A Quasilinear-Time CVP Algorithm for Triangular Lattice Based Fuzzy Extractors and Fuzzy Signatures

Kenta Takahashi[*] and Wataru Nakamura[†]
[*]Hitachi, Ltd., Yokohama, Japan
E-mail: kenta.takahashi.bw@hitachi.com
[†]Hitachi, Ltd., Yokohama, Japan
E-mail: wataru.nakamura.va@hitachi.com

***Abstract*— Fuzzy Extractor (FE) and Fuzzy Signature (FS) are useful schemes for generating cryptographic keys from fuzzy data such as biometric features. To implement FE and FS for fuzzy data in a Euclidean space, such as facial feature vectors, several techniques have been proposed that use lattice-based error correction. In these techniques, it is necessary to find the closest vector in the lattice to a given fuzzy data at the time of key reproduction or signing. However, the closest vector problem (CVP) in a general lattice is known to be NP-hard. Therefore, it is necessary to use a special lattice in which an efficient CVP algorithm can be constructed. In this paper, we propose a faster algorithm for CVP in triangular lattices with $O(n \log n)$-time compared to the conventional $O(n^2)$-time algorithm.**

I. INTRODUCTION

Fuzzy Extractor (FE) [1] is a general method to extract the same random numbers from fuzzy data within a certain error margin. FE enables the realization of cryptographic systems using biometric information or PUFs (Physically Unclonable Function) [2] as key management mediums. Similarly, Fuzzy Signature (FS) [3] is a digital signature scheme that uses fuzzy data itself as a signing key.

The specific algorithms of FE and FS depend on the metric space to which the fuzzy data belong and the error range to be tolerated. For example, if the fuzzy data is in n-bit Hamming space and you want to tolerate up to t-bit errors, there are known FE algorithms using n-bit error correcting codes that can correct t-bit errors [1]. With respect to the strength of the keys to be extracted, the error correcting code (ECC) should have as many codewords as possible for given (n, t). In this sense, it is ideal for the ECC to be a *perfect code*[1], which divides the n-bit space completely with Hamming spheres of radius t.

On the other hand, in biometric authentication methods such as face recognition, features are often extracted as n-dimensional vectors by machine learning or other methods, and errors are often evaluated as Euclidean distance[2]. Applying FE and FS here requires an error correction mechanism in Euclidean space, but since n-dimensional space cannot be divided by n-dimensional hyperspheres, an approximate division must be used instead. As a technique for this purpose, a method using an n-dimensional lattice structure has been proposed [6,7,8]. The desirable properties of the lattice to be applied to FE and FS are as follows.

(1) The lattice gives a dense sphere packing.
(2) The closest lattice point to a given arbitrary vector can be computed efficiently.

The first property is important to ensure the strength of the key to be extracted. Such lattice structures have been studied for many years in the context of the *sphere packing problem* [9], but for the densest structures it is an open problem except in $n = 1, 2, 3, 8$ and 24 dimensions. The second requirement is also important to optimize the balance between security and performance. However, the CVP in general lattices is known to be NP-hard [10].

As a lattice structure that satisfies these properties to some extent, Yoneyama et al. proposed the use of triangular lattices and constructed an efficient algorithm for CVP with $O(n^2)$-time complexity for n-dimensions by exploiting the high symmetry of the lattice [6]. However, when applied to biometric systems, this algorithm may take too much time to compute. For example, the typical number of feature dimensions in face recognition is around $n = 128 \sim 512$, and when applied to a 1:N authentication system with $N = 100{,}000$, the evaluated processing time is $3.8 \sim 62$ seconds, as shown in sections IV.B.

In this paper, we propose a fast CVP algorithm for triangular lattices with a computational time complexity of $O(n \log n)$. The main ideas are as follows. The first is to speed up the coordinate transformation subroutine, which conventionally took $O(n^2)$-time, to $O(n)$-time by using a kind of memorization technique that exploits the properties of the expression of the basis matrix of the triangular lattice. The second is to speed up

[1] Although there are a limited number of (n, t) pairs for which a perfect code exists.
[2] Feature extraction methods such as CosFace [4] and ArcFace [5], where a feature vector is $L2$−normalized and the 'closeness' between two vectors $\boldsymbol{x}, \boldsymbol{y}$ is defined by the cosine similarity $S_C(\boldsymbol{x}, \boldsymbol{y}) := \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{\|\boldsymbol{x}\| \|\boldsymbol{y}\|} = \boldsymbol{x} \cdot \boldsymbol{y}$, are often used for face recognition. This is equivalent to using Euclidean distance, since $\|\boldsymbol{x} - \boldsymbol{y}\|^2 = 2(1 - S_C(\boldsymbol{x}, \boldsymbol{y}))$.

the subroutine that searches for the closest vector from n candidates, which conventionally took $O(n^2)$-time, to $O(n)$-time by showing that the sequence of distances from the target vector to the suitably sorted candidates is convex and introducing a binary search.

II. Preliminaries and Related Works

A. Triangular Lattice

In an n-dimensional real Euclidean space $\mathbb{R}^n$, take n independent vectors $\boldsymbol{b}^1, \cdots, \boldsymbol{b}^n \in \mathbb{R}^n$ (basis vectors) and let $B = (\boldsymbol{b}^1, \cdots, \boldsymbol{b}^n)$ be a matrix (basis matrix). Then the lattice $L(B)$ is defined as

$$L(B) := \{B\boldsymbol{x} \mid \boldsymbol{x} \in \mathbb{Z}^n\}.$$

That is, $L(B)$ is the set of all vectors $\boldsymbol{b}^1, \cdots, \boldsymbol{b}^n$ linearly combined with integer coefficients.

A lattice L is called a triangular lattice if it has a basis $B = (\boldsymbol{b}^1, \cdots, \boldsymbol{b}^n)$ satisfying the following conditions.

$$\forall_i \left\|\boldsymbol{b}^i\right\| = c \text{ (constant)}, \qquad \forall_{i \neq j}\, \boldsymbol{b}^i \cdot \boldsymbol{b}^j = \frac{c^2}{2}.$$

The above conditions imply that an angle between any two basis vectors is π/3 (Figure 1). Triangular lattices have been proven to give the densest sphere-packing in two and three dimensions. It has also been shown experimentally to have relatively good performance in higher dimensions when applied to FE [6].

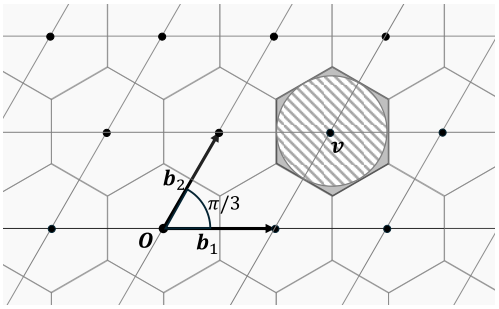

Figure 1: Black dots are the points of a triangular lattice in $\mathbb{R}^2$ spanned by $\boldsymbol{B} = (\boldsymbol{b^1}, \boldsymbol{b^2})$. The gray hexagon represents a Voronoi cell centered at lattice point $\boldsymbol{v}$. The shaded disk represents a sphere with center $\boldsymbol{v}$ whose radius is half the minimum distance.

For simplicity, and without loss of generality, the following discussion deals with the triangular lattice with $c = 1$ and a basis matrix expressed as

$$B = \begin{pmatrix} \alpha_1 & \beta_1 & \beta_1 & \cdots & \beta_1 \\ 0 & \alpha_2 & \beta_2 & \cdots & \beta_2 \\ 0 & 0 & \alpha_3 & \cdots & \beta_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \alpha_n \end{pmatrix}. \tag{1}$$

where α_k and β_k are constants computed by the following simultaneous difference equations:

$$\alpha_k = \sqrt{1 - \sum_{i=1}^{k-1} \beta_i^2}, \qquad \beta_k = \frac{\frac{1}{2} - \sum_{i=1}^{k-1} \beta_i^2}{\alpha_k} \quad (k \in [n]), \tag{2}$$

where $[n]$ denotes the set $\{1, \cdots, n\}$. The first few values are as follows:

$$\alpha_1 = 1, \qquad \alpha_2 = \frac{\sqrt{3}}{2}, \qquad \alpha_3 = \frac{\sqrt{6}}{3}, \qquad \cdots,$$

$$\beta_1 = \frac{1}{2}, \qquad \beta_2 = \frac{\sqrt{3}}{6}, \qquad \beta_3 = \frac{\sqrt{6}}{12}, \qquad \cdots.$$

B. Conventional CVP Algorithm for Triangular Lattices

A CVP algorithm takes as input a target vector $\boldsymbol{y} \in \mathbb{R}^n$ and outputs the closest lattice point. In the following, we outline the conventional CVP algorithm for triangular lattices proposed by Yoneyama et al. See reference [6] for details.

Conventional CVP Algorithm $CV(\boldsymbol{y})$

Input: $\boldsymbol{y} \in \mathbb{R}^n$

1: $(x_1, \cdots, x_n)^t \leftarrow B^{-1}\boldsymbol{y}$

2: **for** $i \leftarrow 1, \cdots, n$ **do**

$w_i \leftarrow \lfloor x_i \rceil$

$r_i \leftarrow x_i - w_i$

end for

$\boldsymbol{w} \leftarrow (w_1, \cdots, w_n)^t$

3: $\left(r_{\sigma(1)}, \cdots, r_{\sigma(n)}\right) \leftarrow Sort(r_1, \cdots, r_n)$ // descending order

4: **for** $k \leftarrow 0, \cdots, n$ **do**

$\boldsymbol{z}^k \leftarrow (z_1^k, \cdots, z_n^k)^t$ where

$$z_j^k \leftarrow \begin{cases} 1, & \text{for } j \leftarrow \sigma(1), \cdots, \sigma(k) \\ 0, & \text{for } j \leftarrow \sigma(k+1), \cdots, \sigma(n) \end{cases}$$

end for

5: $\boldsymbol{z} \leftarrow argmin_{\boldsymbol{z}^k \in \{\boldsymbol{z}^0, \cdots, \boldsymbol{z}^n\}} \|B\boldsymbol{r} - B\boldsymbol{z}^k\|$

6: **return** $B(\boldsymbol{w} + \boldsymbol{z})$

The time complexity of each step is $O(n^2)$ for step 1, 4 and 6, $O(n)$ for step 2 and $O(n \log n)$ for step 3. Step 5 appears to take $O(n^3)$-time at first glance, but can be computed in $O(n^2)$-time by using the following difference equation:

$$B\boldsymbol{z}^n = 0,$$
$$B\boldsymbol{z}^{k-1} = B\boldsymbol{z}^k + \boldsymbol{b}_{\sigma(k)} \ (\text{for } k = n, n-1, \cdots, 1).$$

Therefore, the total complexity is $O(n^2)$.

III. Proposed Algorithm

Based on the conventional algorithm, let us consider speeding up steps 1, 4, 5, and 6, which take O(n^2)-time. Specifically, the coordinate transformation process (product of basis matrix and vector) in steps 1 and 6 and the closest vector search (enumeration of candidate vectors and distance calculation) in steps 4 and 5 are accelerated as subroutines, respectively.

A. Coordinate Transformation Subroutine

Let us consider algorithms for mutual transformation between $\boldsymbol{y} = (y_1, \cdots, y_n)^t$ (Cartesian coordinate system) and $\boldsymbol{x} = (x_1, \cdots, x_n)^t$ (triangular lattice basis coordinate system) that satisfies the following relation:

$$\boldsymbol{y} = B\boldsymbol{x} \ \Leftrightarrow \boldsymbol{x} = B^{-1}\boldsymbol{y}.$$

In general, the product of a matrix and a vector requires $O(n^2)$-time, but by utilizing the basis matrix representation in Eq (1), a linear-time algorithm can be constructed as follows. Defining s_k $(k \in [n])$ as

$$s_k := \sum_{i=k+1}^{n} x_k \ ,$$

we have

$$s_n = 0, \tag{3}$$
$$s_{k-1} = s_k + x_k \ (k = 2, \cdots, n). \tag{4}$$

Furthermore, from Eq (1) the following equation holds:

$$y_k = \alpha_k x_k + \beta_k s_k \ (k = 1, \cdots, n). \tag{5}$$

Therefore, given $(x_1, \cdots, x_n)$, $(y_1, \cdots, y_n)$ can be obtained by solving equation (3), (4) and (5) sequentially in the following order: $y_n \to s_{n-1} \to y_{n-1} \to s_{n-2} \to \cdots \to s_1 \to y_1$. Similarly, given $(y_1, \cdots, y_n)$, we can obtain $(x_1, \cdots, x_n)$ by solving equation (3), (4) and (5) sequentially in the following order: $x_n \to s_{n-1} \to x_{n-1} \to s_{n-2} \to \cdots \to s_1 \to x_1$. The time complexity of these algorithms is $O(n)$.

Triangular to Cartesian Coordinates $T2C(\boldsymbol{x})$

Input: $\boldsymbol{x} = (x_1, \cdots, x_n)^t \in \mathbb{R}^n$

1: $s_n \leftarrow 0$

2: **for** $k \leftarrow n, n-1, \cdots, 1$ **do**

$y_k \leftarrow \alpha_k x_k + \beta_k s_k$

$s_{k-1} \leftarrow s_k + x_k$

end for

3: **return** $\boldsymbol{y} = (y_1, \cdots, y_n)^t$

Cartesian to Triangular Coordinates $C2T(\boldsymbol{y})$

Input: $\boldsymbol{y} = (y_1, \cdots, y_n)^t \in \mathbb{R}^n$

1: $s_n \leftarrow 0$

2: **for** $k \leftarrow n, n-1, \cdots, 1$ **do**

$x_k \leftarrow \dfrac{y_k - \beta_k s_k}{\alpha_k}$

$s_{k-1} \leftarrow s_k + x_k$

end for

3: **return** $\boldsymbol{x} = (x_1, \cdots, x_n)^t$

B. Closest Vector Search Subroutine

Here we describe an idea to speed up steps 4 and 5 of the conventional CVP algorithm. The goal here is to efficiently determine the k that minimize $d_k := \|B\boldsymbol{r} - B\boldsymbol{z}^k\|^2$ without calculating $\boldsymbol{z}^k$ and d_k for each $k = 0, 1, \cdots, n$. d_k can be expanded as follow:

$$\begin{aligned} d_k &= \|B\boldsymbol{r}\|^2 + \|B\boldsymbol{z}^k\|^2 - 2B\boldsymbol{r} \cdot B\boldsymbol{z}^k \\ &= \|B\boldsymbol{r}\|^2 + \left\|\boldsymbol{b}^{\sigma(1)} + \cdots + \boldsymbol{b}^{\sigma(k)}\right\|^2 - 2(r_1\boldsymbol{b}^1 + \cdots r_n\boldsymbol{b}^n) \\ &\qquad \cdot \left(\boldsymbol{b}^{\sigma(1)} + \cdots + \boldsymbol{b}^{\sigma(k)}\right) \\ &= \|B\boldsymbol{r}\|^2 + \frac{1}{2}k(k+1) \\ &\qquad - k(r_1 + \cdots r_n) - \left(r_{\sigma(1)} + \cdots + r_{\sigma(k)}\right) \\ &= \|B\boldsymbol{r}\|^2 + \frac{1}{2}k^2 + k\left(\frac{1}{2} - r_{sum}\right) - \sum_{i=1}^{k} r_{\sigma(i)}, \end{aligned}$$

where $r_{sum} := \sum_{i=1}^{n} r_i$. Therefore the first and second order differences of d_k are:

$$\Delta_k := d_k - d_{k-1} = k - r_{sum} - r_{\sigma(k)},$$
$$\Delta_k^2 := \Delta_k - \Delta_{k-1} = 1 - \left(r_{\sigma(k)} - r_{\sigma(k-1)}\right).$$

Since $0 \le r_i < 1$ $(i \in [n])$ we have $\Delta_k^2 > 0$. Therefore, if we regard d_k as a (discrete) function of k, it is convex downwards, and the k that gives the minimum value is:

i. if $\Delta_1 \ge 0$ then $k = 0$,
ii. if $\Delta_n \le 0$ then $k = n$,
iii. otherwise the $k \in [n-1]$ satisfying $\Delta_k \le 0 \le \Delta_{k+1}$.

In case iii, k can be found in $O(\log n)$ time by a binary search for Δ_k.

C. *Fast CVP Algorithm*

The following is the proposed CVP algorithm that uses the subroutine described in sections III.A and III.B to speed up the conventional algorithm.

Proposed Fast CVP Algorithm $FastCV(\boldsymbol{y})$

Input: $\boldsymbol{y} \in \mathbb{R}^n$

1: $(x_1, \cdots, x_n)^t \leftarrow C2T(\boldsymbol{y})$
2: **for** $i \leftarrow 1, \cdots, n$ **do**
  $w_i \leftarrow \lfloor x_i \rceil$
  $r_i \leftarrow x_i - w_i$
  end for
  $\boldsymbol{w} \leftarrow (w_1, \cdots, w_n)^t$
3: $(r_{\sigma(1)}, \cdots, r_{\sigma(n)}) \leftarrow Sort(r_1, \cdots, r_n)$ //descending order.
4: $r_{sum} \leftarrow \sum_{i=1}^{n} r_i$
5: **If** $1 - r_{sum} - r_{\sigma(1)} \ge 0$ **then**
  $k \leftarrow 0$
  else if $n - r_{sum} - r_{\sigma(n)} \le 0$ **then**
  $k \leftarrow n$
  Otherwise
  determine $k \in [n-1]$ satisfying the following inequality by a binary search:
  $k - r_{\sigma(k)} \le r_{sum} \le k + 1 - r_{\sigma(k+1)}$
  end if
6: $\boldsymbol{z} \leftarrow (z_1, \cdots, z_n)^t$ where

$$z_j \leftarrow \begin{cases} 1, & \text{for } j \leftarrow \sigma(1), \cdots, \sigma(k) \\ 0, & \text{for } j \leftarrow \sigma(k+1), \cdots, \sigma(n) \end{cases}$$

7: **return** $T2C(\boldsymbol{w} + \boldsymbol{z})$

IV. Evaluation

A. *Asymptotic Complexity*

Let us evaluate the time complexity of the proposed CVP algorithm $FasctCV$. The complexity of coordinate transformation algorithms $C2T$ and $T2C$ in Steps 1 and 7 is $O(n)$-time as described in III.A. Step 3 can be computed in $O(n \log n)$-time. Step 5 is a subroutine described in III.B and can be computed in $O(\log n)$-time. Since other steps are $O(n)$-time, the total complexity of $FastCV$ is $O(n \log n)$-time.

B. *Experimental Evaluation*

We implemented the proposed CVP algorithm and the conventional algorithm to evaluate their computation times experimentally. We randomly selected 10,000 real vectors $\boldsymbol{y}_1, \cdots, \boldsymbol{y}_{10000}$ from the range $\boldsymbol{y}_i \in [-100, 100]^n$ as target vectors, executed $CV(\boldsymbol{y}_i)$ and $FastCV(\boldsymbol{y}_i)$ for all $\boldsymbol{y}_i$ $(i = 1, \cdots, 10000)$, measured the total computation time for each algorithm, and evaluated the average time per run for each algorithm. The results for $n = 128, 256$, and 512 are shown in Table 1 and Figure 2. The computing environment is as follows: OS: Windows 10 Pro, CPU: Intel(R) Core(TM) i7-8700K @ 3.70GHz, memory 32GB, storage: SSD.

The proposed algorithm is about 6.6 times faster than the conventional algorithm when $n = 128$, about 12 times faster when $n = 256$, and about 24 times faster when $n = 512$. The number of processes per second when $n = 512$ is 3.82×10^4 with the proposed algorithm while it is 1.60×10^3 with the conventional algorithm.

Table 1: Average Computation time of the algorithms (μs)

Algorithm	$n = 128$	$n = 256$	$n = 512$
Conventional	38.2	156	623
Proposed	5.8	12.5	26.2

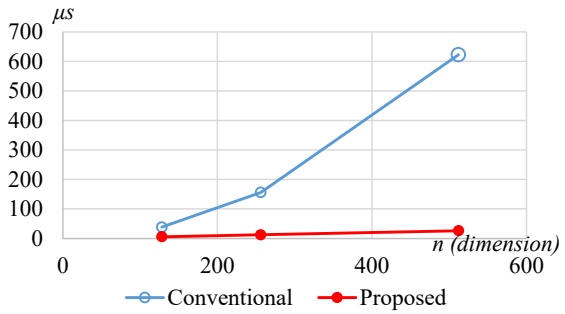

Fig. 2 Graphical comparison of the computation time of the algorithms.

V. Conclusions

In this paper, we proposed a fast algorithm for the closest vector problem (CVP) on high-dimensional triangular lattices. The time complexity of the proposed algorithm is $O(n \log n)$, whereas that of conventional algorithms is $O(n^2)$ for n dimensions. Experimental evaluation shows that the proposed algorithm achieves a speedup of about 24 times when $n = 512$. The CVP algorithm for high-dimensional triangular lattices is useful for constructing Fuzzy Extractors (FE) and Fuzzy Signatures (FS) using biometric data such as facial feature vectors. Therefore it is expected to realize real-time biometric identification for large-scale users with biometric template protection [11] based on FE and FS such as the Public Biometric

Infrastructure (PBI) [3]. A more detailed evaluation (e.g., using large-scale face datasets) is a subject for future work.

REFERENCES

[1] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin and Adam Smith, "Fuzzy extractors: How to generate strong keys from biometrics and other noisy data," *SIAM Journal on Computing,* Vol.38, No.1, 97–139. 2008.

[2] Roel Maes, "Physically unclonable functions: Constructions, Properties and Applications," Springer. ISBN 978-3-642-41395-7, 2013.

[3] Kenta Takahashi, Takahiro Matsuda, Takao Murakami, Goichiro Hanaoka and Masakatsu Nishigaki, "Signature schemes with a fuzzy private key," in *International Journal of Information Security,* vol. 18 (2019), 581–617.

[4] Hao Wang, Yitong Wang, Zheng Zhou, Xing Ji, Dihong Gong, Jingchao Zhou, Zhifeng Li and Wei Liu, "CosFace: Large Margin Cosine Loss for Deep Face Recognition," in Proc. of CVPR, 2018.

[5] Jiankang Deng, Jia Guo, Niannan Xue and Stefanos Zafeiriou, "ArcFace: Additive Angular Margin Loss for Deep Face Recognition," in Proc. of CVPR, 2019.

[6] Yuta Yoneyama, Kenta Takahashi and Masakatsu Nishigaki, "Closest Vector Problem on Triangular Lattice and Its Application to Fuzzy Signature," in *IEICE Trans. Fundamentals* (Japanese Edition), Vol.J98-A, No.6, pp.427-435, 2015.

[7] Kaiyi Zhang, Hongrui Cui and Yu Yu, "Facial Template Protection via Lattice-based Fuzzy Extractors," IACR Cryptology ePrint Arch. 2021: 1559, 2021.

[8] Shuichi Katsumata, Takahiro Matsuda, Wataru Nakamura, Kazuma Ohara and Kenta Takahashi, "Revisiting Fuzzy Signatures: Towards a More Risk-Free Cryptographic Authentication System based on Biometrics," CCS 2021: 2046-2065, 2021.

[9] J. H. Conway and N. J. A. Sloane, "Sphere Packings, Lattices and Groups," second ed., Grundlehren der Mathematischen Wissenschaften Band 290, Springer-Verlag, New York, 1993.

[10] I. Dinur, G. Kindler, R. Raz and S. Safra, "Approximating CVP to Within Almost-Polynomial Factors is NP-Hard," in *Combinatorica,* vol. 23, pp. 205–243, 2003.

[11] ISO/IEC 24745:2022, "Information security, cybersecurity and privacy protection – Biometric information protection," Standard. International Organization for Standardization, Geneva, CH 2022.