Abstract — Lattice-based Cryptography is known as one of the key technologies in modern cryptography. This encryption scheme has the basis vectors from the lattice as the public key and a short-length vector in the lattice consisting of an integer combination of the basis vectors as the secret key. To break this encryption, we need to solve the Shortest Vector Problem (SVP), known as NP-hard. Therefore, instead of finding the shortest vector, LLL algorithm is often used to find a vector of sufficiently short length to break the encryption. The LLL algorithm is a well-known method for breaking this encryption, but there is still no clear answer to the question of how many times the LLL algorithm needs to be used to obtain the desired level of secret key, the average number of the \((\delta, \eta)\)-LLL bases in dimension \(n\) is a tool to measure the probability that the LLL algorithm solves the SVP. We can expect that this number indicates how many times the appropriate algorithm should run. There is a formula for this, but it contains some functions that take a long time to compute. We apply linear regression to the formula of the average number of the \((\delta, \eta)\)-LLL bases in dimension \(n\), and therefore we obtain some formulas to approximate the average number of the \((\delta, \eta)\)-LLL is based on dimension \(n\), which contains simple functions. When the dimensions are high, our model is much better regarding the computation time.

Keywords — Shortest vector; LLL-reduction algorithm; linear regression.

I. INTRODUCTION

Lattice-based cryptography represents a cryptographic framework grounded in the computational complexity of lattice-based problems, as initially expounded by Ajtai [1]. This paradigm embodies a prospective avenue within the domain of post-quantum cryptography. Notably, its utility extends to conventional cryptographic paradigms such as key exchange and digital signatures, as underscored by Nejatollahi et al. [2]. Moreover, this cryptographic approach exhibits considerable promise across diverse applications, including the realm of the Internet of Things (IoT), as articulated by Khalid et al. [3], and the sphere of medical data analytics, as elucidated by Kocabas and Soyata [4]. A foundation in linear algebra is imperative to comprehend the underpinnings of lattice-based cryptography effectively. Specifically, elucidating the concept of the span of a subset within a vector space serves as the inaugural step in this mathematical journey.

Recent research on this lattice encryption has taken a variety of forms. For example, there have been various studies on its mathematical properties and efficient algorithms [5]-[10], some variations [11], electronic voting [12], Blockchains [13], and its protection from fault or DPA [14], [15], and other invasions using the techniques of masking [16]. Furthermore, research on the hardware structure is ongoing [17], and their applications are used in the design of qTESLA [18], [19].

In the context of lattice theory, the Shortest Vector Problem (SVP) is a prominent challenge that garners attention. Identifying a vector possessing notable brevity within a lattice is significant as it emerges as a compelling contender for private key instantiation within the framework of lattice-based cryptography, as elucidated by Hoffstein et al. [20]. The algorithm known as LLL (Lenstra–Lenstra–Lovász), expounded upon by Lenstra, Lenstra, and Lovász in their seminal work [21], facilitates the efficient computation of reduced bases within a polynomial time complexity framework. This culminates in deriving a collection of vectors characterized by their brevity, rendering them apt for employment on a lattice basis. The resultant basis, termed an LLL-reduced basis, embodies a cornerstone outcome of the
LLL algorithm and has subsequently engendered several divergent iterations and adaptations.

There are so many variations of LLL algorithm. For example, the modified greedy LLL algorithm paralleled the greedy LLL algorithm [22], [23]. Furthermore, the LLL algorithm is used in Lattice-based cryptography and in many other fields. For example, L algorithm is used to construct half-Hadamard matrices [24], solve the hidden subset sum problem [25], compute multidimensional theta functions [26], construct Hermite Normal Form [27], detect periodicity in digital images [28], attack ECDSA [29].

In this paper, we use the statistical techniques of linear regression are employed in the investigation of the average number of (δ,η)-LLL bases within the n-dimensional space. The outcomes of this investigation yield a series of analytical expressions, encapsulating elementary functional relationships, that serve as approximations for the outcomes of this investigation. We start with a brief introduction of Lattice based cryptography.

Let us consider the simplest two-dimensional case. [30]

- To receive the message, Alice creates a public key, which is a sufficiently large natural number q.
- And then Alice creates two secret keys f and g that are satisfying f < √q/2, √q/4 < g < √q/2, and f and g are relatively prime.
- After that Alice creates another public key h ≡ f⁻¹g(mod q) and then publish the pair of public key (f, g).
- Bob wants to send a message m < √q/4 to Alice without being seen by Eve.
- So, Bob computes e ≡ rh + m (mod q), 0 < e < q with a random number r and send the ciphertext e to Alice.
- Alice computes a ≡ fe (mod q), 0 < a < q and then compute b ≡ f⁻¹a (mod g), 0 < b < g.
- Then b ≡ m.

Let us see an example with small numbers.

- Let q = 23, f = 2, and g = 3.
- Then h ≡ 2⁻¹·3 ≡ 12 · 3 ≡ 13 (mod 23).
- Let plaintext m = 2. With a random number r = 6, Bob computes e ≡ 6 · 13 + 2 ≡ 11 (mod 23) and send e.
- Alice computes a ≡ 2 · 11 ≡ 22 (mod 23), and compute b ≡ 2⁻¹·22 ≡ 2·1 ≡ 2 (mod 3) and b = m = 2.

Eve can try a brute-force attack, but it is not useful. So, Eve tries to find the private key (f, g) from the public key (q, h). If Eve knows a pair (F, G) such that Fh ≡ G (mod q), F = O(√q), and G = O(√q), then (F, G) behaves as if it were secret key. It is clear that Fh ≡ G (mod q) and Fh = G + qR with an unknown integer R as equivalent. Then finding (F, G) is equivalent to solve F(1, h) = R(0, q) = (F, G) with known vectors (1, h), (0, q) and unknown parameters F, G, and R. So, solving Lattice-based cryptography is equivalent to this problem: Find a sufficiently nonzero small vector between the integer combinations of known vectors. As the dimension increases, this problem becomes much more difficult because the number of integer combinations to consider increases dramatically. This feature has been used to ensure the security of Lattice-based cryptography in sufficiently high dimensions.

LLL algorithm, as introduced in Section 1, is an efficient method to find a sufficiently small vector by reducing given basis within a polynomial time complexity. From Eve’s point of view, it is important to know how many times this algorithm can be used to get a sufficiently small nonzero vector, and from Alice’s or Bob’s point of view, they need an indicator that they should finish sending messages before running the LLL algorithm that sufficient times.

Unfortunately, there is not yet enough research on this algorithm to know how many tries it takes to get a reasonably good small nonzero vector. We are simply utilizing the LLL algorithm in a way that allows us to run it a few times and see if we get a good vector. However, as the dimension is higher, you’re going to have to run these algorithms a really large number of times, and it’s going to take longer and longer to verify that the result of the algorithm is a reasonably short vector. So, there’s definitely a need for research on how many runs are the answer for the range of good choices.

In this paper, we use the following notation to define the set of integer combinations of basis vectors.

\[ \text{span}_\mathbb{Z}(S) = \left\{ \sum_{i=1}^{n} a_i v_i : a_1, \ldots, a_n \in \mathbb{Z} \right\} \]

where S = \{v_1, v_2, \ldots, v_n\}.

Let \( \mu_{ij} = v_i/|v_i|^2 \), \( v_i^* = v_i - \sum_{j=1}^{i-1} \mu_{ij} v_j^* \) for i ≥ 2.

Note that \( \{v_1^*, v_2^*, \ldots, v_n^*\} \) is an orthogonal basis and if this process is possible for given vector space, then we can find the shortest vector because at least one of \( v_i^* \) can be shortened to a shortest vector by just multiplying a scalar. But in general, \( \mu_{ij} = v_i/|v_i|^2 \) is not an integer for a lattice base \( \{v_1, v_2, \ldots, v_n\} \). Therefore, in the original LLL algorithm and its variants, use round (\( \mu \)) instead of \( \mu \).

For the purpose to see the quality of the output of LLL algorithm, we first check the pseudocode of the LLL algorithm as follows: [30]

- Input: a basis \( \{v_1, v_2, \ldots, v_n\} \) for \( \mathbb{R}^n \), \( 1 \leq \delta \leq 1 \).
- Output: a (δ, η)-LLL basis \( \{v_1, v_2, \ldots, v_n\} \).
- k = 2
- while k ≤ n,
  - for i = k - 1 to 1,
    - \( v_k = v_k - \text{round}(\mu_{kk})v_i \)
    - if \( |\delta|v_{k+1}^*| \leq ||v_k^* + \mu_{kk}v_{k+1}|| \)
      - \( k = k + 1 \)
    - else
      - \( v_{k+1} = v_k \)
      - \( v_k = \text{tmp} \)
      - \( k = \text{min}(k - 1, 2) \)
- Output \( \{v_1, v_2, \ldots, v_n\} \), a (δ, η)-LLL basis.
The output, a \((\delta, \eta)\) -LLL basis satisfies the following conditions: [31]
- \(|\mu_{ij}| \leq \eta\) for all \(j < i\).
- \(\delta \| v_{i+1} \| \leq \| v_{i+1} + \mu_{i+1} v_i \|\) for all \(i = 1, \ldots, n - 1\)

Note that the output can vary for the same lattice by starting with different basis vectors. Also, for any fixed lattice, a basis that contains all the shortest vectors is not unique in general. For example, let \(v_i = e_i\). Then \([v_1, v_2, \ldots, v_n]\) is a basis for \(\mathbb{Z}^n\) and the number of bases for \(\mathbb{Z}^n\) that contains all the shortest vector is \(2^n\) because \(v_i = \pm e_i\) for any \(i\), generates \(\mathbb{Z}^n\). Therefore, if the possible number of outputs for fixed LLL is sufficiently close to \(2^n\), then by using LLL algorithm, user expects that the output basis contains the shortest vector(s). For this reason, there are some results for the number of LLL basis and the author of [30] obtain the average number of the \((\delta, \eta)\)-LLL bases in dimension \(n\). That is,

\[
2(2 \eta)^{(n-1)(n-2)/2} \prod_{i=2}^{n} \frac{\eta}{\xi(i)} \prod_{i=2}^{n-1} \frac{1}{i(n-i)} \times \prod_{i=1}^{n-1} \int_{-\eta}^{\eta} \sqrt{\delta^2 - x^2} dx = 2(2 \eta)^{(n-1)(n-2)/2} \prod_{i=2}^{n} \frac{1}{\xi(i)} \prod_{i=2}^{n-1} \frac{1}{i(i-1)\pi^2} \zeta(i) \xi(i)
\]

where \(S_j(x)\) is the surface area of a sphere in \(\mathbb{R}^j\) of radius \(x\). The formula is the exact value of the average number but computing this value takes so long time as \(n\) increases so it is beautiful but somewhat impractical. This paper aims to find a function \(\delta\) and \(n\) after fixing \(\eta\) to 0.53. For all \(\delta\), we see that there is a difference in the computation time when the dimensionality is small, but as the dimensionality increases, there is no significant difference in the computation time. The computation time scales linearly with the number of dimensions.

The purpose of this study is to learn a kind of meta-model for equation (2) by statistical methods or machine learning to produce a value that approximates the true value faster than the number of possible bases resulting from the calculation of the LLL algorithm, and to produce similar results. However, although it is possible to produce input and output data for learning, it is not possible to learn a model in the usual way due to the difficulty in processing the value of the dependent variable, which is the prediction target, due to the floating-point problem of the computer.

Fig. 1 shows the time taken to compute equation (2) as a function of \(\delta\) and \(n\) after fixing \(\eta\) to 0.53. For all \(\delta\), we see that there is a difference in the computation time when the dimensionality is small, but as the dimensionality increases, there is no significant difference in the computation time. The computation time scales linearly with the number of dimensions.

The purpose of this study is to learn a kind of meta-model for equation (2) by statistical methods or machine learning to produce a value that approximates the true value faster than the number of possible bases resulting from the calculation of the LLL algorithm, and to produce similar results. However, although it is possible to produce input and output data for learning, it is not possible to learn a model in the usual way due to the difficulty in processing the value of the dependent variable, which is the prediction target, due to the floating-point problem of the computer.

II. MATERIALS AND METHOD

A. Material

The following experimental combinations of \(n, \delta, \) and \(\eta\) in formula (2) were used to produce input and output data for the LLL algorithm. There were 18 experimental combinations of \(n\) in increments of 10 from 30 to 200, 5 experimental combinations of \(\delta\) in increments of 0.5 from 0.75 to 0.95, and 3 experimental combinations of \(\eta\): 0.51, 0.52, and 0.53. The total number of combinations in the experiment is \(18 \times 5 \times 3 = 270\) combinations.

If we set \(\eta\) to 0.53, \(\delta\) to 0.95, and \(n\) to 200, the calculated value of equation (1) is about 2.81e+118532, which exceeds the number of floating-point digits of the computer, and the result becomes infinite. Therefore, in this study, we used the “Rmpfr” and “gmp” libraries of the R program, which can calculate large integers. These libraries require a large computational cost to produce results.

Fig. 1 shows the time taken to compute equation (2) as a function of \(\delta\) and \(n\) after fixing \(\eta\) to 0.53. For all \(\delta\), we see that there is a difference in the computation time when the dimensionality is small, but as the dimensionality increases, there is no significant difference in the computation time. The computation time scales linearly with the number of dimensions.

The purpose of this study is to learn a kind of meta-model for equation (2) by statistical methods or machine learning to produce a value that approximates the true value faster than the number of possible bases resulting from the calculation of the LLL algorithm, and to produce similar results. However, although it is possible to produce input and output data for learning, it is not possible to learn a model in the usual way due to the difficulty in processing the value of the dependent variable, which is the prediction target, due to the floating-point problem of the computer.

\[
\eta = 0.53
\]

Fig. 1 LLL basis computation time for \(\delta\) and dimension number combinations.

B. Method

To solve the problem mentioned in the previous section, this study proposes a method that separates the integer and exponential parts of a value that exceeds the number of floating-point digits, predicts them separately, and combines the two prediction results. The exponential part is known to be proportional to \(n^3\), so we use it to learn with a linear regression model, while the integer part is filled with as many numbers as the computer's floating-point digits, so it is difficult to find any rules for the dimensionality. To solve this problem, we propose a method that cuts out numbers below a certain number of digits from the integer data to be trained and adds digits proportional to the number of dimensions to predict approximate values, but not true values. Also, since it
predicts approximate values for true values, it predicts an interval with a (1-\(\alpha\))% confidence interval for the predicted value.

To summarize the above algorithm:
- Step 1: Separate dependent variable values into integer and exponential parts
- Step 2: Predict the exponent part by computing \(y = f(x) + \varepsilon\).
- Step 3: Scale the number of digits in the integer part of the data to be proportional to the dimension
- Step 4: By using \(\log(y) = f(x) + \varepsilon\) predict the integer part and inverse transform
- Step 5: Combine the predicted integer part and exponent part

We fix the values of \(\delta\) and \(\eta\) in the algorithm suggested in this study. \(\eta = 0.53\), and \(\delta = 0.95\).

III. RESULTS AND DISCUSSION

The relationship between the exponential part and the integer part is in the form of a cubic polynomial as shown in Fig. 2. Therefore, the exponential part was predicted by fitting the polynomial regression model in equation (4). The regression coefficients in the regression model were estimated using the least squares method.

\[
y = \beta_0 + \beta_1 x^3 + \beta_2 x^2 + \beta_3 x + \varepsilon
\]  

(4)

Fig. 2 Relationship between exponential parts and number of dimensions.

After fitting the model, the corrected coefficient of determination was 1, and the regression coefficients for each term were all statistically significant at the 0.001 level of significance (\(\alpha\)). Fig. 3 is a scatter plot of the true and predicted values. The exponential part of the prediction was almost identical to the true value.

Table shows the 95% confidence intervals for the predicted values of the exponential regression model. In all dimensions, we predicted intervals containing the true value, with a range of 18 on average, which is very close to the true value, and the time required for this prediction is less than one second.

### TABLE I

<table>
<thead>
<tr>
<th>Number of dimension (n)</th>
<th>Lower bound</th>
<th>Exact value</th>
<th>Upper bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>306</td>
<td>323</td>
<td>327</td>
</tr>
<tr>
<td>40</td>
<td>801</td>
<td>809</td>
<td>820</td>
</tr>
<tr>
<td>50</td>
<td>1632</td>
<td>1636</td>
<td>1650</td>
</tr>
<tr>
<td>60</td>
<td>2892</td>
<td>2897</td>
<td>2910</td>
</tr>
<tr>
<td>70</td>
<td>4677</td>
<td>4683</td>
<td>4695</td>
</tr>
<tr>
<td>80</td>
<td>7079</td>
<td>7088</td>
<td>7097</td>
</tr>
<tr>
<td>90</td>
<td>10195</td>
<td>10205</td>
<td>10213</td>
</tr>
<tr>
<td>100</td>
<td>14117</td>
<td>14129</td>
<td>14135</td>
</tr>
<tr>
<td>110</td>
<td>18940</td>
<td>18953</td>
<td>18958</td>
</tr>
<tr>
<td>120</td>
<td>24758</td>
<td>24771</td>
<td>24776</td>
</tr>
<tr>
<td>130</td>
<td>31666</td>
<td>31677</td>
<td>31684</td>
</tr>
<tr>
<td>140</td>
<td>39757</td>
<td>39767</td>
<td>39775</td>
</tr>
<tr>
<td>150</td>
<td>49127</td>
<td>49134</td>
<td>49145</td>
</tr>
<tr>
<td>160</td>
<td>59868</td>
<td>59874</td>
<td>59886</td>
</tr>
<tr>
<td>170</td>
<td>72076</td>
<td>72081</td>
<td>72094</td>
</tr>
<tr>
<td>180</td>
<td>85844</td>
<td>85850</td>
<td>85862</td>
</tr>
<tr>
<td>190</td>
<td>101267</td>
<td>101275</td>
<td>101285</td>
</tr>
<tr>
<td>200</td>
<td>118437</td>
<td>118453</td>
<td>118458</td>
</tr>
</tbody>
</table>

Fig. 4 is a scatter plot of the predicted and true values of the integer part. The modified coefficient of determination is 0.998, which is slightly lower than the prediction performance of the exponential part, but it is analyzed that it is possible to predict the interval of the true value with some accuracy.

Table 2 shows the 95% confidence intervals for the integer part. The output of the LLL algorithm was able to approximate the interval with the true value for the number of possible bases.

### TABLE II

<table>
<thead>
<tr>
<th>Number of dimension (n)</th>
<th>Lower bound</th>
<th>Exact value</th>
<th>Upper bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>6.63e+77</td>
<td>2.43e+78</td>
<td>1.53e+79</td>
</tr>
<tr>
<td>40</td>
<td>7.34e+77</td>
<td>3.89e+78</td>
<td>1.42e+79</td>
</tr>
<tr>
<td>50</td>
<td>7.87e+77</td>
<td>7.44e+78</td>
<td>1.37e+79</td>
</tr>
<tr>
<td>60</td>
<td>8.24e+77</td>
<td>1.83e+78</td>
<td>1.35e+79</td>
</tr>
<tr>
<td>70</td>
<td>8.47e+77</td>
<td>2.66e+78</td>
<td>1.36e+79</td>
</tr>
<tr>
<td>80</td>
<td>8.62e+77</td>
<td>3.06e+78</td>
<td>1.39e+79</td>
</tr>
<tr>
<td>90</td>
<td>8.73e+77</td>
<td>8.38e+78</td>
<td>1.42e+79</td>
</tr>
<tr>
<td>100</td>
<td>8.84e+77</td>
<td>3.17e+78</td>
<td>1.45e+79</td>
</tr>
<tr>
<td>110</td>
<td>8.98e+77</td>
<td>1.61e+78</td>
<td>1.49e+79</td>
</tr>
<tr>
<td>120</td>
<td>9.16e+77</td>
<td>1.63e+78</td>
<td>1.52e+79</td>
</tr>
<tr>
<td>130</td>
<td>9.37e+77</td>
<td>7.01e+78</td>
<td>1.54e+79</td>
</tr>
<tr>
<td>140</td>
<td>9.62e+77</td>
<td>3.73e+78</td>
<td>1.56e+79</td>
</tr>
<tr>
<td>150</td>
<td>9.87e+77</td>
<td>9.22e+78</td>
<td>1.59e+79</td>
</tr>
<tr>
<td>160</td>
<td>1.01e+78</td>
<td>5.03e+78</td>
<td>1.62e+79</td>
</tr>
<tr>
<td>170</td>
<td>1.02e+78</td>
<td>3.51e+78</td>
<td>1.68e+79</td>
</tr>
<tr>
<td>180</td>
<td>1.01e+78</td>
<td>2.16e+78</td>
<td>1.76e+79</td>
</tr>
<tr>
<td>190</td>
<td>9.81e+77</td>
<td>9.53e+78</td>
<td>1.90e+79</td>
</tr>
<tr>
<td>200</td>
<td>9.22e+77</td>
<td>2.81e+78</td>
<td>2.12e+79</td>
</tr>
</tbody>
</table>
ACKNOWLEDGMENT

This research was supported by the 2023 scientific promotion program funded by Jeju National University.

REFERENCES


