underline{x}) {F_1}'(\underline{x}) - {s_1}'(\underline{x}) {F_2}'(\underline{x}). &(4.1.4)
\end{aligned}$$

4.1.1 Step 1: Determination of $\mathbf{s}_j{}'$ $(j = 1, 2)$

Here, we describe how to determine $\mathbf{s}_j{}'$ for $i = 1, 2$. In the equality (4.1.4), we regard the coefficients of $s_j{}'(\underline{x})$ and $g(\underline{x})$ as variables. We then obtain the linear system $\mathbf{u}A = \mathbf{0}$, where A is a $((2q + \sharp\Lambda_{X^2}) \times \sharp\Lambda_{X^3})$ matrix. We denote the kernel lattice of A by $\mathcal{L}_1{}'$. Let $\mathcal{L}_1$ be the lattice spanned by the vectors consisting of the 1-$(2q)$th entries of the elements of $\mathcal{L}_1{}'$. Experimentally, the rank of $\mathcal{L}_1$ is equal to 3 in many cases (see Remark 6.0.2 in Section 6). Thus, we assume the following condition:

Assumption 4.1.1 The rank of $\mathcal{L}_1$ is equal to 3.

Moreover, as we will see in Section 4.3, the correct $(\mathbf{s}'_1, \mathbf{s}'_2)$ and $\mathcal{L}_1$ can be expected to satisfy the situation in Section 2.4. Note that this is true in many cases because of the construction of X (cf. Section 3.2). Thus, we use the weighted LLL algorithm for a weight $\mathbf{w}$ described below. Put $\mathbf{w}' = (w_1{}', \ldots, w_q{}')$ as follows:

$$w_i{}' := \begin{cases} 1 & (X_i = H(X)), \\ 2^{\left\lfloor \log_2\left(\frac{H(X)}{X_i}\right)\right\rfloor} & (otherwise), \end{cases}$$

where $\mathbf{X} := (X_1, \ldots, X_q)$ denotes the vector of the coefficients of $X(\underline{x})$. From the construction of $\mathbf{s}'_i$ for $i = 1, 2$ (cf. Section 3.3), We set $\mathbf{w} := \left(w'_1, \ldots, w'_q, w'_1, \ldots, w'_q\right)$. We assume the following condition.

Assumption 4.1.2 The $(\mathbf{s}'_1, \mathbf{s}'_2)$ is a shortest vector in $\mathcal{L}_1^{\mathbf{w}}$.

Let f_W be the isomorphism described in Section 2 from $\mathbb{R}^{2q}$ to $\mathbb{R}^{2q}$ as $\mathbb{R}$-vector spaces. From Assumption 4.1.1, the rank of $f_W(\mathcal{L}_1)$ is equal to 3. This means that we can expect the LLL algorithm to output a shortest vector in $\mathcal{L}_1^{\mathbf{w}}$ with high probability. Thus it is expected to find the correct $(\mathbf{s}'_1, \mathbf{s}'_2)$ via the weighted LLL algorithm for the weight $\mathbf{w}$ because of Lemma 2.1.2 and Assumption 4.1.2.

Remark 4.1.3 We may not determine $\mathbf{s}_j{}'$ $(j = 1, 2)$ if we apply the LLL algorithm to the lattice $\mathcal{L}_1$ as we will see in Section 4.3. Thus, the above assumptions and applying the weighted LLL algorithm to $\mathcal{L}_1$ are crucial for our attack.

4.1.2 Step 2: Fixing of A Candidate of f

Here, we describe how to determine a candidate of f. We substitute $s_1{}'(\underline{x})$ and $s_2{}'(\underline{x})$ obtained in Step 1 into (4.1.2) and (4.1.3). In the same way as Step 1, by regarding the coefficients of $f(\underline{x})$ and $r_j{}'(\underline{x})$ $(j = 1, 2)$ as variables, we have the linear system. We then fix $f'(\underline{x})$ such that (4.1.2) and (4.1.3) hold and that $f'(\underline{x})$ is close to the correct $f(\underline{x})$, i.e., the absolute values of all coefficients of the polynomial $f'(\underline{x}) - f(\underline{x})$ are small. Note that $f'(\underline{x})$ does not necessarily coincide with the correct $f(\underline{x})$ to recover $\widetilde{m}$ (cf. Remark 4.1.9 and Steps 3-3 and 3-4 in Section 4.2).

4.1.3 Step 3: Recovery of $\widetilde{m}$

Here, we describe how to recover $\widetilde{m}$. It is sufficient for recovering $\widetilde{m}(\underline{x})$ to find $\mathbf{s}_1$ (cf. Remark 4.1.9 and Steps 3-3 and 3-4 in Section 4.2). From the form of the ciphertext (see Section 3.3), we consider the following equality:

$$F_1(\underline{x}) = \widetilde{m}(\underline{x}) + s_1(\underline{x}) f'(\underline{x}) + r_1(\underline{x}) X(\underline{x}), \tag{4.1.5}$$

where $f'(\underline{x})$ is the polynomial obtained in Step 2 and other polynomials $\widetilde{m}(\underline{x})$, $s_1(\underline{x})$ and $r_1(\underline{x})$ are unknown. Note that if we have the correct solution in Step 1 and $\gcd(X, {s_1}') = 1$, then there exists a unique polynomial $r(\underline{x})$ such that the correct $\widetilde{m}(\underline{x})$, $s_1(\underline{x})$ and $f'(\underline{x})$ (not necessarily $f(\underline{x})$) satisfy the equality $F_1 = \widetilde{m} + s_1 f' + rX$ (cf. Remark 4.1.9). In the same way as Steps 1 and 2, by regarding the coefficients of $\widetilde{m}(\underline{x})$, $s_1(\underline{x})$ and $r_1(\underline{x})$ as variables, we have the linear system $\mathbf{w}C = \mathbf{c}$, where C is a $(3q \times \sharp\Lambda_{X^2})$ matrix and $\mathbf{c} \in \mathbb{Z}^{\sharp\Lambda_{X^2}}$. We denote the kernel lattice of C by $\mathcal{L}_3$. The rank of $\mathcal{L}_3$ is low and equal to 3 with high probability (see Remark 6.0.2). From this, we assume the rank of $\mathcal{L}_3$ as follows:

Assumption 4.1.4 The rank of $\mathcal{L}_3$ is equal to 3.

Let $\mathbf{w}_0$ be one solution of $\mathbf{w}C = \mathbf{c}$ and $\{\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3\}$ a basis of $\mathcal{L}_3$. Note that every integral solution of the system is represented as $\mathbf{w}_0 + a_1\mathbf{w}_1 + a_2\mathbf{w}_2 + a_3\mathbf{w}_3$ ($a_i \in \mathbb{Z}$, $i = 1, 2, 3$). The 1-$\sharp\Lambda_X$-th entries of $\mathbf{w}_0$, $\mathbf{w}_1$, $\mathbf{w}_2$ and $\mathbf{w}_3$ correspond to the coefficients of $\widetilde{m}$. We also note that $\mathbf{w}C = \mathbf{0}$ has a solution $\mathbf{w}'$ such that its 1-$\sharp\Lambda_X$-th entries are equal to 0 (see Remark 4.1.8). Thus it is possible to choose $\mathbf{w}_3$ so that its 1-$\sharp\Lambda_X$-th entries are equal to 0.

Assumption 4.1.5 The vector $\mathbf{s}_1$ coincides with the vector consisting of the $(\sharp\Lambda_X + 1)$-$2\sharp\Lambda_X$-th entries of $\mathbf{w}_0 + \mathbf{w}_3 - \mathbf{z}$, where $\mathbf{z}$ is a closest vector in $\langle \mathbf{w}_1, \mathbf{w}_2 \rangle_{\mathbb{Z}}$ to $\mathbf{w}_0 + \mathbf{w}_3$.

The ${\mathcal{L}_3}'$ is a lattice of 2-rank, and so we can expect that we will find $\mathbf{s}_1$ in polynomial time by Babai's nearest plane algorithm for solving CVP ([Bab86]) with sufficiently high probability under Assumption 4.1.5.

Remark 4.1.6 The reason why we assume Assumption 4.1.5 is the following: From the choice of $\mathbf{s}_1$, the absolute values of the entries of $\mathbf{s}_1$ are sufficiently smaller than those of $\widetilde{\mathbf{m}}$ and $\mathbf{r}_1$. Thus we can expect that the value of $\|\mathbf{w}_0 + \mathbf{w}_3 - (a_1\mathbf{w}_1 + a_2\mathbf{w}_2)\|$ is sufficiently small if certain entries of $\mathbf{w}_0 + \mathbf{w}_3 - (a_1\mathbf{w}_1 + a_2\mathbf{w}_2)$ coincide with those of $\mathbf{s}_1$.

Remark 4.1.7 In Step 2, any solution $(f'(\underline{x}), r_1''(\underline{x}))$ of the linear system can be written as $f'(\underline{x}) = f(\underline{x}) + aX(\underline{x})$ and $r_1''(\underline{x}) = r_1'(\underline{x}) - as_1'(\underline{x})$, respectively ($a \in \mathbb{Z}$) if $\gcd(X, s_1') = 1$ and if the solution in Step 1 is the correct $(s_1'(\underline{x}), s_2'(\underline{x}))$. In fact, by putting $p(\underline{x}) := f'(\underline{x}) - f(\underline{x})$ and $q(\underline{x}) := r_1''(\underline{x}) - r_1'(\underline{x})$, we have

$$\begin{aligned} F_1'(\underline{x}) &= s_1'(\underline{x}) f'(\underline{x}) + r_1''(\underline{x}) X(\underline{x}) \\ &= s_1'(\underline{x}) (f(\underline{x}) + p(\underline{x})) + \left(r_1'(\underline{x}) + q(\underline{x})\right) X(\underline{x}) \\ &= \left(s_1'(\underline{x}) f(\underline{x}) + r_1'(\underline{x}) X(\underline{x})\right) + \left(s_1'(\underline{x}) p(\underline{x}) + q(\underline{x}) X(\underline{x})\right) \\ &= F_1'(\underline{x}) + \left(s_1'(\underline{x}) p(\underline{x}) + q(\underline{x}) X(\underline{x})\right). \end{aligned}$$

It implies $s_1'(\underline{x}) p(\underline{x}) = -q(\underline{x}) X(\underline{x})$. Thus if $\gcd(X, s_1') = 1$, there exists an integer $a \in \mathbb{Z}$ such that $p(\underline{x}) = aX(\underline{x})$ and $q(\underline{x}) = -as_1'(\underline{x})$ because $\deg p \leq \deg X$ and $\deg q \leq \deg s_1'$. This fact implies

that the rank of the kernel lattice in Step 2 is equal to 1 with high probability. If the solution obtained in Step 1 is $(-s_1'(\underline{x}), -s_2'(\underline{x}))$, then $(f'(\underline{x}), r_1''(\underline{x}))$ can be written as $f'(\underline{x}) = -f(\underline{x}) + aX(\underline{x})$ and $r_1''(\underline{x}) = r_1'(\underline{x}) - as_1'(\underline{x})$, respectively ($a \in \mathbb{Z}$) by the same argument. Note that since X is irreducible from the construction of X in Section 3, we have $\gcd(X, s_1'(\underline{x})) = 1$ with high probability.

Remark 4.1.8 In Step 3, the linear system $\mathbf{w}C = \mathbf{0}$ has a solution $\mathbf{w}'$ such that its 1-$\sharp\Lambda_X$-th entries are equal to 0. Let $(\mathbf{m}', \mathbf{s}', \mathbf{r}')$ be one solution of $\mathbf{w}C = \mathbf{c}$, i.e., $F_1 = m' + s'f' + r'X$. The vector $(\mathbf{m}', \mathbf{s}', \mathbf{r}') + (\mathbf{0}, \mathbf{X}, -\mathbf{f}')$ is also a solution of $\mathbf{w}C = \mathbf{c}$. Indeed,

$$\left(m' + 0\right) + \left(s' + X\right) f' + \left(r' - f'\right) X = \left(m' + s'f' + r'X\right) + Xf' - f'X = F_1.$$

Thus $(\mathbf{0}, \mathbf{X}, -\mathbf{f}')$ is an element of the kernel lattice of C.

Remark 4.1.9 If we succeed in finding the correct $s_1(\underline{x})$ in Step 3 and $\gcd(X, {s_1}') = 1$, there exists $r(\underline{x})$ satisfying the equality $F_1(\underline{x}) - s_1(\underline{x}) f'(\underline{x}) = \widetilde{m}(\underline{x}) + r(\underline{x}) X(\underline{x})$. In fact, $f'(\underline{x})$ obtained in Step 2 can be written as $f'(\underline{x}) = f(\underline{x}) + aX(\underline{x})$ or $f'(\underline{x}) = -f(\underline{x}) + aX(\underline{x})$ ($a \in \mathbb{Z}$) from Remark 4.1.7. We may assume that $f'(\underline{x}) = f(\underline{x}) + aX(\underline{x})$. Then we have

$$\begin{aligned} F_1(\underline{x}) - \widetilde{m}(\underline{x}) - s_1(\underline{x}) f'(\underline{x}) &= s_1(\underline{x}) f(\underline{x}) + r_1(\underline{x}) X(\underline{x}) - s_1(\underline{x}) f'(\underline{x}) \\ &= s_1(\underline{x}) \left(f'(\underline{x}) - aX(\underline{x})\right) + r_1(\underline{x}) X(\underline{x}) - s_1(\underline{x}) f'(\underline{x}) \\ &= \left(r_1(\underline{x}) - as_1(\underline{x})\right) X(\underline{x}). \end{aligned}$$

Thus we have $F_1(\underline{x}) - s_1(\underline{x}) f'(\underline{x}) = \widetilde{m}(\underline{x}) + r(\underline{x}) X(\underline{x})$ by putting $r(\underline{x}) := r_1(\underline{x}) - as_1(\underline{x})$.

4.2 Algorithm of Our Attack

We write down an algorithm of the attack proposed in Section 4.1. Recall from Section 3 that a public key, a secret key and a ciphertext are $(d, e, X) \in \mathbb{Z}^2 \times (\mathbb{Z}[\underline{x}])$, $\underline{a} = (a_1, \ldots, a_n) \in \mathbb{Z}^n$ and $(F_1, F_2, F_3, N) \in (\mathbb{Z}[\underline{x}])^3 \times \mathbb{Z}$, respectively, such that $X(a_1/d, \ldots, a_n/d) = 0$. Let $m \in \mathbb{Z}[\underline{x}]$ be a plaintext. Note that $F_j = \widetilde{m} + s_j f + r_j X$, where $\widetilde{m}$, f, s_j and r_j are the twisted plaintext and random polynomials chosen uniformly according to Section 3.3, respectively. We also recall that Λ_X and w_X denote the support of X and the total degree of X, respectively (see Notation in Section 1). We fix $\sharp\Lambda_X$ and w_X. Let $\underline{k}$ be the maximal element of Λ_X with respect to the order in Remark 3.1.2 (2).

Algorithm of Proposed Attack
Input: a public key (d, e, X) and a ciphertext (F_1, F_2, F_3, N).
Output: a twisted plaintext $\widetilde{m}(\underline{x})$.

Step 1. Determination of $s_j' := s_j - s_{j+1}$ $(j = 1, 2)$
Step 1-1. Put $F_i' := F_i - F_{i+1}$ $(i = 1, 2)$ and $g := s_2'r_1' - s_1'r_2'$. Solve the linear system $\mathbf{u}A = \mathbf{0}$ obtained by comparing the coefficients of the equality

$$s_2'(\underline{x}) F_1'(\underline{x}) - s_1'(\underline{x}) F_2'(\underline{x}) = g(\underline{x}) X(\underline{x}), \tag{4.2.1}$$

where A is a $(2\sharp\Lambda_X + \sharp\Lambda_{X^2}) \times \sharp\Lambda_{X^3}$ matrix. Let $\{\mathbf{u}_1', \mathbf{u}_2', \mathbf{u}_3'\}$ be the basis of the kernel lattice of A.

Step 1-2. Let $\mathbf{u}_i$ be the vector consisting of the 1-$(2\sharp\Lambda_X)$-th entries of $\mathbf{u}'_i$ for $i = 1, 2, 3$. Execute the weighted LLL algorithm for the weight described in Section 4.1 to the lattice $\mathcal{L}_1 := \langle \mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3 \rangle$, and then obtain $\mathbf{s}_1{}'$ and $\mathbf{s}_2{}'$.

Step 2. Fixing of a candidate of $f(\underline{x})$
Step 2-1. Put $r'_i := r_i - r_{i+1}$ $(i = 1, 2)$. Solve the linear system $\mathbf{v}B = \mathbf{b}$ obtained by comparing the coefficients of the equalities

$$F'_1(\underline{x}) = s'_1(\underline{x}) f(\underline{x}) + r'_1(\underline{x}) X(\underline{x}), \tag{4.2.2}$$
$$F'_2(\underline{x}) = s'_2(\underline{x}) f(\underline{x}) + r'_2(\underline{x}) X(\underline{x}), \tag{4.2.3}$$

where B is a $(3\sharp\Lambda_X \times \sharp\Lambda_{X^2})$ matrix. Let $\mathbf{v}_0$ be a solution of $\mathbf{v}B = \mathbf{b}$ and $\{\mathbf{v}_1\}$ a basis of the kernel lattice $\mathcal{L}_2$ of B. Note that if $\gcd(X, s'_1) = 1$, then $\mathcal{L}_2$ is always a lattice of 1-rank (cf. Remark 4.1.7).

Step 2-2. Let $\mathbf{v}'_0 := \mathbf{v}_0 - \lfloor \langle \mathbf{v}_0, \mathbf{v}_1 \rangle / \langle \mathbf{v}_1, \mathbf{v}_1 \rangle \rceil \mathbf{v}_1$ be the other solution of $\mathbf{v}B = \mathbf{b}$. Let $\mathbf{v}''_0$ be the vector consisting of the 1-$(\sharp\Lambda_X)$-th entries of $\mathbf{v}'_0$. Construct $f'(\underline{x}) \in \mathbb{Z}[\underline{x}]$ so that $\mathbf{f}' = \mathbf{v}''_0$. Note that $\mathbf{v}'_0$ provides the polynomial closer to the correct f than $\mathbf{v}_0$ in many cases (cf. Step 2 in Section 4.3).

Step 3. Recovery of $\widetilde{m}(\underline{x})$
Step 3-1. Solve the linear system $\mathbf{w}C = \mathbf{c}$ obtained by comparing the coefficients of the equality

$$F_1(\underline{x}) = \widetilde{m}(\underline{x}) + s_1(\underline{x}) f'(\underline{x}) + r_1(\underline{x}) X(\underline{x}), \tag{4.2.4}$$

where C is a $3\sharp\Lambda_X \times \sharp\Lambda_{X^2}$ matrix and f' is the polynomial obtained in Step 2-2. Let $\mathbf{w}_0$ be a solution of $\mathbf{w}C = \mathbf{c}$ and $\{\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3\}$ a basis of the kernel lattice $\mathcal{L}_3$ of C.

Step 3-2. Execute Babai's nearest plane algorithm to find a closest vector $\mathbf{z}$ in the lattice $\mathcal{L}'_3 := \langle \mathbf{w}_1, \mathbf{w}_2 \rangle_{\mathbb{Z}}$ to $\mathbf{w}_0 + \mathbf{w}_3$. Let $\mathbf{s}_1$ be the vector consisting of the $(\sharp\Lambda_X + 1)$-$2\sharp\Lambda_X$-th entries of $\mathbf{w}_0 + \mathbf{w}_3 - \mathbf{z}$.

Step 3-3. Solve the linear system $\mathbf{x}H = \mathbf{h}$ obtained by comparing the coefficients of the equality

$$F_1(\underline{x}) - \widetilde{m}(\underline{x}) - s_1(\underline{x}) f'(\underline{x}) = r(\underline{x}) X(\underline{x}), \tag{4.2.5}$$

where the coefficients of $\widetilde{m}$ and r are variables and H is a $(2\sharp\Lambda_X \times \sharp\Lambda_{X^2})$ matrix. Let $\mathbf{x}$ be a solution of $\mathbf{x}H = \mathbf{h}$. Let $\mathbf{r}'$ be the vector consisting of the entries corresponding to r of $\mathbf{x}$. Then we obtain a polynomial r' whose coefficients coincide with those of r except the constant part, i.e., $r = r' + t$ for some $t \in \mathbb{Z}$.
Step 3-4. Compute

$$\begin{aligned}
e' &:= e^{-1} \bmod \varphi(d), \\
H_1(\underline{x}) &:= F_1(\underline{x}) - s_1(\underline{x}) f'(\underline{x}) - r'(\underline{x}) X(\underline{x}), \\
\mu &:= c_{\underline{k}}(H_1), (\underline{k} : \text{maximal element of } \Lambda_X) \\
c_{\underline{k}}(m') &:= \mu^{e'} \pmod d \quad (0 < c_{\underline{k}}(m') < d), \\
c_{\underline{k}}(\widetilde{m}) &:= \left(c_{\underline{k}}(m')\right)^e \pmod{Nd} \quad (0 < c_{\underline{k}}(\widetilde{m}) < Nd), \\
t &:= \left(\mu - c_{\underline{k}}(\widetilde{m})\right) / c_{\underline{k}}(X), \\
\widetilde{m}(\underline{x}) &:= F_1(\underline{x}) - s_1(\underline{x}) f'(\underline{x}) - \left(r'(\underline{x}) + t\right) X(\underline{x}).
\end{aligned}$$

Output $\widetilde{m}(\underline{x})$.

Remark 4.2.1 Let $\underline{k}$ be the maximal element of Λ_X. In Step 3-4 of the above algorithm, we use the fact that $c_{\underline{k}}(X)$ is divisible by d to compute an integer t (see (3.2.1) for the divisibility of $c_{\underline{k}}(X)$).

4.3 Cryptanalysis of Toy Example

We break the one-wayness of the instance in Section 3.6 of DEC. We use the notation in Section 3.6. In this case, $\Lambda_g = \Lambda_{X^2} = \{(6,0),(3,1),(3,0),(0,2),(0,1),(0,0)\}$. We recover $\widetilde{m}(x,y)$ only from known $F_j(x,y)$ $(j=1,2,3)$, $X(x,y)$ and Λ_X.

4.3.1 Step 1: Determination of $s'_j = s_j - s_{j+1}$ $(j=1,2)$

Here, we determine $s'_j = s_j - s_{j+1}$ for $j = 1, 2$. Compute

$$\begin{aligned} F'_1(x,y) &:= F_1(x,y) - F_2(x,y) \\ &= 11140319x^6 - 129039168x^3y + 52374932282x^3 - 199195126y^2 + 72078461753y + 115343333092, \\ F'_2(x,y) &:= F_2(x,y) - F_3(x,y) \\ &= 3327020x^6 - 107361990x^3y - 20505232066x^3 - 311567430y^2 + 6054708965y - 130662866703. \end{aligned}$$

We put

$$\begin{aligned} s'_j(x,y) &:= c_1^{(j)}x^3 + c_2^{(j)}y + c_3^{(j)} \quad (j=1,2), \\ g(x,y) &:= c_1^{(g)}x^6 + c_2^{(g)}x^3y + c_3^{(g)}x^3 + c_4^{(g)}y^2 + c_5^{(g)}y + c_6^{(g)}, \end{aligned}$$

where $c_i^{(j)}$'s and $c_i^{(g)}$'s are variables. By comparing the coefficient of $\underline{x}^{\underline{i}}$ for each $\underline{i} \in \Lambda_{X^3}$ in the equation (4.1.4), we have the linear system $\mathbf{u}A' = \mathbf{0}$, where A' is a (9×9) matrix.
The rank of the kernel lattice $\mathcal{L}'_1$ of A' is equal to 3. Compute a basis $\{\mathbf{u}'_1, \mathbf{u}'_2, \mathbf{u}'_3\}$ of $\mathcal{L}'_1$, and let $\mathbf{u}_j$ be the vector of the 1-6th entries of $\mathbf{u}'_j$ for $j = 1, 2, 3$. We then have

$$\begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \mathbf{u}_3 \end{pmatrix} = \begin{pmatrix} 1 & 11464 & -3475226 & 80 & 5520 & 916415 \\ 0 & 27025 & -8194204 & 0 & 12000 & 2328055 \\ 0 & 0 & 0 & 125 & 675 & -110438 \end{pmatrix}.$$

By applying the LLL algorithm to the lattice $\mathcal{L}_1$ spanned by $\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3$, we have

$$\begin{pmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \\ \mathbf{a}_3 \end{pmatrix} = \begin{pmatrix} 1568 & 3927 & -8708 & -435 & -4365 & -6789 \\ -1792 & -4488 & 9952 & 515 & 5085 & -8018 \\ 3841 & 9499 & 15250 & -1095 & -10905 & -1034 \end{pmatrix}.$$

However, actually, the target vector $(\mathbf{s}'_1, \mathbf{s}'_2)$ is

$$(\mathbf{s}'_1, \mathbf{s}'_2) = (33, -38, 36398, 15, -15, -31877).$$

Thus $\mathbf{a}_i$ does not coincide with both of $(\mathbf{s}'_1, \mathbf{s}'_2)$ and $-(\mathbf{s}'_1, \mathbf{s}'_2)$ for any $i = 1, 2, 3$. Note that 1-2nd and 4-5th entries of the correct $(\mathbf{s}'_1, \mathbf{s}'_2)$ are much smaller than its other entries. This is true in many cases from the constructions of X, s'_1 and s'_2 described in Sections 3 and 4. On the other hand, the absolute values of all entries of $\mathbf{a}_i$ almost have the same sizes for $i = 1, 2, 3$. Moreover, the

norm $\| (\mathbf{s}'_1, \mathbf{s}'_2) \|$ is larger than $\max\{\|\mathbf{a}_1\|, \|\mathbf{a}_2\|, \|\mathbf{a}_3\|\}$. Indeed, we have $\| (\mathbf{s}'_1, \mathbf{s}'_2) \| \approx 48383.47$ and $\max\{\|\mathbf{a}_1\|, \|\mathbf{a}_2\|, \|\mathbf{a}_3\|\} \approx 21418.08$. This means that our target vector $(\mathbf{s}'_1, \mathbf{s}'_2)$ is not shortest in $\mathcal{L}_1$ of 3-rank. Thus, the LLL algorithm does not seem to work for finding $(\mathbf{s}'_1, \mathbf{s}'_2)$.

To obtain $(\mathbf{s}'_1, \mathbf{s}'_2)$, we apply the weighted LLL algorithm for the weight $\mathbf{w}$ described below to $\mathcal{L}_1$ since the above situation is good for the weighted LLL algorithm (cf. Section 2). Recall that $\mathbf{X} = (125, 675, -110438)$. We have

$$\left(\frac{H(X)}{125}, \frac{H(X)}{675}, \frac{H(X)}{110438} \right) = \left(\frac{110438}{125}, \frac{110438}{675}, 1 \right).$$

Put

$$\mathbf{w} = \left(2^{\lfloor \log_2 (\frac{110438}{125}) \rfloor}, 2^{\lfloor \log_2 (\frac{110438}{675}) \rfloor}, 1, 2^{\lfloor \log_2 (\frac{110438}{125}) \rfloor}, 2^{\lfloor \log_2 (\frac{110438}{675}) \rfloor}, 1 \right) = \left(2^9, 2^7, 1, 2^9, 2^7, 1 \right).$$

We obtain the following weighted LLL reduced basis of $\mathcal{L}_1^{\mathbf{w}}$:

$$\begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{b}_3 \end{pmatrix} = \begin{pmatrix} 33 & -38 & 36398 & 15 & -15 & -31877 \\ -33 & 38 & -36398 & 110 & 690 & -78561 \\ -158 & -637 & 74040 & -15 & 15 & 31877 \end{pmatrix}.$$

Note that $\mathbf{b}_1$ coincides with the target vector $(\mathbf{s}'_1, \mathbf{s}'_2)$.

4.3.2 Step 2: Fixing of A Candidate of f

Here, we fix a candidate of f. We set

$$\begin{aligned} f(x, y) &:= c_1^{(f)} x^3 + c_2^{(f)} y + c_3^{(f)}, \\ r'_j(x, y) &:= c_1^{(j)} x^3 + c_2^{(j)} y + c_3^{(j)} \quad (j = 1, 2), \end{aligned}$$

where $c_i^{(f)}$'s and $c_i^{(j)}$'s are variables. By substituting s'_1 and s'_2 obtained in Step 1 into the equalities (4.1.2) and (4.1.3), and by comparing the coefficient of $\underline{x}^{\underline{i}}$ for each $\underline{i} \in \Lambda_{X^2}$, we have the linear system $\mathbf{v}B = \mathbf{b}$, where B is a (9×6) matrix. The rank of the kernel lattice $\mathcal{L}_2$ of B is equal to 1. We obtain a solution $\mathbf{v}_0$ of $\mathbf{v}B = \mathbf{b}$ and a basis $\{\mathbf{v}_1\}$ of $\mathcal{L}_2$ as follows:

$$\begin{pmatrix} \mathbf{v}_0 \\ \mathbf{v}_1 \end{pmatrix} = \begin{pmatrix} -32 & -3804373 & 840328137 & 89131 & -509276 & 275909743 & 26620 & -546123 & -241370517 \\ 125 & 675 & -110438 & -33 & 38 & -36398 & -15 & 15 & 31877 \end{pmatrix}.$$

Compute another solution $\mathbf{v}'_0 := \mathbf{v}_0 - \lfloor \langle \mathbf{v}_0, \mathbf{v}_1 \rangle / \langle \mathbf{v}_1, \mathbf{v}_1 \rangle \rceil \mathbf{v}_1$ of $\mathbf{v}B = \mathbf{b}$. Let $\mathbf{v}''_0$ be the vector consisting of the 1-3rd entries of $\mathbf{v}'_0$. We then have

$$\mathbf{v}''_0 = (950468, 1328327, 557585).$$

We set

$$f'(x, y) := 950468x^3 + 1328327y + 557585. \tag{4.3.1}$$

Note that the polynomial f' obtained from $\mathbf{v}''_0$ is closer to the correct f than the one obtained from $\mathbf{v}_0$. We also note that it is possible to proceed to the next step even if f' does not coincide with f (cf. Remark 4.1.9).

4.3.3 Step 3: Recovery of $\widetilde{m}$

Finally, we recover $\widetilde{m}(x,y)$. We find $s_1(x,y)$ before recovering $\widetilde{m}(x,y)$. Put

$$\begin{aligned} \widetilde{m}(x,y) &:= c_1x^3 + c_2y + c_3, && (4.3.2)\\ s_1(x,y) &:= c_4x^3 + c_5y + c_6, && (4.3.3)\\ r_1(x,y) &:= c_7x^3 + c_8y + c_9, && (4.3.4) \end{aligned}$$

where c_i's are variables. By substituting f' obtained in Step 2 into the equalities (4.1.5), and by comparing the coefficient of $\underline{x}^{\underline{i}}$ for each $\underline{i} \in \Lambda_{X^2}$, we have the linear system $\mathbf{w}C = \mathbf{c}$, where C is a (9×6) matrix. The rank of the kernel lattice $\mathcal{L}_3$ of C is equal to 3. We fix a solution $\mathbf{w}_0$ of the system and a basis $\{\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3\}$ of $\mathcal{L}_3$ as follows:

$$\begin{pmatrix} \mathbf{w}_0 \\ \mathbf{w}_1 \\ \mathbf{w}_2 \\ \mathbf{w}_3 \end{pmatrix} = \begin{pmatrix} 225204073068 & 315361848743 & -6569529455 & -10 & 249 & 0 & 1640912 & 2687357 & 0 \\ 1 & 163580614 & -36132895073 & 0 & 0 & 43 & 0 & 0 & -326961 \\ 0 & 475525025 & -105037483109 & 0 & 0 & 125 & 0 & 0 & -950468 \\ 0 & 0 & 0 & 125 & 675 & -110438 & -950468 & -1328327 & -557585 \end{pmatrix}.$$

We find a vector $\mathbf{z}$ of the lattice $\langle \mathbf{w}_1, \mathbf{w}_2 \rangle_{\mathbb{Z}}$ close to $\mathbf{w}_0 + \mathbf{w}_3$ by applying Babai's nearest plane algorithm. We then have the matrix

$$\begin{pmatrix} 225203926700 & 315361701825 & -6569550089 & 0 & 0 & -236775 & 0 & 0 & -1254928 \\ 146368 & 146918 & 20634 & 115 & 924 & 126337 & 690444 & 1359030 & 697343 \end{pmatrix},$$

where 1st and 2nd rows are the vectors $\mathbf{z}$ and $\mathbf{w}_0 + \mathbf{w}_3 - \mathbf{z}$, respectively. The vector consisting of the 4-6th entries of $\mathbf{w}_0 + \mathbf{w}_3 - \mathbf{z}$ is equal to the correct $\mathbf{s}_1$.
Next, we compute $r(\underline{x})$ satisfying $F_1(\underline{x}) - \widetilde{m}(\underline{x}) - s_1(\underline{x})'(\underline{x}) = r(\underline{x})X(\underline{x})$. Note that there exists a polynomial r satisfying the above equality, and that we can recover $\widetilde{m}$ if we obtain such an r (cf. Remark 4.1.9 and Step 3-4 in Section 4.2). We set

$$\begin{aligned} r(x,y) &:= c_1x^3 + c_2y + c_3, && (4.3.5)\\ \widetilde{m}(x,y) &:= c_4x^3 + c_5y + c_6, && (4.3.6) \end{aligned}$$

where c_i's are variables. In the equality $F_1(\underline{x}) - s_1(\underline{x})f'(\underline{x}) = \widetilde{m}(\underline{x}) + r(\underline{x})X(\underline{x})$, by comparing the coefficient of $\underline{x}^{\underline{i}}$ for each $\underline{i} \in \Lambda_{X^2}$, we have the linear system $\mathbf{x}H = \mathbf{h}$, where H is a (6×6) matrix. The rank of the kernel lattice $\mathcal{L}_4$ of H is equal to 1. We fix a solution $\mathbf{x}_0$ of $\mathbf{x}H = \mathbf{h}$ and a basis $\{\mathbf{x}_1\}$ of $\mathcal{L}_4$ as follows:

$$\begin{pmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \end{pmatrix} = \begin{pmatrix} 2591380 & 4015684 & 0 & 226710493 & 1223593193 & -200170290060 \\ 0 & 0 & 1 & -125 & -675 & 110438 \end{pmatrix}.$$

We set

$$r'(x,y) := 2591380x^3 + 4015684y + 1. \quad (4.3.7)$$

There exists a unique $t \in \mathbb{Z}$ such that $r(\underline{x}) = r'(\underline{x}) + t$. Our aim is to find such an integer t (cf.

Steps 3-3 and 3-4 in Section 4.2). Let $\underline{k}$ be the maximal element of Λ_X. Put

$$\begin{aligned}
e' &:= e^{-1} \bmod \varphi(d) \\
&= 1, \\
H_1(\underline{x}) &:= F_1(\underline{x}) - s_1(\underline{x}) f'(\underline{x}) - r'(\underline{x}) X(\underline{x}) \\
&= 226710368x^3 + 1223592518y - 200170179622, \\
\mu &:= c_{\underline{k}}(H_1) \\
&= -200170179622, \\
c_{\underline{k}}(m') &:= \mu^{e'} \pmod{d} \quad \left(0 < c_{\underline{k}}(m') < d\right) \\
&= 3, \\
c_{\underline{k}}(\widetilde{m}) &= \left(c_{\underline{k}}(m')\right)^e \pmod{Nd} \quad \left(0 < c_{\underline{k}}(\widetilde{m}) < Nd\right) \\
&= 146243, \\
t &= \left(\mu - c_{\underline{k}}(\widetilde{m})\right) / c_{\underline{k}}(X) \\
&= 1812513, \\
\widetilde{m}(\underline{x}) &= F_1(\underline{x}) - s_1(\underline{x}) f'(\underline{x}) - \left(r'(\underline{x}) + t\right) X(\underline{x}) \\
&= 146243x^3 + 146243y + 131072.
\end{aligned}$$

We succeeded in recovering $\widetilde{m}(\underline{x})$ in Section 3.6.

5 Complexity Analysis

In this section, we investigate the complexity of the algorithm in Section 4.2. We analyze our attack according to the parameter sizes in Section 3.5 (cf. [Oku15]). To simplify the notations, we set $w := w_X$, assume $w = \sharp\Lambda_X$, and fix b. We show that the attack performs in polynomial time with respect to the parameters w and λ. The parameters d and e are $O\left(2^\lambda\right)$ and $O(w\lambda)$, respectively. Note that the size of each coefficient of F_j is $O(w\lambda)$ bits for $j = 1, 2, 3$ (see Section 3.5 for the representation of the parameters by w and λ).

Remark 5.0.1 First, let us determine the bit complexity of the computation of polynomials with integer coefficients in the algorithm. We suppose the arithmetic operations of addition and subtraction of two polynomials $F, G \in \mathbb{Z}[\underline{x}]$ are $O(\min\{q_F, q_G\})$ in $\mathbb{Z}$, where q_F and q_G are the number of the terms of F and G, respectively. Moreover, the arithmetic operations of multiplication of them are $O\left(\left(\max\{q_F, q_G\}\right)^2\right)$ in $\mathbb{Z}$. We compute ${F_1}' := F_1 - F_2$ and ${F_2}' := F_2 - F_3$ at the beginning of the algorithm. Note that the number of the terms of F_1, F_2 and F_3 are at most w^2. The sizes of the coefficients of F_j are $O(w\lambda)$ for $j = 1, 2, 3$. Thus the arithmetic complexity of computing F_1' and F_2' is $O\left(w^2\right)$, and its bit complexity is

$$O\left(w^2 (w\lambda)\right) = O\left(w^3\lambda\right). \tag{5.0.8}$$

We do such computations in (4.2.1)-(4.2.5). Actually the arithmetic complexity of (4.2.1)-(4.2.5) is negligible because we regard the coefficients of certain polynomials as variables. For example, in (4.2.1), we regard the coefficients of ${s_1}'$, ${s_2}'$ and g as variables. On the other hand, we compute $H_1 := F_1 - s_1 f' - rX$ in Step 3-4. In this case, we do not regard any coefficient as variables. Since

the number of the terms of s_1, f', r and X are w, we require $O\left(w^2\right)$ arithmetic operations for computing $s_1 f'$ and rX. In addition, the number of the terms of F_1, $s_1 f'$ and rX are $O\left(w^2\right)$. From this, we require $O\left(w^2\right)$ additions and subtractions. Here recall that the size of each coefficient of the polynomials F_1, $s_1 f'$ and rX is $O\left(w\lambda\right)$ bits. Thus the bit complexity of computing H_1 is

$$O\left(w^2\left(w\lambda\right)^2\right) \quad = \quad O\left(w^4\lambda^2\right). \tag{5.0.9}$$

Remark 5.0.2 Second, we solve one or two linear systems in each step of our attack. Then, we obtain one solution and the kernel lattice for each linear system. We assume that the bit complexity of solving a linear system is equivalent to the bit complexity of computing the Hermite Normal Form (HNF) of the augmented matrix of the system. According to Chapter 2 in [Gal12], we assume that the computation of the HNF of an $n \times m$ matrix $M = (M_{i,j})$ requires $O(nm^4(\log(\|M\|_\infty))^2)$ bit operations, where $\|M\|_\infty := \max_{i,j}\{|M_{i,j}|\}$.

To simplify the notations, we assume the sizes of the entries of one solution and an output basis of the kernel lattice of each linear system are $O\left(\ell\right)$ bits if the sizes of the entries of its augmented matrix are $O\left(\ell\right)$ bits.

Remark 5.0.3 Third, we discuss the size of the norm of a vector with integer entries. Let $\mathbf{a} = (a_1, \ldots, a_k) \in \mathbb{Z}^k$ be a vector with $|a_i| \leq 2^l$ $(i = 1, \ldots, k)$. Since $\|\mathbf{a}\| \leq \sqrt{k2^{2l}}$, the size of $\|\mathbf{a}\|$ is bounded by $\log\left(\sqrt{k2^{2l}}\right) = \log\left(k^{1/2}\right) + l = O\left(\log\left(k\right) + l\right)$ bits. Similarly, the size of $\|\mathbf{a}\|^2$ is $O\left(\log\left(k\right) + l\right)$ bits.

5.1 The Complexity of Step 1

Step 1-1. We estimate the bit complexity for solving the linear system $\mathbf{u}A = \mathbf{0}$ with at most $2w + w^2$ variables and w^3 equations. The size of each entry of A is $O\left(w\lambda\right)$ bits, and thus Step 1-1 requires

$$O\left(w^{16}\lambda^2\right) \tag{5.1.1}$$

bit operations from Remark 5.0.2. In addition, we note that the sizes of the entries of $\mathbf{u}'_1, \mathbf{u}'_2, \mathbf{u}'_3$, that are basis vectors of the kernel lattice $\mathcal{L}'_1$ of A, are $O\left(w\lambda\right)$ bits from Remark 5.0.2.

Step 1-2. In the beginning of this step, we compute UW, where U is a basis matrix of $\mathcal{L}_1$ with $3 \times 2w$ entries and W is a $(2w \times 2w)$ diagonal matrix. The arithmetic complexity of multiplying these matrices is $3 \cdot (2w) = O\left(w\right)$. Since the size of each entry of U and W is $O\left(w\lambda\right)$ bits, the multiplying runs in

$$O\left(w \cdot \left(w\lambda\right)^2\right) \quad = \quad O\left(w^3\lambda^2\right) \tag{5.1.2}$$

bit operations. We note that the size of each entry of UW is $O\left(w\lambda\right)$ bits. After the multiplying, we execute the LLL algorithm to the $2w$-dimensional lattice $f_W\left(\mathcal{L}_1\right)$ of 3-rank with the basis matrix UW. According to [LLL82], the complexity of the LLL algorithm requires $O\left(3^5\left(2w\right)\left(\log\left(2w \cdot 2^{2w\lambda}\right)\right)^3\right)$ bit operations in this case because the norms of the row vectors of UW are $O\left(\sqrt{2w \cdot 2^{2w\lambda}}\right)$. Thus the LLL algorithm of this step runs in

$$O\left(w^4\lambda^3\right) \tag{5.1.3}$$

bit operations. Any entry of the vectors of the LLL reduced basis is $O\left(\sqrt{3\left(w \cdot 2^{2w\lambda}\right)}\right)$ because the rank of $f_W(\mathcal{L}_1)$ is equal to 3, and because $\|\mathbf{u}_i W\|^2 = O\left(w \cdot 2^{2w\lambda}\right)$ for $i = 1, 2, 3$ and row vectors $\mathbf{u}_i$ of U. Thus the size of any entry of the basis matrix is $O(w\lambda)$ bits. We multiple the diagonal matrix W^{-1} by the LLL reduced basis matrix. The arithmetic complexity of the multiplying is $3 \cdot 2w = O(w)$. Thus the multiplying runs in

$$O\left(w(w\lambda)^2\right) \quad = \quad O\left(w^3\lambda^2\right) \tag{5.1.4}$$

bit operations.

5.2 The Complexity of Step 2

Step 2-1. In this step, we solve the linear system $\mathbf{v}B = \mathbf{b}$ with $3w$ variables and at most w^2 equations. In the same way as Step 1-1, the bit complexity of this step can be estimated as

$$O\left(w^{11}\lambda^2\right). \tag{5.2.1}$$

Every entry of a solution and basis vectors of the kernel lattice $\mathcal{L}_2$ has the size of $O(w\lambda)$ bits from the same reason as Step 1-1. Note that $\mathcal{L}_2$ is a $3w$-dimensional lattice of 1-rank. Hence the sizes of the norms of $\mathbf{v}_0$ and $\mathbf{v}_1$ are $O\left(\sqrt{(3w \cdot 2^{2w\lambda})}\right)$.

Step 2-2. In this step, we compute $\mathbf{v}_0' := \mathbf{v}_0 - \lfloor \langle \mathbf{v}_0, \mathbf{v}_1 \rangle / \langle \mathbf{v}_1, \mathbf{v}_1 \rangle \rceil \mathbf{v}_1$. This computation requires $O\left(2^4 (3w) \left(\log\left(3w \cdot 2^{2w\lambda}\right)\right)^2\right)$ bit operations according to Chapter 17 in [Gal12]. Hence Step 2-2 requires

$$O\left(w^3\lambda^2\right) \tag{5.2.2}$$

bit operations.

5.3 The Complexity of Step 3 and The Total Complexity of Our Attack

Step 3-1. We solve the linear system $\mathbf{w}C = \mathbf{c}$ with $3w$ variables and at most w^2 equations. In the same way as Steps 1-1 and 2-1, the computation requires

$$O\left(w^{11}\lambda^2\right) \tag{5.3.1}$$

bit operations. Every entry of a solution $\mathbf{w}_0$ and basis vectors $\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3$ of the kernel lattice $\mathcal{L}_3$ has the size of $O(w\lambda)$ bits. Note that the the norms of $\mathbf{w}_0$, $\mathbf{w}_1$, $\mathbf{w}_2$ and $\mathbf{w}_3$ are $O\left(\sqrt{(3w \cdot 2^{2w\lambda})}\right)$.

Step 3-2. We execute Babai's nearest plane algorithm to the $3w$-dimensional lattice $\mathcal{L}_3{}' := \langle \mathbf{w}_1, \mathbf{w}_2 \rangle$ of 2-rank and the vector $\mathbf{w}_0 + \mathbf{w}_3$. Before executing Babai's nearest plane algorithm, we execute the LLL algorithm to $\mathcal{L}_3{}'$. Since $\mathcal{L}_3{}'$ has 2-rank and $3w$-dimension, the LLL algorithm requires $O\left(2^5 (3w) \left(\log\left(3w \cdot 2^{2w\lambda}\right)\right)^3\right)$ bit operations. Thus the LLL algorithm in Step 3-2 requires $O\left(w^4\lambda^3\right)$ bit operations. The norm of any vector of the LLL reduced basis is $O\left(\sqrt{2\left(3w \cdot 2^{2w\lambda}\right)}\right)$ (cf. Chapter

17 in [Gal12]). Thus Babai's nearest plane algorithm requires $O\left(2^5\left(\log\left(\sqrt{2\left(3w\cdot 2^{2w\lambda}\right)}\right)\right)^2\right)$ bit operations. From this, the bit complexity of Babai's nearest plane algorithm is $O\left(w^2\lambda^2\right)$ in this case. Hence Step 3-2 runs in

$$O\left(w^4\lambda^3\right) \tag{5.3.2}$$

bit operations.

Step 3-3. We solve the linear system $\mathbf{x}H = \mathbf{h}$ with $2w$ variables and at most w^2 equations. The size of any entry of H and $\mathbf{h}$ is $O\left(w\lambda\right)$ bits. Hence Step 3-3 runs in

$$O\left(w^{16}\lambda^2\right) \tag{5.3.3}$$

bit operations.

Step 3-4. At the beginning of this step, we compute $e' := e^{-1} \bmod \varphi\left(d\right)$ by using the extended Euclid's algorithm. According to Remark 3.5 in [Oku15], the integer d should be chosen so that one can compute $\varphi\left(d\right)$ efficiently because the computation is needed in the decryption process (see [Oku15], Section 3.4). In Remark 3.5 of [Oku15], the integer d is expected to be a prime number as such an example. From this, we assume d is a prime number, and then we have $\varphi\left(d\right) = d-1$.
Next, we compute $c_{\underline{k}}\left(m'\right) := \mu^{e'} \pmod d$ $\left(0 < c_{\underline{k}}\left(m'\right) < d\right)$, where $e' := e^{-1} \pmod{\varphi(d)}$ and μ is a certain coefficient of $H_1\left(\underline{x}\right)$ (cf. Step 3-4 in Section 4.2). Recall that the bit sizes of e', μ and d are $O\left(\lambda\right)$, $O\left(w\lambda\right)$ and $O\left(\lambda\right)$, respectively. Thus this computation can be done in $O\left(w\lambda^2+\lambda^3\right)$ bit operations by the square-and-multiply algorithm for modular exponentiation.
Third, we compute $c_{\underline{k}}\left(\widetilde{m}\right) := \left(c_{\underline{k}}\left(m'\right)\right)^e \pmod{Nd}$ $\left(0 < c_{\underline{k}}\left(\widetilde{m}\right) < Nd\right)$. Note that the sizes of $c_{\underline{k}}\left(m'\right)$, e and Nd are $O\left(\lambda\right)$, $O\left(\log\left(w\lambda\right)\right)$ and $O\left(w\lambda\right)$ bits, respectively. Thus, the square-and-multiply algorithm requires $O\left(\left(w\lambda\right)^2\log\left(w\lambda\right)\right)$ bit operations to compute $c_{\underline{k}}\left(\widetilde{m}\right)$. As a consequence, those modular exponential arithmetic can be performed in $O\left(\lambda^3+w^2\lambda^2\log\left(w\lambda\right)\right)$ bit operations. Finally, the computation of $t := \left(\mu - c_{\underline{k}}\left(\widetilde{m}\right)\right)/c_{\underline{k}}\left(X\right)$ runs in $O\left(w^2\lambda^2\right)$ bit operations.
The total bit complexity of Step 3-4 is

$$O\left(\lambda^3+w^2\lambda^2\log\left(w\lambda\right)\right). \tag{5.3.4}$$

Putting all the steps together, namely considering (5.0.8)-(5.3.4), we can determine the complexity of our attack.

Theorem 5.3.1 The total bit complexity of the attack in Section 4.2 is

$$O\left(w^{16}\lambda^2\right) + O\left(w^4\lambda^3\right).$$

Consequently, our attack performs in polynomial time for all the parameters λ and w_X.

6 Experimental Results

We show experimental results[1] on our attack described in Section 4 against the DEC scheme for $n = 3$, i.e., the number of variables of a public key X is equal to 3. We conduct experiments for recommended parameters which can let the DEC have 128 bit security (cf. Section 3.5).

Experimental Procedure
For a public key X, let $\underline{k}$ be the maximal element in Λ_X with respect to the order described in Remark 3.1.2 (2). For $\underline{i} \in \Lambda_X \setminus \{\underline{k}, \underline{0}\}$, we suppose that $2^{b-1} \leq |c_{\underline{i}}(X)| < 2^b$ for some $b \in \mathbb{Z}_{>0}$. For given parameters w_X and Λ_X we repeat the following procedure 100 times:

1. Make a secret key and a public key according to Section 3.2.
2. By using the public key constructed above, make a ciphertext according to Section 3.3.
3. Execute **Algorithm of Proposed Attack** in Section 4.2 for the above public key and the ciphertext.

We count the number of successes and time if $\widetilde{m}$ or $-\widetilde{m}$ are recovered.

Table 1 shows our experimental results. In Step 1 of Table 1, we show the number of successes only if we succeed in recovering the target vector $(\mathbf{s}'_1, \mathbf{s}'_2)$ or $-(\mathbf{s}'_1, \mathbf{s}'_2)$ (see Sections 4.1 and 4.1.1). In Step 2 of Table 1, we show the number of successes only if the linear system obtained in Step 2 of our attack has a solution (see Section 4.1.2). In Step 3 of Table 1, we show the number of successes only if a twisted plaintext $\widetilde{m}$ or $-\widetilde{m}$ is recovered. From Table 1, we see that the weighted LLL algorithm found the target vector in Step 1 of our attack with probability being about from 70 to 90%. We consider that the success probability of Step 3 is sufficiently high for practical cryptanalysis. As we mentioned in Section 3.5, from the viewpoint of the efficiency of the key generation, encryption and decryption of DEC, those parameters in Table 1 are practical (see also Tables 4, 5 and 6 in Section 6 of [Oku15]). This suggests that our attack with the weighted LLL algorithm can break the one-wayness of DEC efficiently for practical parameters with sufficiently high probability.

Remark 6.0.2 In Section 4.1, we assume that ranks of some lattices occuring in Steps 1 and 3 of our attack are equal to 3 (see Assumptions 4.1.1 and 4.1.4). These assumptions are important to analyze the complexity of our attack. We also did experiments for the same instances of DEC as instances of the above experiments whether those assumptions are satisfied or not. As a result, those assumptions are satisfied for all the instances.

[1] We use a standard note PC with 2.60 GHz CPU (Intel Core i5), 16 GB memory and Mac OS X 64 bit. We implemented the attack in Magma V2.21-3 ([BCP97]).

Table 1: Experimental results on **Algorithm of Proposed Attack** in Section 4.2 against DEC with three variables of 128 bit security. We did experiments according to **Experimental Procedure** described in the beginning of Section 6. Parameters e and d are public keys and N is an integer chosen in the encryption process of DEC. The parameter b is the bit length of the coefficients of a public key X except the terms of its maximal degree and constant. "Average Time" means that the average of time for performing our attack. (We show the timing data in successful cases.)

Recommended parameters for DEC (Section 3.5)						Experimental results			
Total degree of a public key X	Number of monomials of X	Value of b	Sizes of e, d, N (bit)			Number of successes of Attack Algorithm / 100			Average time (sec.)
			e	d	N	Step 1	Step 2	Step 3	
10	3	10	11	65	714	80	80	27	0.02
10	4	10	11	65	714	79	79	23	0.03
10	5	10	11	65	714	87	87	24	0.04
10	6	10	11	65	714	87	87	22	0.06
10	7	10	11	65	714	93	93	29	0.08
10	8	10	11	65	714	96	96	40	0.10
10	9	10	11	65	714	88	88	30	0.16
10	10	10	11	65	714	92	92	36	0.24
10	3	50	11	65	712	79	79	22	0.03
10	4	50	11	65	711	81	81	29	0.03
10	5	50	11	65	713	88	88	33	0.04
10	6	50	11	65	712	86	86	40	0.07
10	7	50	11	65	712	92	92	38	0.08
10	8	50	11	65	711	89	89	32	0.11
10	9	50	11	65	712	87	87	22	0.18
10	10	50	11	65	711	94	94	33	0.26
10	3	100	11	65	711	80	80	30	0.02
10	4	100	11	65	711	79	79	28	0.03
10	5	100	11	65	711	84	84	31	0.04
10	6	100	11	65	712	88	88	31	0.07
10	7	100	11	65	712	89	89	33	0.08
10	8	100	11	65	711	88	88	31	0.10
10	9	100	11	65	712	86	86	37	0.18
10	10	100	11	65	713	91	91	32	0.29

In order to show the efficiency of our attack for large w_X and $\sharp\Lambda_X$, we show the average time for performing our attack and present sizes of public keys, secret keys and ciphertexts in Table 2. Note that Table 2 shows only the results of successful cases.

From the timing ("Average time") in Table 2, we see that our attack succeeds in recovering plaintexts in practical time for sufficiently large w_X and $\sharp\Lambda_X$. Note that DEC becomes impractical as w_X and $\sharp\Lambda_X$ get larger although from our complexity analysis in Section 5, the sizes of w_X and $\sharp\Lambda_X$ are deeply affect time for performing our attack. Thus, we infer that our attack via the weighted LLL algorithm can fully break the one-wayness of DEC.

Table 2: Experimental results with three variables for increasing w_X and $\sharp\Lambda_X$ and for 128 bit security. We did experiments according to **Experimental Procedure** described in the beginning of Section 6. The parameter w_X is the total degree of a public key X. The parameter $\sharp\Lambda$ is the number of terms of X. The parameter b is the bit length of the coefficients of X except the terms of its maximal degree and constant.

Parameters for DEC						Size of public key (bit)	Size of secret key (bit)	Size of ciphertext (bit)	Average time (sec.)
Total degree of a public key X	Number of monomials of X	Value of b	Sizes of e, d, N (bit)						
			e	d	N				
5	5	10	10	65	387	752	198	29,216	0.03
10	10	10	11	65	714	1,453	198	163,562	0.24
15	15	10	11	65	1,037	2,154	198	473,974	1.27
20	20	10	11	65	1,360	2,778	198	1,036,956	4.98
25	25	10	12	65	1,687	3,555	198	1,924,894	14.20
30	30	10	12	65	2,014	4,257	198	3,208,830	32.62
35	35	10	12	65	2,341	4,957	198	4,965,487	89.88
40	40	10	13	65	2,665	5,658	198	7,259,410	154.76
45	45	10	13	65	2,986	6,358	198	10,157,566	411.52

7 Conclusion and Future Work

In this paper, we proposed an attack against the one-wayness of the public key cryptosystem based on Diophantine equations of degree increasing type (DEC) via the weighted LLL algorithm. From this, we show that the one-wayness of DEC can be transformed to the problem of finding certain relatively shorter vectors in lattices of low ranks obtained by linearization techniques. Our most important target vector takes a special form: it is not necessarily shortest in some lattice of low rank but only some entries are relatively small. The usual LLL algorithm (with respect to the Euclidean norm) does not work well for finding such vectors in our attack.

Our heuristic analysis suggests that the target vector becomes a (nearly) shortest vector with respect to a weighted norm for some weight chosen appropriately. Therefore, we tried to apply the weighted LLL algorithm, which is the LLL algorithm with respect to the weighted norm, to our attack. From our complexity analysis and experimental results, we proved that by choosing an appropriate weight, our attack with the weighted LLL algorithm can break the one-wayness of DEC in polynomial time for all the parameters with sufficiently high probability under some assumptions.

The idea of the weighted LLL algorithm is known ([Mag]), but to the best of our knowledge, it has not been widely used in cryptography yet. Therefore, our results show that it is possible that the weighted LLL algorithm contributes to cryptography. Our future work is to develop the theory of the weighted LLL algorithm which may become a more useful tool for analyzing the security of other cryptosystems based on the difficulty of lattice problems.

References

[AG04] K. Akiyama, Y. Goto, *An Algebraic Surface Public-key Cryptosystem*, IEICE Technical Report, **104** (421), pp. 13–20, (2004).

[AG06] K. Akiyama, Y. Goto, *A Public-key Cryptosystem using Algebraic Surfaces*, In: Proceedings of PQCrypto., pp. 119–138, (2006), available at `http://postquantum.cr.yp.to/`.

[AG08] K. Akiyama, Y. Goto, *An improvement of the algebraic surface public-key cryptosystem*, In: Proceedings of 2008 Symposium on Cryptography and Information Security, SCIS 2008, CD-ROM, 1F1-2, (2008).

[AGM09] K. Akiyama, Y. Goto, H. Miyake, *An Algebraic Surface Cryptosystem*, In: Proceedings of PKC'09, Lecture Notes in Computer Science, **5443**, pp. 425–442, Springer, Berlin Heidelberg, (2009).

[Bab86] L. Babai, *On Lovász' lattice reduction and the nearest lattice point problem*, Combinatorica, **6** (1), pp. 1–13, (1986), (Preliminary version in STACS 1985).

[BHHKP14] A. Bérczes, L. Hajdu, N. Hirata-Kohno, T. Kovács, A. Pethö, *A key exchange protocol based on Diophantine equations and S-integers*, JSIAM Letters, **6** (0), pp. 85–88, (2014).

[BBD08] D. J. Bernstein, J. Buchmann, E. Dahmen (Eds.), *Post-Quantum Cryptography*, Springer-Verlag, Berlin Heidelberg, (2009).

[BCP97] W. Bosma, J. Cannon, C. Playoust, *The Magma algebra system. I. The user language*, Journal of Symbolic Computation, **24** (3-4), pp. 235–265, (1997).

[Cus95] T. W. Cusick, *Cryptoanalysis of a public key system based on diophantine equations*, Information Processing Letters, **56** (2), pp. 73–75, (1995).

[DGS06] J. Ding, J. E. Gower, D. S. Schmidt, *Multivariate Public Key Cryptosystems*, Advances in Information Security, **25**, Springer, US, (2006).

[DMR76] M. Davis, Y. Matijasevič, J. Robinson, *Hilbert's tenth problem, Diophantine equations: positive aspects of a negative solution*, Mathematical Developments Arising from Hilbert Problems, pp. 323–378, American Mathematical Society, Providence, RI., (1976).

[Eis07] K. Eisenträger, *Hilbert's Tenth Problem for function fields of varieties over number fields and p-adic fields*, Journal of Algebra, **310** (2), pp. 775–792, (2007).

[FS10] J. -C. Faugere, P. -J. Spaenlehauer, *Algebraic Cryptanalysis of the PKC'2009 Algebraic Surface Cryptosystem*, In: Proceedings of PKC'10, Lecture Notes in Computer Science, **6056**, pp. 35–52, Springer, Berlin Heidelberg, (2010).

[Gal12] S. D. Galbraith, *Mathematics of Public Key Cryptography*, Cambridge University Press, (2012).

[HP13] N. Hirata-Kohno, A. Pethö, *On a key exchange protocol based on Diophantine equations*, Infocommunications Journal, **5** (3), pp. 17–21, Scientific Association for Infocommunications (HTE), (2013).

[Iwa08] M. Iwami, *A Reduction Attack on Algebraic Surface Public-Key Cryptosystems*, Lecture Notes in Computer Science, **5081**, pp. 323–332, Springer, Berlin Heidelberg, (2008).

[LLL82] A. K. Lenstra, H. W. Lenstra, L. Lovász, *Factoring polynomials with rational coefficients*, Mathematische Annalen, **261** (4), pp. 515–534, Springer-Verlag, (1982).

[LCL95] C. H. Lin, C. C. Chang, R. C. T. Lee, *A new public-key cipher system based upon the diophantine equations*, IEEE Transactions on Computers, **44** (1), pp. 13–19, IEEE Computer Society Washington, DC, USA, (1995).

[Mag] The description of parameters of Magma's LLL available at `http://magma.maths.usyd.edu.au/magma/handbook/text/312#2934`.

[Oku15] S. Okumura, *A public key cryptosystem based on diophantine equations of degree increasing type*, Pacific Journal of Mathematics for Industry, **7**, Springer, Berlin Heidelberg, (2015), available at `http://link.springer.com/article/10.1186%2Fs40736-015-0014-4`.

[Phe91] T. Pheidas, *Hilbert's Tenth Problem for fields of rational functions over finite fields*, Inventiones mathematicae, **103** (1), pp. 1–8, Springer-Verlag, (1991).

[Sho97] P. Shor, *Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer*, SIAM Journal on Computing, **26** (5), pp. 1484–1509, Society for Industrial and Applied Mathematics Philadelphia, PA, USA, (1997).

[UT07] S. Uchiyama, H. Tokunaga, *On the Security of the Algebraic Surface Public-key Cryptosystems* (in Japanese), In: Proceedings of 2007 Symposium on Cryptography and Information Security, SCIS 2007, CD-ROM, 2C1-2, (2007).

[Vid94] C. R. Videla, *Hilbert's Tenth Problem for Rational Function Fields in Characteristic 2*, Proceedings of the American Mathematical Society, **120** (1), pp. 249–253, American Mathematical Society, (1994).

[Vol07] F. Voloch, *Breaking the Akiyama-Goto cryptosystem*, Contemporary mathematics, Arithmetic, Geometry, Cryptography and Coding Theory, **487**, pp. 113–118, American Mathematical Society, Providence, RI., (2007).

[Yos11] H. Yosh, *The Key Exchange Cryptosystem Used with Higher Order Diophantine equations*, International Journal of Network Security & Its Applications Journal, **3** (2), pp. 43–50, (2011).