Journal of Information Security and Applications 55 (2020) 102609

Contents lists available at ScienceDirect

Journal of Information Security and Applications

journal homepage: www.elsevier.com/locate/jisa

Efficient protocols for private wildcards pattern matching ☆

Tushar Kanti Saha [a,*], Deevashwer Rathee [b], Takeshi Koshiba [c]

[a] Department of Computer Science and Engineering, Jatiya Kabi Kazi Nazrul Islam University, Bangladesh
[b] Department of Computer Science and Engineering, Indian Institute of Technology (BHU), Varanasi, India
[c] Faculty of Education and Integrated Arts and Sciences, Waseda University, Tokyo, Japan

ARTICLE INFO

Keywords:
Privacy-preserving
Repetitive wildcards
Compound wildcards
Secure pattern matching
Somewhat homomorphic encryption
Hamming and Euclidean distances

ABSTRACT

A wildcard character in a pattern adds an additional feature in the field of pattern matching. In this paper, we consider two problems of secure pattern matching (SPM) with wildcards: (i) SPM with repetitive wildcards (SPM-RW) and (ii) SPM with compound wildcards (SPM-CW). Here we consider that a type of wildcard characters "∗" is used to represent gaps in the pattern for the first problem of SPM with wildcards. Usually, a wildcard character "∗" is used to replace with zero or more letters in the text for the pattern matching problem. Yasuda et al. (ACISP 2014) proposed a protocol with an existing data packing method for secure wildcards pattern matching using symmetric somewhat homomorphic encryption (SwHE) in the semi-honest model in which a wildcard character in the pattern is replaced with just one letter in the text. Furthermore, we enhance their work to replace a wildcard with any sequence of letters in the text then propose SPM-RW protocols by using the symmetric and public-key SwHE schemes in the semi-honest model. Also, we propose a packing method that improves the number of homomorphic multiplications by a factor of k compared to a naive usage of Yasuda et al.'s method to solve the SPM-RW problem in which k is the number of sub-patterns. Next, we consider the SPM-CW problem for processing private database queries, which allows a few types of wildcards ("$", "∗", and "!") to appear in the pattern. To solve this problem, we propose an SPM-CW protocol using a *double-query* technique with public-key SwHE encryption in the semi-honest model. Our experiments exhibit the practicality of the new protocols for SPM-RW and SPM-CW, which outperforms state-of-the-art.

1. Introduction

Outsourced computation is now popular due to the availability of cloud services at a low cost. In addition, cloud service providers like Google, Amazon, Microsoft, IBM, and so on have already established themselves as reliable service providers for their users. They are also facilitating massive storage service of data along with the computation using those data. Furthermore, data are gradually increasing due to the extensive use of computers, laptops, smart-phones, and so on by their users. Currently, the users are interested in storing their data online not only for saving space of local computers but also for accessing those data anytime and anywhere from the world. Besides, private computation has vast applications in various fields like biometrics authentication [1], bioinformatics [2,3], data searching [4], private database queries (PDBQ) [5]. Now these computations are outsourced to the cloud securely without revealing any private information to the public. Here, data encryption is one of the techniques to secure the outsourced data. Moreover, the users need to perform computations on the encrypted data without decryption because the decryption within the cloud reveals the data to the cloud. In these respects, securing data and searching and computing on secured data are needed at the same time.

In particular, the secure pattern matching (SPM) within a text is a vital problem that is explored in the earlier works [2–4,6–10]. Since patterns may include errors and gaps in the practical applications, it is important to cope patterns with errors and gaps. We can handle them by inserting wildcard characters ("$", "∗", "!") in the patterns. Here "$" is used in the pattern to replace with a single non-empty letter in the text, which is similar to the method in [6]. A wildcard "∗" replaces with zero or more letters in the text. "!" takes a variable π as an argument, where π specifies any non-empty string of arbitrary length such as "a", "bcc", "adcab", and so on. Now "!(π)" is

☆ This is a full version of the work (Saha and Koshiba, 2017) which was presented at the 9th International Symposium on Foundations & Practice of Security (FPS 2016). This paper extends the contribution in Saha and Koshiba (2017) by adding a protocol using a public-key encryption scheme (Section 4.4.2 and 7.2) with experimental analysis (Section 9.5.1). This paper also adds another problem called SPM with compound wildcards and its application to private database queries (Section 5 and 8) and shows the experimental evaluation as well (Section 9.5.2).

* Corresponding author.
E-mail address: tushar@jkkniu.edu.bd (T.K. Saha).

https://doi.org/10.1016/j.jisa.2020.102609

Available online 18 September 2020

replaced with any non-empty string other than π. Essentially, SPM with wildcards can be used in many fields like database search [5,11,12], bioinformatics [2,3,6,10], and information retrieval [4]. Depending on the appearance of wildcards in the pattern, we can classify pattern into several categories such as patterns with no wildcards (e.g., AGT), patterns with a single wildcard (e.g., AT*), patterns with repetitive wildcards (e.g., AC*CTA*T), and patterns with compound wildcards (e.g., *AAG$CC!(AA)TG*). Similarly, we can also classify SPM into several categories such as SPM with no wildcards, SPM with a single wildcard, SPM with repetitive wildcards (SPM-RW), and SPM with compound wildcards (SPM-CW). Most of the earlier works in secure computation handle the patterns with no wildcards [2,8] or a single wildcard [6,7,9]. Generally speaking, a pattern with a single wildcard may find an erroneous pattern in the text. However, patterns with repetitive wildcards and compound wildcards provide more flexible search to users than the patterns with a single wildcard do. We consider ways to outsource computations of the SPM problems to the cloud.

Here one solution to the above problem is to use homomorphic computation. Homomorphic encryption (HE) has allowed service providers to facilitate computation on encrypted text for their users. At the early stages of the investigation on secure computation, cryptosystems of Goldwasser and Micali [13], El Gamal [14], Cohen and Fischer [15], Paillier [16] allow only single homomorphic computation either addition or multiplication but not both operations. In 2005, Boneh et al. enabled us to perform both operations at the same time [17]. However, they have the limitation of doing many additions but a single multiplication. After that Gentry [18] did the revolutionary work in the field of homomorphic computation, which can freely perform both operations on encrypted data. His method is called fully homomorphic encryption (FHE) that was designed by applying squashing and bootstrapping techniques to somewhat homomorphic encryption (SwHE), which allows us to perform many additions and a few multiplications on encrypted data. The limitation of FHE is that it generates ciphertext of the large size and makes the processing speed slow [19]. After these breakthroughs, Brakerski and Vaikuntanathan [20] proposed more efficient SwHE than Gentry's one which is based on ring learning with errors (ring-LWE) problem supporting many additions and a few multiplications. Actually, they used polynomial ring-based computations in which many plaintexts are packed into a single polynomial of a finite degree, called the packing method that produces a single packed ciphertext after the encryption using the key of SwHE. Therefore, it reduces the ciphertext size and speeds up the computation performances of SwHE. In 2011, Lauter et al. [21] used ring-LWE based SwHE for the efficient computation of statistics (e.g., mean and variance) and discussed some other practical applications including medical, financial, and advertising and pricing. Using the same ring-LWE based SwHE [21], Yasuda et al. [2,6,22] and Saha and Koshiba [23,24] presented several efficient secure applications in the cloud with semi-honest model by introducing new packing methods to speed up the performances of those applications. Thus, we consider to use ring-LWE based SwHE with yet another packing methods for solving the SPM problems in the cloud.

1.1. Recent secure pattern matching techniques

Pattern matching computation can be secured in two ways. One is by application-specific protocols and the other is by general protocols. In this paper, we focus on application-specific protocol. To our knowledge, Troncoso-Pastoriza et al. [3] first addressed the SPM problem using an oblivious automata evaluation for their secure computation. Jha et al. applied Yao's garbled-circuit protocol [25] to secure genomic computation [26]. Here they improved Yao's protocol by dividing problem instances into smaller sub-circuits and sharing the result of evaluating each sub-circuit between the participants. Blanton et al. presented an SPM technique using finite automata for DNA searching [8]. Katz et al. described a new keyword search protocol for private DNA pattern matching by modifying Yao's garbled circuit approach [27].

In the above works, the authors did not mention any pattern which contains a wildcard character (*) to be matched with actual text. So far we have observed that SPM with single-character wildcard was first solved by Baron et al. [7]. They proposed a new protocol for more expressive search queries including a single-character wildcard and substring pattern matching of an arbitrary alphabet. Beck and Kerschbaum [28] proposed a method for the string matching using additively homomorphic encryption scheme with Bloom filters. Thereafter, Defrawy et al. performed a comparative study among some SPM protocols and measured their performances for a wildcard pattern matching [4]. Hazay et al. [9] proposed a few protocols for variations of the SPM problem, which were significantly more efficient than the existing solutions. Their protocols are not suitable for cloud computing because supporting wildcards increases the communication and computational complexity linearly with the increase of size of the text and the pattern. Yasuda et al. [2] proposed a practical solution to an SPM problem for analyzing personal DNA sequence by using ring-LWE based SwHE of [21], which is suitable for the cloud. To perform the SPM with the Hamming distance technique, they proposed a new packing method suitable for an efficient computation of many Hamming distances over the encrypted data. Furthermore, they have a limitation of using the pattern with a single wildcard (*) like "AT*G", "AT*", "*AT" in their paper. To overcome this limitation, Yasuda et al. in their succeeding paper [6] proposed an SPM technique with a wildcard using the same SwHE for searching real-world genome data. To solve the problem of SPM with a wildcard, they used the packing method of [2] to match the pattern including a single wildcard which is replaced with a letter in the text of practical genome data. For example, for the DNA alphabet $\Sigma = \{A, G, C, T\}$, the pattern "AT*" matches any of the texts like "ATC", "ATG", "ATA". Here Yasuda et al. [6] addressed only a letter in the text to be replaced with a wildcard character "*" occurred in the pattern. Usually, "*" is used to replace with zero or more letters in the text for the pattern matching problem. However, none of the methods discussed so far address SPM-RW or SPM-CW problems. Therefore, a new method is indispensable, which allows several letters in the text to be replaced with a wildcard character appeared in the pattern. Very recently, Kim et al. [5] proposed an SPM-CW protocol for the PDBQ problem in which they engaged two-step computation of high-depth for the pattern matching of their protocol. Moreover, they in [5] admit that their performance is far from the practicality. To solve these SPM-RW and SPM-CW problems, new methods are urgently necessary.

1.2. Problem statements

In this paper, we consider two problems of SPM with wildcards: (i) the SPM-RW problem and (ii) the SPM-CW problem.

1.2.1. Secure pattern matching with repetitive wildcards

To define the SPM-RW problem, let us consider a text $T = (a_0 a_1, \ldots, a_{l-1})$ of length l and a pattern $P = (b_{1,0} b_{1,1} \cdots b_{1,p_1-1} * b_{2,0} b_{2,1} \cdots b_{2,p_2-1} * \cdots * b_{k,0} b_{k,1} \cdots b_{k,p_k-1})$ containing k sub-patterns, where the length of each sub-pattern P_y is p_y for $1 \le y \le k$ and $\sum_{y=1}^{k} p_y \le l$. Now securely searching a text T of a cloud database for a pattern P is the first task of our work in which "*" replaces with zero or several letters in the text. In addition, the wildcard appears in the pattern many times; we call it repetitive-wildcards pattern matching. In reality, we have been motivated by the DNA search method used in mtDB[1] in which a wildcard "*" has represented a gap. Furthermore, we consider two scenarios for the security of the SPM-RW problem. The first one is a symmetric scenario which provides security to the user's pattern only, that is similar to the security used in [6]. The second one is an asymmetric scenario which provides security to the text and the pattern and is suitable for many practical applications in which data owner and query owner do not want to reveal their information to each other.

[1] (See http://www.mtdb.igp.uu.se/ for the on-line version of DNA searching.)

1.2.2. Secure pattern matching with compound wildcards

Similar to the recent work in [5], we study another wildcards-based SPM problem called SPM-CW problem and its application to the PDBQ problem. In the SPM-CW problem, a query with compound wildcards (e.g., `*AAG$CC!(AA)TG*`) means that the query string represents a pattern with many wildcards ("$", "*", and "!") rather than repetitive wildcards stated in the first problem. To define this problem, suppose that we need to compare the pattern with compound wildcards $\mathcal{P} = (b_0 \cdots \$b_i!(c_{i+1} \cdots c_{i+\xi})b_{i+\xi+1} \cdots \$b_{\vartheta-1})$ of length ϑ with τ records of the table for any attribute in a database. Moreover, each record of that attribute can be represented by the text $\mathcal{T} = (a_0 a_1 \cdots a_{\varphi-1})$ of length φ. Here the wildcard "$" matches any letter in the text and "!(π)" matches a single non-empty string except π.

1.3. Existing solutions of secure pattern matching with repetitive wildcards problem

Now we describe a solution to the SPM-RW problem using Yasuda et al.'s scheme in [6]. In this paper, we consider the SPM-RW problem for non-binary vectors, which can be realized by measuring the squared Euclidean distance (SED) between sub-text vector T_d and each sub-pattern vector P_y as

$$E_{y,d} = \sum_{h=0}^{p_y-1} (a_{d+h} - b_{y,h})^2$$

for $1 \le y \le k$ and $0 \le d \le (l - p_y)$. Here T_d denotes the $(d+1)$-th sub-vector $(a_{d+0}, \ldots, a_{d+p_y-1})$ of the text T with $|T_d| = p_y$. Moreover, $P_y = (b_{y,0}, b_{y,1}, \ldots, b_{y,p_y-1})$ is the yth sub-pattern vector of the pattern of P where $|P_y| = p_y$ for $1 \le y \le k$ and $\sum_{y=1}^{k} p_y \le l$.

Yasuda et al. [6] showed a solution to the SPM problem with a wildcard using symmetric SwHE based on ring-LWE. They used the SED technique to measure the distance between the text and the pattern of non-binary vectors. Using the technique in [6], we can compute many SEDs using three homomorphic multiplications. If we use the technique in [6] for our SPM-RW problem containing k sub-patterns in the query, each sub-pattern P_y is needed to be matched separately with the text T. Then we need to send k queries separately to the cloud which incurs an extra communication cost. Besides, this technique needs $3k$ homomorphic multiplications to find the SEDs between each sub-pattern P_y and sub-text T_d, which will increase the computation cost to solve the SPM-RW problem (See Section 4.3.1 for the details).

1.4. Existing solution of secure pattern matching with compound wildcards problem

If we want to solve the SPM-CW problem using the technique of Kim et al. [5], we need to use their two-step solution. In the first step, they used the equation to decide the equality between the text $\mathcal{T} = (a_0, a_1, \ldots, a_{\varphi-1})$ and pattern $\mathcal{P} = (w_0, w_1, \ldots, w_{\vartheta-1})$ as follows:

$$z_{\hat{i}} = \prod_{\hat{j}=0}^{\varphi-1} \left(EQ(a_{\hat{j}}, w_{\hat{j}} + \hat{e}_{\hat{j}}) \right)$$

where $0 \le \hat{i} \le \varphi - \vartheta$ and $w_{\hat{j}}$ is any character in the pattern $\mathcal{P}$ with the wildcard indicator flag $\hat{e}_{\hat{j}}$. Moreover, $\hat{e}_{\hat{j}} = 1$ if $w_{\hat{j}}$ is a wildcard; otherwise, $\hat{e}_{\hat{j}} = 0$. From the above equation, it is evident that the multiplicative depth of the above equality computation is $\log \varphi$. In the second step, they used another equation to finalize the pattern matching result with (non)-equalities with the help of a set of constants $\hat{d}_{\hat{i}}$. Here the constants $\hat{d}_{\hat{i}} = 1$ if $0 \le \hat{i} \le \varphi - \vartheta$ and $\hat{d}_{\hat{i}} = 0$ if $\varphi - \vartheta + 1 \le \hat{i} \le \varphi - 1$. The equation used in the second step is expressed as follows:

$$\hat{t} = 1 + \prod_{\hat{i}}^{\varphi-1} (1 + z_{\hat{i}} \cdot \hat{d}_{\hat{i}}).$$

If the pattern $\mathcal{P}$ matches the text $\mathcal{T}$, $\hat{t} = 1$; otherwise, $\hat{t} = 0$. From the above equation, it is obvious that the multiplicative depth of the above computation for pattern matching is $\log \varphi$. Furthermore, the depth increases with the increase of the text length φ. We know that high-depth computation is expensive. Therefore, a new technique is required to improve the cost of the computation for solving the SPM-CW problem.

1.5. Our contributions

Our contributions in this paper are threefold which are stated as follows:

1. For the SPM-RW problem with k sub-patterns, we propose two efficient protocols in the semi-honest model. The first one is using symmetric SwHE based on ring-LWE, called symmetric SPM-RW protocol in which only the pattern is encrypted. The second one is using public-key SwHE based on ring-LWE, called public-key SPM-RW protocol in which the text and the pattern are encrypted. To avoid k queries for k sub-patterns and enable a faster computation of the SPM-RW protocols than [6], we propose a new packing method which allows us to pack all sub-patterns into a single polynomial and send the encrypted polynomial as a single query to the cloud. In addition, our packing method allows us to calculate all the SEDs using 3 homomorphic multiplications instead of $3k$ multiplications required if we follow the technique in [6].
2. For the SPM-CW problem of processing PDBQ, we propose an SPM-CW protocol using a *double-query* technique with the same packing method and public-key SwHE in the semi-honest model. Here we deal with the pattern with compound wildcards, which is absent in [6]. In addition, we consider three types of wildcard symbols "$", "*", and "!" to solve the SPM-CW problem.
3. To evaluate the efficiency of our SPM-RW and SPM-CW protocols regarding two problems, we make experiments of the protocols. The experiments of SPM-RW protocols are performed to show their applications in searching DNA sequences. The experiment of the SPM-CW protocol is performed for processing PDBQ with compound wildcards. For searching DNA sequence, our technique shows more efficiency than the technique of Yasuda et al. [6]. Besides, our symmetric encryption protocol is suitable for evaluating private patterns of any user. In addition, our public-key protocol is suitable for the applications where the text and the pattern are needed to be secured during the pattern matching computation. To process the PDBQ with compound wildcards, our SPM-CW protocol outperforms that in [5]. Moreover, our experiments demonstrate the practicality of our protocols.

1.6. Outline

This paper is organized in the following way. Section 2 gives the preliminaries related to distance measurement techniques for pattern matching. The symmetric and public-key variants of SwHE scheme for our SPM protocols with their security and correctness are described in Section 3. Section 4 mentions about the solving technique of the SPM-RW problem in the cloud along with their protocols. Section 5 highlights the solving technique of the SPM-CW problem along with the protocol. In addition, many inner products computation which helps secure computation of SPM-RW and SPM-CW protocols is explored in Section 6. Section 7 narrates the secure computation of the protocol mentioned in Section 4. Section 8 shows secure computation of the protocol as referred in Section 5. We evaluate the performance of our protocols both theoretically and practically in Section 9. Finally, we conclude our paper with an indication of future works through Section 10.

2. Preliminaries

In this section, we discuss the SPM using the distance measurement techniques. Before the discussion, we present some notations used in the paper.

Notations. $\mathbb{Z}$ denotes the ring of integers. For a prime number q, the ring of integers modulo q is denoted by $\mathbb{Z}_q$. Let n be the lattice dimension of the RLWE-based SwHE. For a vector $A = (a_0, a_1, \ldots, a_{n-1})$, the maximum norm of A is $\max |a_i|$. Let $\langle T, P\rangle$ be the inner product between two vectors T and P. The distribution $D_{\mathbb{Z}^n,\sigma}$ indicates the discrete Gaussian error distribution with an n-dimensional integer vector $\mathbb{Z}^n$ and a standard deviation σ. Moreover, let l be the length of the text T for the SPM-RW problem. Also, let P be a pattern for the SPM-RW problem and P_y be the yth sub-pattern of P with $|P_y| = p_y$. Also, $b_{y,i}$ denotes the ith character of the yth sub-pattern. Let φ (resp., ϑ) be the length of the text $\mathcal{T}$ (resp., pattern $\mathcal{P}$) for the SPM-CW problem. Furthermore, ϖ, Y, and τ denote the number of attributes in a table, a record in the table, and total records in the table respectively. In addition, γ represents the number of texts packed within the lattice dimension n. Finally, $\log(t_{adv})$ defines the logarithmic runtime of an attack that is used to show the security level of a protocol using ring-LWE SwHE scheme and ϵ defines the adversarial advantage.

2.1. Pattern matching using distance measurement

We present how the distance measurement techniques help us in pattern matching. For a given alphabet Σ, the exact pattern matching is the process of deciding whether a pattern $\hat{P} \in \Sigma^{\mu}$ appears in the text $\hat{T} \in \Sigma^{\eta}$ or not, where $\mu \leq \eta$. The matching decision can be made by measuring the distance between the text vector $\hat{T}$ and the pattern vector $\hat{P}$. We can measure the distance for pattern matching using the Hamming distance technique for binary vectors [2] and the SED technique for non-binary vectors [6]. Let us consider a text vector $\hat{T} = (a_0, a_1, \ldots, a_{\eta-1})$ and a pattern vector $\hat{P} = (b_0, b_1, \ldots, b_{\mu-1})$ for the distance measurements that are discussed in the following subsections.

2.1.1. Hamming distance

If $\hat{T}$ and $\hat{P}$ are binary vectors, the Hamming distance H_d between them can be measured by the following equation as

$$H_d = \sum_{h=0}^{\mu-1} |a_{d+h} - b_h| = \sum_{h=0}^{\mu-1} (a_{d+h} + b_h - 2 \cdot a_{d+h} \cdot b_h)$$

for $0 \leq d \leq \eta - \mu$. Here if $H_d = 0$ for some position d then the pattern $\hat{P}$ is found in the text $\hat{T}$ at the index $d + 1$; otherwise, the pattern is not found. For example, we set $\hat{T} = (11101010)$ and $\hat{P} = (1010)$, where $\eta = 8$ and $\mu = 4$. By following the arithmetic computation in the above equation, we obtain that $\eta - \mu + 1 = 5$ Hamming distances as $H_0 = 1$, $H_1 = 3$, $H_2 = 0$, $H_3 = 4$, and $H_4 = 0$. From this result, we can say that there are matches at the 3rd and 5th indices of the text, i.e., the pattern $\hat{P}$ occurs at two places in the text $\hat{T}$.

2.1.2. Squared Euclidean distance

We know that the Euclidean distance technique is used to find the distance between two non-binary vectors, which includes a square root operation. Here we use the SED instead of the Euclidean distance to avoid square the root operation and its cost over the encrypted data. Suppose that $\hat{T}$ and $\hat{P}$ are non-binary vectors, then the SED E_d between them can be measured by the following equation as

$$E_d = \sum_{h=0}^{\mu-1} (a_{d+h} - b_h)^2 = \sum_{h=0}^{\mu-1} (a_{d+h}^2 - 2 \cdot a_{d+h} \cdot b_h + b_h^2)$$

for $0 \leq d \leq \eta - \mu$. If $E_d = 0$ for some position d, we can say that the pattern $\hat{P}$ is found in the text $\hat{T}$ at the index $d+1$; otherwise, the pattern is not found. For instance, consider the DNA alphabet as $\Sigma = \{\mathtt{A}, \mathtt{G}, \mathtt{C}, \mathtt{T}\}$ which can be encoded by the set $\{1, 2, 3, 4\}$. Also, consider that we need to find the pattern $\hat{P} = (\mathtt{ATT}) = (144)$ in the text $\hat{T} = (\mathtt{AGCGATTG}) = (12321442)$, where $\eta = 8$ and $\mu = 3$. Here we can find at most 6 SEDs, where 6 comes from the equations $\eta - \mu + 1 = 8 - 3 + 1 = 6$. Using the arithmetic in the above equation, we compute 6 SEDs as $E_0 = 5$, $E_1 = 6$, $E_2 = 17$, $E_3 = 10$, $E_4 = 0$, and $E_5 = 13$. Among these distances, since $E_4 = 0$, the pattern $\hat{P}$ is found at the index 5 of the text $\hat{T}$.

Remark 1. From the above discussion, it is evident that the distance-based techniques can be used in pattern matching. This paper considers two problems of wildcards pattern matching over the alphabet Σ, which are encoded as non-binary vectors. Therefore, we use the SED technique for the distance measurement between the text and the pattern instead of the Hamming distance technique.

3. Security using homomorphic encryption

We review both symmetric [6] and public-key [2] SwHE schemes based on ring-LWE and their correctness, which are the variants of SwHE in [20] and [21].

3.1. Required parameters

For the SwHE schemes in [2,6], we consider the following parameters.

- n: the lattice dimension such that $n \in \mathbb{Z}$ and $n = 2^{\aleph}$ with $\aleph \in \mathbb{Z}$.
- $f(x)$: a cyclotomic polynomial such that $f(x) = x^n + 1$.
- q: modulus q is an odd prime such that $q \equiv 1 (\mathrm{mod}\ 2n)$.
- R_q: a ring where $R_q = R/qR = \mathbb{Z}_q[x]/f(x)$, denoting a ciphertext space.
- t: an integer $t < q$, defines the message space such that $R_t = \mathbb{Z}_t[x]/f(x)$.
- σ: a parameter which defines a discrete Gaussian error distribution $\chi = D_{\mathbb{Z}^n,\sigma}$ with a standard deviation σ such that $4 \leq \sigma \leq 8$.

3.2. Symmetric somewhat homomorphic encryption

Yasuda et al. [6] showed a symmetric SwHE based on the public-key SwHE scheme in [21]. Now we can discuss the key generation, encryption, homomorphism, and decryption property of this scheme as follows.

Key generation. Generate a ring element $R \ni s \leftarrow \chi$ for the secret key $sk = s$;

Encryption. Firstly, samples $a \leftarrow R_q$ and $e \leftarrow \chi$. For $m \in R_t$ encryption is defined as follows:

$$Enc(m, sk) = (as + te + m, -a) = (c_0, c_1) = ct\,. \tag{1}$$

Homomorphic operations. In our symmetric SPM-RW protocol (See Section 4.4.1 for the details), one is plaintext m' and another is ciphertext ct. Now the homomorphic operation between $ct = (c_0, c_1)$ and $m' \in R_q$ can be defined as

$$\begin{cases} \text{Hom. Add.} : ct_{add} = ct \boxplus m' = (c_0 \boxplus m', c_1) \\ \text{Hom. Mul.} : ct_{mul} = ct \boxtimes m' = (c_0 \boxtimes m', c_1 \boxtimes m')\,. \end{cases} \tag{2}$$

Decryption. For $ct \in (R_q)^2$, $t \in R_t$, and the secret key sk, a general decryption can be defined as

$$Dec(ct, sk) = [\tilde{m}]_q \bmod t\,, \tag{3}$$

where $\tilde{m} = \sum_{i=0}^{\alpha} c_i s^i$. For the secret key vector $\mathbf{s} = (1, s, s^2, \ldots, s^{\alpha})$, we can simply rewrite $\mathrm{Dec}(ct, sk) = [\langle ct, \mathbf{s}\rangle]_q \bmod t$. For example, a fresh ciphertext $ct = (c_0, c_1)$ generated by Eq. (1) can be decrypted as follows:

$$\langle ct, \mathbf{s}\rangle = c_0 + c_1 \cdot s = (as + te + m) + (-a) \cdot s = m + t \cdot e\,. \tag{4}$$

From the above equation, we can recover text m by mod t operation. In the same way, homomorphic decryption can be defined by the following ways:

$$\begin{cases} Dec(ct_{add}, sk) = [\tilde{m}_{add}]_q \bmod t, \text{ where } \tilde{m}_{add} = c_0 + m' + c_1 s = m + m' \\ Dec(ct_{mul}, sk) = [\tilde{m}_{mul}]_q \bmod t, \text{where } \tilde{m}_{mul} = c_0 m' + c_1 m' s = m \cdot m'. \end{cases}$$

3.3. Correctness of symmetric scheme

The correctness of symmetric SwHE scheme depends on how the decryption can recover the original result from the ciphertext after some homomorphic operations. We can write the decryption process as follows:

$$\begin{cases} Dec(ct_{add}, sk) = Dec((ct \boxplus m'), sk) = m + m' \\ Dec(ct_{mul}, sk) = Dec((ct \boxtimes m'), sk) = m \cdot m'. \end{cases} \tag{5}$$

Actually, the above process is already described in Section 3.1 of [20]. Here ciphertext ct comes from $m \in R_q$ after encryption and another plaintext $m' \in R_q$. The correctness of SwHE scheme in Section 3.2 holds if it satisfies the following lemma as shown in [6].

Lemma 1 (*Condition for Successful Decryption*). *For a ciphertext ct, the decryption $Dec(ct, sk)$ recovers the correct result if $\langle ct, \mathbf{s}\rangle \in R_q$ does not wrap-around mod q, namely, if the condition $\|\langle ct, \mathbf{s}\rangle\|_\infty < \frac{q}{2}$ is satisfied, where $\|a\|_\infty = \max |a_i|$ for an element $a = \sum_{i=0}^{n-1} a_i x^i \in R_q$. Specifically, for a fresh ciphertext ct, the ∞-norm $\|\langle ct, \mathbf{s}\rangle\|_\infty$ is given by $\|m+te\|_\infty$. Moreover, for a homomorphically operated ciphertext, the ∞-norm can be computed by Eq. (2).*

3.4. Public-key somewhat homomorphic encryption

In 2013, Yasuda et al. [2] showed a SwHE scheme which is a public-key variant of BV's SwHE scheme [20] and SwHE by Lauter et al. [21]. Now we review the algorithms of SwHE scheme in [21] and [2] as follows.

SwHE.KeyGen. Generate secret key $sk = s$ such that $R \ni s \xleftarrow{\$} \chi$. Sample a uniformly random element $a_1 \in R_q$ and an error $R \ni e \xleftarrow{\$} \chi$. Now we get the public-key pair as $pk = (a_0, a_1)$ with $a_0 = a_1 s + te$.

SwHE.Enc. First samples $R \ni u, f, g \xleftarrow{\$} \chi$. For a given message $m \in R_t$ and $pk = (a_0, a_1)$, the encryption can be defined as follows:

$$\text{Enc}(m, pk) = (a_0 u + tg + m, -(a_1 u + tf)) = (c_0, c_1) = ct. \tag{6}$$

At this place, the plaintext $m \in R_t$ is also in R_q because $t < q$.

Homomorphic operations. Generally, homomorphic operations between two ciphertexts $ct = (c_0, \ldots, c_\alpha)$ and $ct' = (c'_0, \ldots, c'_\beta)$. can be defined as follows:

$$\begin{cases} \text{Hom. Add.} : ct_{add} = ct \boxplus ct' = \left(c_0 + c', \ldots, c_{max(\alpha,\beta)} + c'_{max(\alpha,\beta)}\right) \\ \text{Hom. Mul.} : ct_{mul} = ct \boxtimes ct' = \sum_{i=0}^{\alpha+\beta} \hat{c}_i z^i = \left(\sum_{i=0}^{\alpha} c_i z^i\right)\left(\sum_{j=0}^{\beta} c'_j z^j\right). \end{cases} \tag{7}$$

Here z denotes a symbolic variable such that $\{ct, ct'\} \in R_q[z]$. In addition, we can define the subtraction as similar to component-wise addition.

SwHE.Dec. For a fresh or homomorphically operated ciphertext $ct = (c_0, \ldots, c_\alpha)$ and $t \in R_t$, a general decryption can be defined as

$$\text{Dec}(ct, sk) = [\tilde{m}]_q \bmod t, \tag{8}$$

where $\tilde{m} = \sum_{i=0}^{\alpha} c_i s^i$. For the secret key vector $\mathbf{s} = (1, s, s^2, \ldots)$, we can simply rewrite $\text{Dec}(ct, sk) = [\langle ct, \mathbf{s}\rangle]_q$ mod t. For instance, a fresh ciphertext $ct = (c_0, c_1)$ generated by Eq. (6) can be decrypted as

$$\langle ct, \mathbf{s}\rangle = (a_0 u + tg + m) - s \cdot (a_1 u + tf) = m + t \cdot (ue + g - sf), \tag{9}$$

where $a_0 - a_1 s = te$. If the value $m + t \cdot (ue + g - sf)$ does not wrap-around mod q (all errors $e, f, g, u \xleftarrow{\$} \chi$ must be sufficiently small), we have $[\langle ct, \mathbf{s}\rangle]_q = m + t \cdot (ue + g - sf) \in R$ and can recover plaintext m by mod t operation. If no wrap-around happens in the encrypted results after homomorphic operations for two ciphertexts ct and ct', we clearly have the relations as follows:

$$\begin{cases} \langle ct \boxplus ct', \mathbf{s}\rangle = \langle ct, \mathbf{s}\rangle + \langle ct', \mathbf{s}\rangle = m + m' \\ \langle ct \boxtimes ct', \mathbf{s}\rangle = \langle ct, \mathbf{s}\rangle \cdot \langle ct', \mathbf{s}\rangle = m \cdot m'. \end{cases} \tag{10}$$

3.5. Correctness of public-key scheme

The correctness of this public-key SwHE scheme also depends on the recovery of the original result from the ciphertext after some homomorphic operations. Now we can show the decryption procedure as follows:

$$\begin{cases} \text{Dec}(ct_{add}, sk) = \text{Dec}((ct \boxplus ct'), sk) = m + m' \\ \text{Dec}(ct_{mul}, sk) = \text{Dec}((ct \boxtimes ct'), sk) = m \cdot m'. \end{cases} \tag{11}$$

Brakerski and Vaikuntanathan discussed the above procedure in Section 3.2 in [20]. Here ciphertexts ct and ct' come from $m \in R_q$ and $m' \in R_q$ respectively after the encryption. The SwHE scheme in Section 3.4 holds if it satisfies the following lemma as shown in [2].

Lemma 2 (*Condition for Successful Decryption*). *For a ciphertext ct, the decryption $Dec(ct, sk)$ recovers the correct result if $\langle ct, \mathbf{s}\rangle \in R_q$ does not wrap around mod q, namely, if the condition $\|\langle ct, \mathbf{s}\rangle\|_\infty < \frac{q}{2}$ is satisfied, where $\|a\|_\infty = \max |a_i|$ for an element $a = \sum_{i=0}^{n-1} a_i x^i \in R_q$. Specifically, for a fresh ciphertext ct, the ∞-norm $\|\langle ct, \mathbf{s}\rangle\|_\infty$ is given by $\|m+t(ue+g-sf)\|_\infty$. Moreover, for a homomorphically operated ciphertext, the ∞-norm can be computed by Eq. (10).*

3.6. Security of the schemes

We can show the security of the mentioned SwHE schemes by the polynomial ring-LWE assumption (ring-LWE$_{n,q,\chi}$) as mentioned in [21] for the given parameters (n, q, t, σ). The assumption holds for any polynomial number of samples of the form as $(a_i, b_i = a_i \cdot s + e_i) \in (R_q)^2$ where $a_i \xleftarrow{\$} R_q$ and $e \xleftarrow{\$} \chi$. Here the a_i's are uniformly random in R_q and b_i's ($b_i = a_i \cdot s + e_i$) are also uniform in R_q. Therefore, it is hard to distinguish (a_i, b_i) from a uniformly random pair $(a_i, b_i) \in (R_q)^2$. Besides, Lyubashevsky et al. [29] showed that the ring-LWE assumption is reducible to the worst-case hardness of problems on ideal lattices, that is believed to be secure against the attacks by a quantum computer.

4. Technique of secure pattern matching with repetitive wildcards in the cloud

For the SPM-RW problem, let us consider the text vector $T = (a_0, a_1, \ldots, a_{l-1})$ of length l outsourced to the cloud. Also, consider the pattern $P = (b_{1,0} b_{1,1} \cdots b_{1,p_1-1} * b_{2,0} b_{2,1} \cdots b_{2,p_2-1} * \cdots * b_{k,0} b_{k,1} \cdots b_{k,p_k-1})$ containing k sub-patterns in which each sub-pattern can be represented by a vector $P_y = (b_{y,0}, b_{y,1}, \ldots, b_{y,p_y-1})$ with $|P_y| = p_y$. For this pattern matching problem, we need to search the text T for k sub-patterns by maintaining the sequences of all sub-patterns as they appear in the main pattern. Here we need to calculate every distance between each sub-pattern P_y and each sub-text T_d of length p_y for $1 \le y \le k$ and $0 \le d \le (l - p_y)$. Besides, T_d denotes the $(d+1)$-th sub-vector $(a_{d+0}, \ldots, a_{d+p_y-1})$ of the text T with $|T_d| = p_y$. Now we use the SED-based pattern matching technique to solve the SPM-RW problem. For $1 \le y \le k$ and $0 \le d \le (l - p_y)$, the SED $E_{y,d}$ between the sub-text T_d and each sub-pattern P_y is calculated by the following equation:

$$E_{y,d} = \sum_{h=0}^{p_y-1} (a_{d+h} - b_{y,h})^2 = \sum_{h=0}^{p_y-1} (a_{d+h}^2 - 2 \cdot a_{d+h} \cdot b_{y,h} + b_{y,h}^2). \tag{12}$$

In the case of pattern matching with P_y, if $E_{y,d} = 0$ for some d, the pattern P_y is found at the index $d+1$ of the text T. For a faster computation, we can use the inner product property in [30] for computing $E_{y,d}$ in Eq. (12) using some packing methods.

Algorithm 1: Sub-patterns' order preserving

Data: All SEDs and their indices, k.
Result: Pattern matches, or no match found.

```
ψ := −1 ;                                   // set the pointer
y := 1 ;                                    // Initialization
for y ≤ k do
    d := 0; θ := 0; ρ[y] := 0;
    for d ≤ l − p_y do
        if E_{y,d} = 0 ς d > ψ then
            ρ[y] := d + 1 ;      // P_y found at the position d + 1 in T
            ψ := d + p_y − 1 ;   // Move forward the pointer p_y − 1 positions
            θ := 1;
            return;
        else
            Return to the loop;
    if θ = 1 then
        y := y + 1 ;                        // Increment y
    else
        Print("No match found"); return;
Print("Pattern matches"); return;
```

4.1. Order preserving of the sub-patterns

In this subsection, we discuss how the sequences of k sub-patterns $\{P_1, P_2, \ldots, P_k\}$ are maintained after computing the SEDs between the sub-patterns P_y and the sub-texts of T. Here, we use "match and move (MaM)" technique to preserve the sub-patterns' order during matching all sub-patterns with the text T. After computing the SEDs, the algorithm used for pattern matching by preserving sub-patterns' sequences with the MaM technique is shown in Algorithm 1. In this algorithm, a pointer ψ is maintained to keep track of the index in the text where the current pattern ends. Furthermore, the algorithm maintains another array variable ρ to keep the indices of each sub-pattern's match. For the SPM-RW problem, the sub-pattern ordering will be done according to their appearance in the main pattern P. For $1 \le y \le k$, Algorithm 1 stores the non-zero index in $\rho[y]$ in the order $P_1, P_2, \ldots, P_k$; otherwise, declare no match found. Before processing SEDs for any sub-pattern P_y, ψ is set to -1. Then the algorithm process k sub-patterns one by one. First, Algorithm 1 finds whether $E_{y,d} = 0$ or $E_{y,d} \neq 0$ for each sub-pattern P_y at the position $d+1$ in the text T. If $E_{y,d} = 0$ and $d > \psi$, P_y is found in the text at the position $d+1$. Then the pointer ψ is set to the current position d and moved forward $P_y - 1$ positions in the text T. Moreover, the next sub-pattern P_{y+1} matching, if $E_{y+1,d} = 0$ for some position d, our algorithm choose d if $d > \psi$. If $\rho[y] = 0$ for some y, the algorithm prints no match found and terminates; otherwise, pattern matches. In this way, sub-patterns' sequences are maintained by Algorithm 1.

Before discussing a solution to the SPM-RW problem, we review some of the existing packing methods and describe our packing method that will be used in our protocols.

4.2. Packing methods

The encoding of many bits into a single polynomial is called packing method. Early researchers [2,21,22,31] used the packing methods to make the secure computation faster using HE. In this subsection, we review some of the existing packing methods and describe our packing method for the SPM-RW problem.

4.2.1. Chinese remainder theorem packing

In 2011, Smart and Vercauteren [31] presented the CRT packing, also called polynomial-CRT based on the Chinese remainder theorem (CRT), which helps to implement component-wise homomorphic operations especially addition and multiplication. Suppose that $\mathbb{Z}$ is a ring of integers and $x^n + 1$ is an irreducible polynomial over $\mathbb{Z}$ in which n defines the ring dimension. The component-wise operations are possible because the polynomial x^n+1 can be factored completely (into linear factors) over $\mathbb{Z}_{\mathbf{t}}$, where $\mathbf{t}$ is a prime that satisfies the condition $\mathbf{t} \equiv 1 \bmod 2n$. The polynomial $x^n + 1$ factorizes as $x^n + 1 = \Pi_{i=1}^{n}(x - a_i) \bmod \mathbf{t}$, where $a \in \mathbb{Z}_{\mathbf{t}}$ is an element of order $2n$ and $a_i \equiv a^{2i-1} \bmod \mathbf{t}$. In addition, for each linear factor $(x-a_i)$ of x^n+1, we have a quotient ring $R_{\mathbf{t}_i} = \mathbb{Z}_{\mathbf{t}}[x]/\langle x - a_i \rangle \cong \mathbb{F}_{\mathbf{t}}$ of order $\mathbf{t}$. Since the $(x - a_i)$'s are co-prime, the following isomorphism holds from CRT as follows:

$$R_{\mathbf{t}} \cong R_{\mathbf{t}_1} \times R_{\mathbf{t}_2} \times \cdots \times R_{\mathbf{t}_n} \cong \mathbb{F}_{\mathbf{t}} \times \mathbb{F}_{\mathbf{t}} \times \cdots \times \mathbb{F}_{\mathbf{t}} \,. \tag{13}$$

Now we can pack all the messages independently in these smaller rings, only to apply CRT to construct a polynomial $a(x) \in R_{\mathbf{t}}$, which will be finally encrypted. During the homomorphic operations, the homomorphism between R_q and $R_{\mathbf{t}}$ applies to each smaller ring $R_{\mathbf{t}_i}$ independently due to co-primality as well as isomorphism. Moreover, let $\phi : \mathbb{Z}_{\mathbf{t}}^n \to R_{\mathbf{t}}$ be the isomorphic mapping which maps an integer vector $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{Z}_{\mathbf{t}}^n$ to a polynomial in $R_{\mathbf{t}}$. The mapping is defined as

$$\phi(\mathbf{x}) = \sum_{i=1}^{n} x_i \cdot A_i(x) \cdot B_i \bmod \mathbf{t} \,, \tag{14}$$

where $A_i(x) = (x^n + 1)/(x - a_i)$ and $B_i = \big(A_i(x)\big)^{-1} \bmod (x - a_i)$ given by the CRT. Moreover, the inverse mapping $\phi^{-1} : R_{\mathbf{t}} \to \mathbb{Z}_{\mathbf{t}}^n$ is defined as follows:

$$\phi^{-1}(a) = \big(a(x) \bmod (x - a_1), \ldots, a(x) \bmod (x - a_n)\big) \,.$$

From the above equation, for an arbitrary polynomial $P(x)$, the evaluation of the polynomial $P(a_i)$ is the same as doing $P(x) \bmod (x - a_i)$. If we have two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{Z}_{\mathbf{t}}^n$, the following equations hold:

$$\begin{aligned} \phi^{-1}\big(\phi(\mathbf{x}) \times \phi(\mathbf{y})\big) &= (x_1 \cdot y_1, x_2 \cdot y_2, \ldots, x_n \cdot y_n) \bmod \mathbf{t} \\ \phi^{-1}\big(\phi(\mathbf{x}) + \phi(\mathbf{y})\big) &= (x_1 + y_1, x_2 + y_2, \ldots, x_n + y_n) \bmod \mathbf{t} \,. \end{aligned}$$

In this way, this packing method can help component-wise operations. But this packing method requires additional procedure to obtain inner product [6] for Hamming distance computation.

4.2.2. Packing method of Lauter et al.

For encoding a long integer like 128 bits, a packing method was proposed by Lauter et al. [21]. They have packed an integer of n bits into a single polynomial of degree $n-1$. They encoded an n-bit integer into an n-bit binary vector $A = (a_0, a_1, \ldots, a_{n-1})$ and used a univariate polynomial to show the packing as follows:

$$Poly(A) = \sum_{i=0}^{n-1} a_i x^i \,.$$

Limitations: By using SwHE schemes in Section 3, $Poly(A)$ can be encrypted as $ct(A) = Enc(Poly(A), \mathbf{K})$, where $\mathbf{K}$ represents either a symmetric key or a public key. Besides, the homomorphic operations may require several additions and multiplications. This packing method has no problem for adding vectors like $A = (a_0, a_1, \ldots, a_{n-1})$ and $B = (b_0, b_1, \ldots, b_{n-1})$. On the contrary, the multiplication of $Poly(A)$ and $Poly(B)$ will increase the degree of resultant polynomial greater than $n-1$. If the highest degree of the polynomial is considered at most $n-1$, this packing method can perform only d multiplications if the polynomial is reduced to a polynomial of degree $n/d - 1$ to pack n/d bits for $2 \le d \le n$.

4.2.3. Packing method of Yasuda et al.

To overcome the problem of Lauter et al.'s packing method mentioned above and support many multiplications within the degree $n-1$, Yasuda et al. [2] proposed an extension of the packing method in [21]. Their packing method also facilitated secure inner product of two vectors $A = (a_0, a_1, \ldots, a_{n-1})$ and $B = (b_0, b_1, \ldots, b_{n-1})$. They proposed another packing method to serve a wide range of computations by applying the inner products to the packed ciphertexts. Here the packing method for the integral vectors A and B of length n are represented by the following two univariate polynomials as follows:

$$\left.\begin{aligned} PK_1(A) &= \textstyle\sum_{i=0}^{n-1} a_i x^i \\ PK_2(B) &= -\textstyle\sum_{i=0}^{n-1} b_i x^{n-i} = a_0 - \textstyle\sum_{i=1}^{n-1} b_i x^{n-i}. \end{aligned}\right\} \quad (15)$$

Here the first polynomial $PK_1(A)$ is similar to packing method in [21], but the main modification is performed in the second polynomial $PK_2(B)$ in which Yasuda et al. [2] choose the degree of x in the decreasing order for each coefficient b_i. They used the trick to get inner product $\langle A, B\rangle = a_0 \cdot b_0 + a_1 \cdot b_1 + \cdots + a_{n-1} \cdot b_{n-1} = \sum_{i=0}^{n-1} a_i b_i$ of vector A and B as a coefficient of x^n from the polynomial which is formed by multiplying above polynomials as $PK_1(A) \times PK_2(B)$. As mentioned by Yasuda et al. in [30], this inner product helps the Hamming distance and the Euclidean distance calculations.

Limitations: Yasuda et al. [6] used the above packing method with inner product technique for their secure wildcards pattern matching using symmetric SwHE. In their protocol, they also require three homomorphic multiplications for a pattern like "AT*" to match some DNA sequences "ATA", "ATT", "ATG", and "ATC" over the DNA alphabet. In their paper, they used a wildcard character "*" in the pattern to replace with a single letter in the text. In this paper, we consider the problem in which the pattern "GTGCT*CC*GT*T" including repetitive wildcards can be matched with a DNA sequence "AAGTGCTGCCAGTCGT" as shown in Fig. 1(a). Furthermore, there exist some gaps in the pattern represented by the wildcard character "*" which can be used to replace with zero or several letters in the text. If we split the pattern into sub-patterns excluding the wildcards, we get 4 different sub-patterns as "GTGCT", "CC", "GT", and "T;;. For matching these sub-patterns with the given DNA sequence in the cloud, Alice needs to send 4 queries if we apply the SED technique to our SPM using the technique of Yasuda et al. [6] with the packing method in [2]. Here 4 queries need 4 separate matches in the text as shown in Fig. 1(b). As mentioned in Section 2.1.2, the SED calculation for each query requires 3 multiplications. Then 4 queries require 12 multiplications if we use Yasuda et al.'s technique (See Section 4.3.1 for the details). If we could accomplish the computations by sending one query by Alice as shown in Fig. 1(c) using some extended packing method, we can reduce the multiplication cost along with communication cost, which will reduce the overall computation cost of the pattern matching.

4.2.4. Our packing method

To overcome the above limitation, we define another packing method than that in [2]. Consider the same pattern P and text vector $T \in \mathbb{Z}^l$ with $|T| = l$, where $l \le n$. Here n defines the lattice dimension of the ring-LWE based SwHE scheme used for homomorphic computation as mentioned in Section 3. For any computation using ring-LWE based SwHE, the degree of the polynomial should be $n-1$, which includes n elements as the coefficients of that polynomial. Since each letter of the text will be encoded into the coefficient of the polynomial, the length of the text should be less than or equal to n. Furthermore, the pattern P with k sub-patterns can be represented as an integer vector $\bar{P} = (P_1, P_2, \ldots, P_k)$ by omitting the wildcards, where $P_y = (b_{y,0}, b_{y,1}, \ldots, b_{y,p_y-1})$, $|P_y| = p_y$, and $\sum_{y=1}^{k} p_y \le l$. To solve the SPM-RW problem, we can perform the matching by measuring the SEDs between the text and each sub-pattern of the same length as mentioned in Section 2.1.2. Now we need to measure many SEDs of every sub-pattern P_y from every sub-text T_d of the text T with $|P_y| = |T_d| = p_y$. Moreover, T_d is the $d+1$-th sub-vector $(a_d, a_{d+1}, \ldots, a_{d+p_y-1})$ of T of length p_y with $0 \le d \le (l - p_y)$. In addition, P_y is the y-th sub-pattern vector of P for $1 \le y \le k$.

Here we can find the SEDs between sub-text T_d and each sub-pattern P_y by the inner product property in [30] (See Section 4.3.2 for the details). Then we can place those distances as a coefficient of different degrees of x of a polynomial with the degree $n-1$. Therefore, if we use the packing method in [2] which packs all the sub-patterns into a single polynomial (See $PK_2(B)$ in the previous sub-section), *the pattern matching results of most sub-patterns will be wrap-around the coefficient with some degrees of* x. Then it is difficult to extract each sub-pattern matching result separately from the resultant polynomial. As a result, it is necessary to get each sub-pattern matching result as a coefficient of different degrees of x. Now we need to pack the pattern $\bar{P}$ in another way than the packing in [2]. *To overcome the above problem that is, to avoid this wrap-around of coefficient for any degree of* x, *we take the highest degrees of* x *as* ly *for the first element* $b_{y,0}$ *in* P_y *and decrease those degrees for other elements* $b_{y,v}$ *in that* P_y *with* $1 \le v \le p_y - 1$ *and* $1 \le y \le k$. Therefore, using the packing method of [21] and modifying packing method of [2], our packing method can be represented by the following two polynomials in the ring $R = \mathbb{Z}[x]/(x^n + 1)$ with $n \ge (k+1)l$.

$$\left.\begin{aligned} Poly_1(T) &= \textstyle\sum_{j=0}^{l-1} a_j x^j \\ Poly_2(\bar{P}) &= \textstyle\sum_{y=1}^{k} \sum_{i=0}^{p_y-1} b_{y,i} x^{ly-i}. \end{aligned}\right\} \quad (16)$$

Here the packing method $Poly_1(T)$ is similar to the packing in [21]. The main modification is done in the second packing $Poly_2(\bar{P})$ to pack k sub-patterns in a single polynomial.

Remark 2. Here we see that the degree of the polynomial $Poly_2(\bar{P})$ increases with the increase of text length l. Furthermore, we know that multiplication of higher degree polynomials is costlier than that of lower degree polynomials. Consequently, we keep the value of l as small as possible to reduce the degree of the polynomial and obtain a better result than existing methods (See Section 9.4 for the details). Besides, we hope that this limitation will be overcome using the upcoming high configurable machines.

4.3. Solving techniques of secure pattern matching with repetitive wildcards

We discuss two techniques to solve the SPM-RW problem as follows.

4.3.1. Yasuda et al.'s technique

Now we present the solving technique of the SPM-RW problem using the method in [6]. If we perform the SPM computation of k sub-patterns separately, the arithmetic computation in Eq. (12) can be calculated by engaging the inner product property in [30] as follows:

$$E_{y,d} = \langle T_d^2, V_{p_y}\rangle + \langle P_y^2, V_l\rangle - 2 \cdot \langle T_d, P_y\rangle. \quad (17)$$

Here V_l (resp., V_{p_y}) denotes a vector like (1,1,…,1) of length l (resp., p_y). If $E_{y,d} = 0$ for some position d in the text T, we can say that the sub-pattern P_y is found at the index $d+1$ of the text T; otherwise, the pattern is not found in the text. For instance, consider the DNA alphabet as $\Sigma = \{A, G, C, T\}$ encoded as $\{1, 2, 3, 4\}$. For the SPM-RW problem, suppose that the text T = (AGCGATTGC) = (123214423) with $l = 9$. Also, suppose that the pattern P = (GC ∗ TTG) = (23 ∗ 442) with 2 sub-patterns P_1 = (GC) = (23) and P_2 = (TTG) = (442). Here the lengths $p_1 = 2$ and $p_2 = 3$. Now we can get at most $l - p_y + 1$ SEDs for each sub-pattern P_y. First we need decide whether the pattern P_1 is occurred in the text T. We compute $E_{1,0}$ by measuring the distance between P_1 = (23) and sub-text T_0 = (12) using the arithmetic in Eq. (17) as

$$E_{1,0} = \langle (1,4), (1,1)\rangle + \langle (4,9), (1,1)\rangle - 2 \cdot \langle (1,2), (2,3)\rangle = 5 + 13 - 2 \cdot 8 = 2.$$

From the above calculation of $E_{1,0}$, it is evident that we can get the SED through the inner product property. In the similar fashion, we

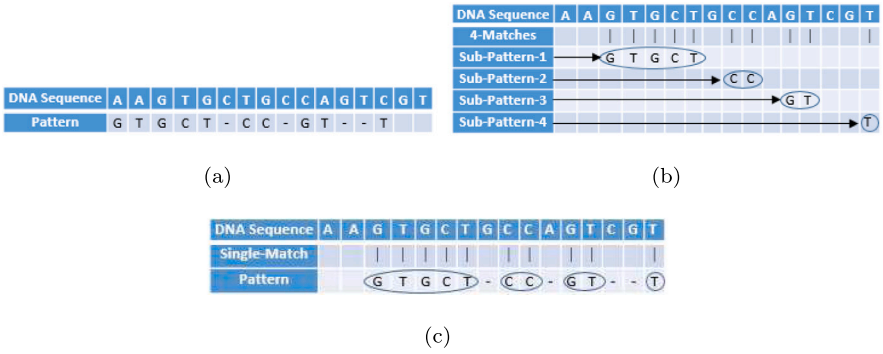

Fig. 1. (a) Our problem domain; (b) Pattern matching for a non-binary vector of text and pattern using our sub-pattern matching concept; (c) Our pattern matching technique.

can compute rest of the SEDs for sub-pattern P_1 as $E_{1,1} = 0$, $E_{1,2} = 2$, $E_{1,3} = 4$, $E_{1,4} = 2$, $E_{1,5} = 5$, $E_{1,6} = 5$, and $E_{1,7} = 0$. Among these distances, since $E_{1,1}$ and $E_{1,7}$ are "0", we can say that the pattern P_1 is found at the indices 2 and 8 of the text T. By following the same approaches like sub-pattern P_1, it is easy to see that the sub-pattern P_2 is found at the index 6 since $E_{2,5} = 0$.

From the above discussion and example, it is obvious that the Eq. (17) represents many SEDs required to compute for our pattern matching. If we follow naive approach to compute the SEDs, we need to use $l-p_y+1$ times Eq. (17) for each sub-pattern P_y to find the pattern matching distances from text T. Furthermore, Yasuda et al. [6] used an efficient technique to compute many SEDs using many inner products for their SPM with a wildcard. They also showed a polynomial-based computation technique in which each of the SEDs can be found separately as a coefficient of different degrees of x of a polynomial with the degree $n-1$. Now we use their approach to our SPM-RW problem. For a vector $A = (a_0, \ldots, a_{l-1})$ in the same ring R with lattice dimension n, they define the packing method as mentioned in Eq. (15). In addition, we use this packing method to get the many inner products from our vectors T and P_y with $p_y \leq l \leq n$. Since $x^n = -1$ in the ring R, the polynomial multiplication of $PK_1(T)$ and $PK_2(P_y)$ is obtained as

$$\left(\sum_{i=0}^{l-1} a_i x^i\right) \times \left(-\sum_{h=0}^{p_y-1} b_{y,h} x^{n-h}\right) = -\sum_{i=0}^{l-1}\sum_{h=0}^{p_y-1} a_i b_{y,h} x^{n+i-h}$$
$$= \sum_{d=0}^{l-p_y}\sum_{h=0}^{p_y-1} a_{d+h} b_{y,h} x^d + \text{sub-poly of deg.} > l - p_y$$
$$= \sum_{d=0}^{l-p_y} \langle T_d, P_y\rangle x^d + \text{sub-poly of deg.} > l - p_y \quad (18)$$

for $1 \leq y \leq k$. As mentioned in [30], the multiple inner products in the above equation can be used for computing many SEDs in Eq. (17). Now we use the polynomials in Eq. (15) and the inner product in Eq. (18) for the computation of the SEDs between text T and sub-pattern P_y as follows:

$$E_y = PK_1(T^2)\cdot PK_2(V_{p_y}) + PK_2(P_y^2)\cdot PK_1(V_l) - 2\cdot PK_1(T)\cdot PK_2(P_y). \quad (19)$$

Here the above polynomial contains all the SEDs between sub-text T_d and sub-pattern P_y as the coefficients of x^d for $0 \leq d \leq l - p_y$. To serve the purpose of secure computation in the symmetric case, the above computation is performed between the encrypted pattern and plaintext in the cloud. Then client needs to send the encrypted k sub-patterns in the form of $ct_2(P_y)$ and $ct_2(P_y^2)$ in the cloud using SwHE using the secret key. Now the secure computation can be realized by the following equation.

$$ct(E_y) = PK_1(T^2)\cdot PK_2(V_{p_y}) + ct_2(P_y^2)\cdot PK_1(V_l) - 2\cdot PK_1(T)\cdot ct_2(P_y). \quad (20)$$

Moreover, for the asymmetric case, the computation mentioned above is performed between the encrypted pattern and ciphertext in the cloud. Thereafter, one client needs to send the encrypted text in the form of $ct_1(T)$ and $ct_1(T^2)$ using the public key of SwHE. Another client needs to send k sub-patterns in the form of $ct_2(P_y)$ and $ct_2(P_y^2)$ in the cloud using the same public key. Then the secure computation can be realized by the following equation.

$$ct(E_y) = ct_1(T^2)\cdot ct_2(V_{p_y}) + ct_2(P_y^2)\cdot ct_1(V_l) - 2\cdot ct_1(T)\cdot ct_2(P_y). \quad (21)$$

Now we use all these secure SEDs to finalize pattern matching by preserving the sub-pattern sequences with the help of Algorithm 1.

Limitations: If we use Yasuda et al.'s technique [6] to solve the SPM-RW problem, we need to send k queries and requires $3k$ homomorphic multiplications in the server, which is time-consuming along with the communication cost (See the results in Sections 9.2 and 9.5.1). Can we solve the SPM-RW problem with a few multiplications rather than $3k$ multiplications to reduce the computation cost?

4.3.2. Our technique

To achieve more efficiency, we need to compute many SEDs of Eq. (12) in a fewer multiplications than the technique in [6]. As mentioned in Section 4.3.1, Yasuda et al.'s technique allowed processing k sub-patterns by sending k queries to the cloud separately. Here we can achieve more efficiency if we are able to pack all the sub-patterns in one polynomial and send it as a single query to the cloud. Now we engage the polynomials in Eq. (16) which enable us to pack all sub-patterns into a single polynomial. Moreover, we can get the SEDs between each sub-pattern P_y and sub-text T_d as the coefficients of a polynomial for different degrees of x with the inner product property as follows.

Inner product property. As discussed in Section 3.2 of [30], the inner product of two vectors helps to compute the Hamming distance and Euclidean distance. To find the inner product property using our packing method in Eq. (16), the polynomial multiplication of $Poly_1(T)$ and $Poly_2(\bar{P})$ can be shown as follows:

$$Poly_1(T) \times Poly_2(\bar{P}) = \left(\sum_{j=0}^{l-1} a_j x^j\right) \times \left(\sum_{y=1}^{k}\sum_{i=0}^{p_y-1} b_{y,i} x^{ly-i}\right)$$
$$= \sum_{y=1}^{k}\sum_{j=0}^{l-1}\sum_{i=0}^{p_y-1} a_j b_{y,i} x^{j+ly-i}$$
$$= \sum_{y=1}^{k}\sum_{d=0}^{l-p_y}\sum_{h=0}^{p_y-1} a_{d+h} b_{y,h} x^{ly+d} + \text{ToHD} + \text{ToLD}$$
$$= \sum_{y=1}^{k}\sum_{d=0}^{l-p_y} \langle T_d, P_y\rangle x^{ly+d} + \text{ToHD} + \text{ToLD}. \quad (22)$$

The above equation shows that one polynomial multiplication includes many inner products $\langle T_d, P_y\rangle$. In addition, ToHD (terms of higher degree) defines a sub-polynomial such that $deg(x) > (ly + l - p_y)$ and ToLD (terms of lower degree) also defines another sub-polynomial

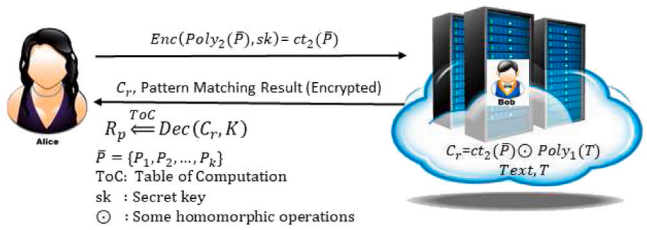

Fig. 2. SPM-RW protocol with symmetric SwHE.

such that $deg(x) < ly$. For instance, consider the problem of searching DNA sequences over the alphabet $\Sigma = \{A, G, C, T\}$. We can encode the alphabet by the set of integers as $\{1, 2, 3, 4\}$. To illustrate the inner product property using our packing method for solving the SPM-RW problem, consider the text $T = (CAGCGACTTG) = (3123213442)$ of length $l = 10$ and the pattern $P = (AGCG * TTG) = (1232 * 442)$ with 2 sub-patterns $P_1 = (1232)$ and $P_2 = (442)$ such that $p_1 = 4$ and $p_2 = 3$. Since $l = 10$ and $k = 2$, we set $n = 32$ to satisfy $n \geq (k+1)l$. If we pack the text T using our packing method in Eq. (16), we can get the polynomial as $Poly_1(T) = 3 + x + 2x^2 + 3x^3 + 2x^4 + x^5 + 3x^6 + 4x^7 + 4x^8 + 2x^9$. Now we pack two sub-patterns in one polynomial using our packing method as $Poly_2(\bar{P}) = x^{10} + 2x^9 + 3x^8 + 2x^7 + 4x^{20} + 4x^{19} + 2x^{18}$. If we multiply $Poly_1(T)$ and $Poly_2(\bar{P})$ according to inner product property in Eq. (22), we get the polynomial as follows:

$$
\begin{aligned}
& Poly_1(T) \times Poly_2(\bar{P}) \\
&= 6x^7 + 11x^8 + 13x^9 + 17x^{10} + 18x^{11} + 16x^{12} + 16x^{13} + 21x^{14} + 27x^{15} \\
&+ 27x^{16} + 18x^{17} + 14x^{18} + 16x^{19} 20x^{20} + 18x^{21} + 24x^{22} + 22x^{23} + 18x^{24} \\
&+ 24x^{25} + 36x^{26} + 36x^{27} + 24x^{28} + 8x^{29} \\
&= \overbrace{(17x^{10} + 18x^{11} + 16x^{12} + 16x^{13} + 21x^{14} + 27x^{15} + 27x^{16})}^{\langle T_d, P_1 \rangle} \\
&+ \overbrace{(20x^{20} + 18x^{21} + 24x^{22} + 22x^{23} + 18x^{24} + 24x^{25} + 36x^{26} + 36x^{27})}^{\langle T_d, P_2 \rangle} \\
&+ \overbrace{(6x^7 + 11x^8 + 13x^9)}^{\text{ToLD}} + \overbrace{(18x^{17} + 14x^{18} + 16x^{19} + 24x^{28} + 8x^{29})}^{\text{ToHD}}. \qquad (23)
\end{aligned}
$$

From the above example, it is obvious that we get two inner products $\langle T_d, P_1 \rangle$ and $\langle T_d, P_2 \rangle$ for our two sub-patterns P_1 and P_2 respectively according to inner product property in Eq. (22). Since $l - p_1 + 1 = 10 - 4 + 1 = 7$, we get total 7 SEDs for the sub-pattern P_1 which is indicated in Eq. (23) as the coefficients of x^{10+d} for $0 \leq d \leq 6$, where $ly = 10 \cdot 1 = 10$ and $l - p_1 = 10 - 4 = 6$. According to Eq. (23), since $l - p_2 + 1 = 10 - 3 + 1 = 8$, we get 8 SEDs for the sub-pattern P_2 as the coefficients of x^{20+d} for $0 \leq d \leq 7$, where $ly = 10 \cdot 2 = 20$ and $l - p_2 = 10 - 3 = 7$. Furthermore, we indicate lower degree sub-polynomial and higher degree sub-polynomial in Eq. (23).

The above inner product property will help us to get all SEDs as the coefficients of x^{ly+d} of a large polynomial. Moreover, we perform the arithmetic computation of the SEDs in Eq. (12) using the polynomials in Eq. (16) as

$$
\begin{aligned}
E_{dis} = & Poly_1(T^2) \cdot Poly_2(V_{|\bar{P}|}) + Poly_2(\bar{P}^2) \cdot Poly_1(V_l) - 2 \cdot Poly_1(T) \\
& \cdot Poly_2(\bar{P}), \qquad (24)
\end{aligned}
$$

where V_l (resp., $V_{|\bar{P}|}$) denotes a vector like (1,1,…,1) of length l (resp., $|\bar{P}|$).

4.4. Our protocols for secure pattern matching with repetitive wildcards

As mentioned in Section 1.2.1, we consider two scenarios for the SPM-RW problem among several parties: one using symmetric ring-LWE based SwHE and another using public-key ring-LWE based SwHE. Moreover, every pair of parties use private and authenticated point-2-point channel for communication to avoid eavesdropping. Now we discuss the SPM-RW protocols using two settings of ring-LWE based SwHE by engaging the technique mentioned in Section 4.3.2.

4.4.1. Our protocol using symmetric encryption

To describe our protocol using symmetric SwHE, we consider a scenario for an SPM between Alice and Bob. For example, Bob has the text vector $T = (a_0, a_1, \ldots, a_{l-1})$ of length l as $Poly_1(T)$ and $Poly_1(T^2)$ using our packing method in Eq. (16) to his secure server which has a huge computation capability. Furthermore, Alice has a pattern $P = (b_{1,0}b_{1,1} \cdots b_{1,p_1-1} * b_{2,0}b_{2,1} \cdots b_{2,p_2-1} * \cdots * b_{k,0}b_{k,1} \cdots b_{k,p_k-1})$ with some wildcards which can be found in the plaintext T. Moreover, Alice divides the pattern into sub-patterns as $\{P_1, P_2, \ldots, P_k\}$ excluding the wildcard characters such that the length of each sub-patterns $P_y = p_y$ with $1 \leq y \leq k$ and $\sum_{y=1}^{k} p_y \leq l$. Then Alice concatenates all sub-patterns sequentially to form a pattern vector as $\bar{P} = (b_{1,0}, b_{1,1}, \ldots, b_{1,p_1-1}, b_{2,0}, b_{2,1}, \ldots, b_{2,p_2-1}, \ldots, b_{k,0}, b_{k,1}, \ldots, b_{k,p_k-1})$. Now Alice does not want to reveal his pattern to Bob. On the contrary, Bob cannot reveal his information outside his server. Here homomorphic encryption property [32] can handle this situation. We know that FHE can perform arbitrary operations which have large time complexity. However, less costly SwHE as mentioned in Section 3.2 is suitable for the above scenario. In addition, as shown in Fig. 2, our protocol can be expressed concisely by the following points.

1. Alice has a pattern P with multiple wildcards. She breaks down the pattern P into sub-patterns $\{P_1, P_2, \ldots, P_k\}$ excluding the wildcard characters to form the vector $\bar{P}$ and packs them as $Poly_2(\bar{P})$ and $Poly_2(\bar{P}^2)$ with our packing method mentioned in Eq. (16).
2. Then, she encrypts $Poly_2(\bar{P})$ and $Poly_2(\bar{P}^2)$ with the homomorphic key generated by herself and sends the encrypted patterns to Bob for performing a secure search.
3. With Eq. (24), Bob performs the required pattern matching computation between text T and encrypted sub-patterns $\bar{P}$ by Eq. (24) and adds some random masks to the result to protect extra information leakage.
4. Bob then returns the resultant polynomial containing the encrypted SEDs to Alice in which all the SEDs can be found separately as the coefficients of x^{ly+d} of the polynomial according to Eq. (22).
5. Alice uses her keys to decrypt the results to check all the SEDs separately whether they are "0" or not. Finally, she uses Algorithm 1 for considering the wildcards in the pattern and calculates her desired result of pattern matching.

4.4.2. Our protocol using public-key encryption

The symmetric SwHE encryption is not able to provide security to the text and the pattern if we consider their owners are different entities and do not want to disclose their information to each other. Consequently, we propose another protocol using a public-key SwHE to provide security to the text and the pattern. For this protocol, we consider an SPM scenario among Alice, Bob, Charlie, and a special party called cryptographic service provider (CSP). For example, let us consider a laboratory (Charlie) has DNA text $T = (a_0 a_1 \cdots a_{l-1})$ of length l. Moreover, a doctor (Alice) wants to match her patient's DNA pattern $P = (b_{1,0}b_{1,1} \cdots b_{1,p_1-1} * b_{2,0}b_{2,1} \cdots b_{2,p_2-1} * \cdots * b_{k,0}b_{k,1} \cdots b_{k,p_k-1})$ with wildcards against the data T belonging to Charlie. In addition, neither Charlie can disclose his data to Alice nor Alice can disclose her patient's DNA pattern to Charlie due to the security. But they are agreed to accomplish the computation in a secured fashion. Moreover, CSP is a trusted party with less resources for many computations whereas all other parties are semi-honest. We assume that Bob and CSP do not collude with one another. If Alice and Bob agree, CSP performs the SPM with the help of another party like Bob in the cloud server who

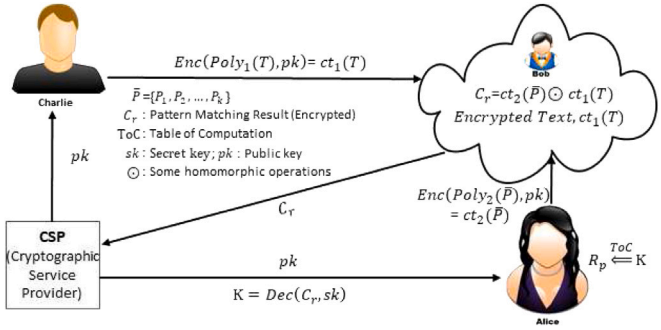

Fig. 3. SPM-RW protocol with public-key SwHE.

has massive computation capability. From the above scenario, the SPM-RW protocol (See Fig. 3) based on public-key SwHE as mentioned in Section 3.4 can be described concisely by the following points.

1. CSP generates a secret key and public key and provides the public key to Alice and Charlie through a secure channel.
2. Using the packing method in Eq. (16), Charlie packs his text T as $Poly_1(T)$ and $Poly_1(T^2)$ and uses the public key to encrypt his text $Poly_1(T)$ and $Poly_1(T^2)$ and sends them to Bob.
3. Then Alice breaks down the pattern P into sub-patterns $\{P_1, P_2, \ldots, P_k\}$ excluding the wildcard characters to form the vector $\bar{P}$. Then she packs $\bar{P}$ as $Poly_2(\bar{P})$ and $Poly_2(\bar{P}^2)$ by utilizing the packing method in Eq. (16).
4. Alice then encrypts $Poly_2(\bar{P})$ and $Poly_2(\bar{P}^2)$ with the public key and sends the encrypted patterns to Bob for performing a secure search.
5. According to Eq. (24), Bob performs the required pattern matching computation between the encrypted text T and all sub-patterns $\bar{P}$ by Eq. (24).
6. Bob then returns the resultant polynomial containing the encrypted SEDs to CSP in which all the SEDs can be found separately as the coefficients of $x^{l\gamma+d}$ of the polynomial according to Eq. (22).
7. CSP decrypts the result using its secret keys and send the result to Alice.
8. Alice check all the SEDs separately whether it equals "0" or does not equal "0". Then she uses Algorithm 1 for considering wildcards in the pattern and calculates the final result of pattern matching.

5. Technique of secure pattern matching with compound wildcards

In this section, we discuss the technique to solve the SPM-CW problem in which we show the application of SPM for PDBQ processing with compound wildcards. In the presentation of database query, the wildcard "%" represents zero or multiple characters. We consider the symbols "∗" and "%" synonymously for the presentation of query with compound wildcards. For example, a pattern "%con$lu%" can be matched with the texts such as "conclude", "conclusion", "conclusive", and so on. In addition, a pattern with compound wildcards "%in!(con)$tab%" that can be used to exclude the words started with "incon" from a word database. For processing the PDBQ, consider a table "Record" with ϖ attributes as $Y = \{\Lambda_1, \Lambda_2, \ldots, \Lambda_\varpi\}$ and τ records. For this table, a user can send a pattern matching query with compound wildcards as *select* $\mathbf{V}$ *from Record where* $\Lambda_{\hat{k}}$ *like "%P%"*, where $\mathbf{V} \in Y$, $\Lambda_{\hat{k}} \in Y$, and $\mathcal{P}$ is a pattern string with compound wildcards for $1 \le \hat{k} \le \varpi$. Now we need to compare the pattern $\mathcal{P}$ with τ records of the table for the attribute $\Lambda_{\hat{k}}$. Moreover, each record of attribute $\Lambda_{\hat{k}}$ can be represented by the text $\mathcal{T} = (a_0 a_1 \cdots a_{\varphi-1})$ of length φ, where a_i is any lowercase letter. Also, the pattern with compound wildcards $\mathcal{P} = (b_0 \cdots \$b_i!(c_{i+1} \cdots c_{i+\xi})b_{i+\xi+1} \cdots \$b_{\vartheta-1})$ of length ϑ with $b_j, c_j \in \mathbb{A}$ in which $\mathbb{A}$ represents the English alphabet. In addition, the wildcard "$" matches any character in the text and "!(π)" matches a single non-empty character except π. Furthermore, we consider the pattern with compound wildcards as a single pattern in which we do not use any pattern splitting technique as mentioned in our SPM-RW protocols. Therefore, we need an efficient method of processing compound wildcards in the pattern to match the records. Here we propose a double-query technique to efficiently process PDBQ with compound wildcards. Let us discuss the technique to solve the SPM-CW problem along with the SPM-CW protocol in the following subsections.

5.1. Double-query technique

We form the integer vectors from the text and the pattern by mapping the letters appearing in that text and pattern to integers. Since the wildcard "$" matches any character in the text, "$" is encoded as 0 for the ease of our pattern matching. Now consider the record $\mathcal{T} = (a_0 a_1 \cdots a_{\varphi-1})$ of length φ ($\varphi < n$), we can pack $\gamma = \lfloor n/\varphi \rfloor$ records of an attribute in a single ciphertext within the lattice dimension n. For γ records $\mathcal{T}_\iota \in \mathbb{Z}_t^\varphi$, we merge them to construct an integer record vector $\mathbf{T} = (\mathcal{T}_1, \ldots, \mathcal{T}_\gamma) \in \mathbb{Z}_t^{\varphi\cdot\gamma}$, where $\mathcal{T}_\iota = (a_{\iota,0}, a_{\iota,1}, \ldots, a_{\iota,\varphi-1}) \in \mathbb{Z}^\varphi$ with $1 \le \iota \le \gamma$. Moreover, we deal with compound wildcards of the form "%$\mathcal{P}$%" in which $\mathcal{P}$ can have "$" and "!" wildcards. For the pattern "%$\mathcal{P}$%", we do not need to process "%" wildcard because it appears at the beginning and end of the pattern in which "%" is replaced with zero or more letters in the text. In addition, consider a pattern string $\mathcal{P} = (b_0 \cdots \$b_i!(c_{i+1} \cdots c_{i+\xi})b_{i+\xi+1} \cdots \$b_{\vartheta-1})$ with "$" and "!" wildcards in which the sub-pattern $c_{i+1} \cdots c_{i+\xi}$ of length ξ appears inside the "!()" wildcard. To handle the "!" wildcard using our technique, a user needs to send two patterns as a query to accomplish the matching for compound wildcards that we call "double-query". Therefore, we call our technique double-query technique. Here final result can be calculated by taking the set difference of two pattern matching results regarding two queries. Now we construct two pattern strings from the single pattern $\mathcal{P}$. The first pattern string is $\mathcal{P}_1 = (b_0 \cdots \$b_i\$ \cdots \$b_{i+\xi+1} \cdots \$b_{\vartheta-1})$ by inserting one "$" character in place of every character in sub-pattern $c_{i+1} \cdots c_{i+\xi}$. Furthermore, the second one is $\mathcal{P}_2 = (b_0 \cdots \$b_i c_{i+1} \cdots c_{i+\xi} b_{i+\xi+1} \cdots \$b_{\vartheta-1})$ which is formed by removing "!()" wildcard from the pattern $\mathcal{P}$. From $\mathcal{P}_1$ and $\mathcal{P}_2$, let us form a general pattern vector $\mathbf{P} = (\mathbf{p}_0, \ldots, \mathbf{p}_{\vartheta-1}) \in \mathbb{Z}_t^\vartheta$ of length ϑ. For $1 \le \iota \le \gamma$ and $0 \le d \le \varphi - \vartheta$, we compute the distance between the pattern vector $\mathbf{P} = (\mathbf{p}_0, \ldots, \mathbf{p}_{\vartheta-1})$ and sub-texts vector $\mathbf{T}_{\iota,d} = (a_{\iota,d}, \ldots, a_{\iota,d+\vartheta-1})$ of $\mathbf{T}_\iota$ by the modified SED technique as follows:

$$E_{\iota,d} = \sum_{h=0}^{\vartheta-1} \mathbf{p}_h \cdot (\mathbf{p}_h - a_{\iota,d+h})^2 . \tag{25}$$

If $E_{\iota,d} = 0$ for some ι and d, the pattern is found in the text $\mathcal{T}_\iota$ at the position $d+1$. To support an efficient batch computation in Eq. (25), we use our packing method in Eq. (16) to pack $\mathbf{T}$ and $\mathbf{P}$ by considering the number sub-pattern $k = 1$. Now the distance between any character in the sub-text and the character "$" will be 0 always, i.e., it will match any character in the sub-text. Using our packing method in Eq. (16), the arithmetic computation in Eq. (25) can be represented as

$$\Delta = Poly_1(\mathbf{V}_{|\mathbf{T}|}) \cdot Poly_2(\mathbf{P}^3) + Poly_1(\mathbf{T}^2) \cdot Poly_2(\mathbf{P}) - 2 \cdot Poly_1(\mathbf{T}) \cdot Poly_2(\mathbf{P}^2), \tag{26}$$

where $\mathbf{V}_{|\mathbf{T}|}$ is a vector like (1,...,1) of length $|\mathbf{T}|$. We calculate $\mathbf{P}^2$ and $\mathbf{P}^3$ from $\mathbf{P} = (\mathbf{p}_0, \ldots, \mathbf{p}_{\vartheta-1})$ with each $\mathbf{p}_h$ being replaced with $\mathbf{p}_h^2$ and $\mathbf{p}_h^3$ respectively. To calculate these distances, we encode the text vectors $\mathbf{T}$ and $\mathbf{T}^2$ using $Poly_1(\mathbf{X})$ encoding, where $\mathbf{X}$ is any integer vector of length $\varphi \cdot \gamma = l$. In addition, the pattern vectors $\mathbf{P}, \mathbf{P}^2$, and $\mathbf{P}^3$ are encoded using $Poly_2(\mathbf{Y})$, where $\mathbf{Y}$ is any integer vector of length ϑ with $k = 1$. Using the inner product property in Eq. (22), we can get each $E_{\iota,d}$ as

the coefficient of $x^{\varphi\cdot\iota+d}$ from Δ in Eq. (26). Since we have two query strings $\mathcal{P}_1$ and $\mathcal{P}_2$ for the pattern $\mathcal{P}$, we need to construct two query vectors $\mathbf{P}_1$ and $\mathbf{P}_2$ as well.

Algorithm 2: Post-processing steps

Data: Δ_1 (Query-1 result vector), Δ_2 (Query-2 result vector), γ (Number of texts), ϑ (Pattern length), φ(Text length)
Result: Result of pattern matching

```
for ι ← 1 to γ do
    flag := false;
    P̂_ι := {} // Pattern Indices
    N̂_ι := {} // Not Pattern Indices
    for d ← 0 to φ − ϑ − 1 do
        if Δ1[φ · ι + d] = 0 then
            P̂_ι.add(φ · ι + d);
            if flag = false then
                flag := true;
            end
        end
    end
    if flag = true then
        for d ← 0 to φ − ϑ − 1 do
            if Δ2[φ · ι + d] = 0 then
                N̂_ι.add(φ · ι + d);
            end
        end
        R_ι := P̂_ι − N̂_ι; // set difference
        if R_ι ≠ {} then
            print("Pattern matched with the ι−th
            text");
        else
            print("No pattern matched");
        end
    end
end
```

5.2. Post processing

Now we describe the post-processing steps that are needed to solve the SPM-CW problem as mentioned in Algorithm 2. In this algorithm, we maintain the sets $\hat{P}_\iota$ and $\hat{N}_\iota$ that contain locations where the pattern $\mathcal{P}_1$ and $\mathcal{P}_2$ match the sub-text respectively. Let Δ_1 and Δ_2 be the polynomials which are constructed by computing the Eq. (26) for $\mathbf{P_1}$ and $\mathbf{P_2}$ respectively. Now we add the indices $d \in [0, \varphi - \vartheta]$ to $\hat{P}_\iota$ if the coefficient of $x^{\varphi\cdot\iota+d}$ in Δ_1 is 0. Similarly, we add the indices $d \in [0, \varphi - \vartheta]$ to $\hat{N}_\iota$ if the coefficient of $x^{\varphi\cdot\iota+d}$ in Δ_2 is 0. The "!()" wildcard character can match any string of same length other than the sub-pattern appeared inside the "!()" wildcard. Therefore, if we take the set difference between sets $\hat{P}_\iota$ and $\hat{N}_\iota$ for each text $\mathcal{T}_\iota$, i.e., $\mathbf{R}_\iota = \hat{P}_\iota - \hat{N}_\iota$, and if the set $\mathbf{R}_\iota$ is not null, i.e., $\mathbf{R}_\iota \neq \{\}$, we can say that the pattern $\mathcal{P}$ matches the text $\mathcal{T}_\iota$.

5.3. Our protocol for secure pattern matching with compound wildcards

In this subsection, we describe the SPM-CW protocol regarding PDBQ processing. Suppose that Alice has the pattern $\mathcal{P}$ with compound wildcards and Charlie is the database owner who has the text $\mathcal{T}$. Here Alice and Charlie want to perform the pattern matching blindly. Now Bob in the cloud can help them in this secure computation. In addition, we use three-party protocol among Alice, Charlie, and Bob in which Alice generates a secret key and a public key by herself and provides the public-key to Bob and Charlie through a secure channel. Since our computation is polynomial-based, we show some extra steps to hide every coefficient of the resultant polynomial except those coefficients which are equal to "0" to provide more security to this protocol. From the above scenario, the SPM-CW protocol can be described by the following steps.

1. Charlie packs his text as $Poly_1(\mathbf{T})$ and $Poly_1(\mathbf{T}^2)$ using the packing method in Eq. (16) and uses Alice's public key to encrypt $Poly_1(\mathbf{T})$ and $Poly_1(\mathbf{T}^2)$ and sends them to Bob.
2. Depending on the compound wildcards appeared in the pattern, Alice constructs two patterns string $\mathcal{P}_1$ and $\mathcal{P}_2$ from $\mathcal{P}$ using the double-query technique and then constructs two integer vectors $\mathbf{P}_1$ and $\mathbf{P}_2$ respectively from them.
3. By utilizing the packing method in Eq. (16), Alice packs $\mathbf{P}_1$ (resp., $\mathbf{P}_2$) as $Poly_2(\mathbf{P}_1)$, $Poly_2(\mathbf{P}_1^2)$, and $Poly_2(\mathbf{P}_1^3)$ (resp., $Poly_2(\mathbf{P}_2)$, $Poly_2(\mathbf{P}_2^2)$, and $Poly_2(\mathbf{P}_2^3)$).
4. Alice then encrypts $Poly_2(\mathbf{P}_1)$, $Poly_2(\mathbf{P}_1^2)$, and $Poly_2(\mathbf{P}_1^3)$ (resp., $Poly_2(\mathbf{P}_2)$, $Poly_2(\mathbf{P}_2^2)$, and $Poly_2(\mathbf{P}_2^3)$) for the double-query with her public key and sends the encrypted patterns to Bob for performing the pattern matching according to Eq. (26).
5. Now Bob performs the required pattern matching computation between the encrypted text $\mathbf{T}$ and two patterns $\mathbf{P}_1$ and $\mathbf{P}_2$ separately and adds some random masks $R_1 \in \mathbb{Z}_\mathbf{t}^n$ and $R_2 \in \mathbb{Z}_\mathbf{t}^n$ to the results respectively through addition.
6. Bob then returns the two encrypted results to Alice. She then decrypts the randomized results and again packs them according to CRT packing method as mentioned in Eq. (14) in Section 4.2.1.
7. Alice now encrypts them again and sends them to Bob for component-wise masking.
8. After receiving the encrypted results from Alice, Bob removes first random mask R_1 and R_2 by subtraction with the help of CRT encoding.
9. Bob again generates two random masks according to CRT packing method and adds the masking to the results respectively through multiplication and sends the randomized results back to Alice.
10. Finally, Alice uses her keys to decrypt the results and checks all distances for each $\mathcal{T}_\iota$ separately whether it is "0" or not. Then she uses our Algorithm 2 to calculate her desired result from the set difference of two results obtained from two patterns $\mathcal{P}_1$ and $\mathcal{P}_2$.

Remark 3. Note that our protocols are secure under the assumption that Bob is semi-honest (also known as honest-but-curious), i.e., Bob always follows the protocols but tries to learn information from the protocols.

6. Many inner products computation

As mentioned in our symmetric SPM-RW protocol in Section 4.4.1, Alice has encrypted pattern $\bar{P}$ and Bob has the text T in the plaintext space R_t. Here we define packed ciphertext $ct_2(M)$ for a plaintext vector $M \in R_t$ using our packing method and symmetric encryption scheme in Section 3.2 as

$$ct_2(M) = Enc(Poly_2(M), sk) \in (R_q)^2 \tag{27}$$

in which M can be $\bar{P}$ and $\bar{P}^2$. Here the following proposition is required to satisfy to compute the SEDs for the SPM-RW protocol using symmetric SwHE scheme in Section 3.2. Moreover, we can prove the proposition with the help of the inner product property in Eq. (22).

Proposition 1. *Let $T = (a_0, a_1, \ldots, a_{l-1}) \in \mathbb{Z}^l$ be a vector of text with $|T| = l$. Moreover, $\bar{P} = (b_{1,0}, b_{1,1}, \ldots, b_{1,p_1-1}, b_{2,0}, b_{2,1}, \ldots, b_{2,p_2-1}, \ldots, b_{k,0}, b_{k,1}, \ldots, b_{k,p_k-1})$ is pattern vector with k sub-patterns in which the length of each sub-pattern is represented by p_y with $\sum_{y=1}^{k} p_y \leq l \leq n$. If the ciphertext of $\bar{P}$ is represented by $ct_2(\bar{P})$ by Eq. (27), under the condition of Lemma 1, decryption of homomorphic multiplication $Poly_1(T) \boxtimes ct_2(\bar{P}) \in (R_q)^2$ will produce a polynomial of R_t with $x^{l\gamma+d}$ including coefficient $\langle T_d, P_y \rangle$ =*

$\sum_{h=0}^{p_y-1} a_{d+h}b_{y,h} \bmod t$ *for* $1 \le y \le k$ *and* $0 \le d \le l-p_y$. *Alternatively, we can say that homomorphic multiplication of* $Poly_1(T)$ *and* $ct_2(\bar{P})$ *simultaneously computes multiple inner products* $\langle T_d, P_y\rangle$ *for* $1 \le y \le k$ *and* $0 \le d \le (l-p_y)$.

Proof. By the correctness in Eq. (5) and Lemma 1, we can say that homomorphic multiplication of $Poly_1(T)$ and $ct_2(\bar{P})$ corresponds to the polynomial multiplication of $Poly_1(T)$ and $Poly_2(\bar{P})$ in the ring R_t. In addition, we already know from the inner product property in Eq. (22), the polynomial multiplication of $Poly_1(T)$ and $Poly_2(\bar{P})$ produces many inner products $\langle T_d, P_y\rangle$ as the coefficients of x^{ly+d}, which equals $\sum_{h=0}^{p_y-1} a_{d+h}b_{y,h}$ for $1 \le y \le k$ and $0 \le d \le l-p_y$. Consequently, the proposition holds under the correctness in Eq. (5). □

For our public-key SPM-RW protocol in Section 4.4.2, Alice has encrypted pattern of $\bar{P}$ and Bob has the ciphertext of T in R_q. Now we define packed ciphertext of a vector $N \in R_t$ using our packing method and public-key encryption scheme in Section 3.4 for $\omega = \{1, 2\}$ as

$$ct_\omega(N) = Enc(Poly_\omega(N), pk) \in (R_q)^2 \tag{28}$$

in which vector N can be replaced with T, T^2, $\bar{P}$, and $\bar{P}^2$ separately. We define another proposition that is also needed to satisfy to compute the SEDs for SPM-RW protocol based on public-key SwHE scheme in Section 3.4. In addition, we can prove the following proposition by using the inner product property in Eq. (22).

Proposition 2. *Consider the same text vector* T *and pattern vector* $\bar{P}$ *as defined in Proposition 1. If the ciphertext of* T *(resp.,* $\bar{P}$*) can be expressed by* $ct_1(\bar{T})$ *(resp.,* $ct_2(\bar{P})$*) by Eq. (28), under the condition of Lemma 2, decryption of homomorphic multiplication* $ct_1(T) \boxtimes ct_2(\bar{P}) \in (R_q)^2$ *will produce a polynomial of* R_t *with* x^{ly+d} *including coefficient* $\langle T_d, P_y\rangle = \sum_{h=0}^{p_y-1} a_{d+h}b_{y,h} \bmod t$ *for* $1 \le y \le k$ *and* $0 \le d \le l-p_y$. *Strictly speaking, homomorphic multiplication of* $ct_1(T)$ *and* $ct_2(\bar{P})$ *simultaneously computes multiple inner products* $\langle T_d, P_y\rangle$ *for* $1 \le y \le k$ *and* $0 \le d \le (l-p_y)$.

Proof. With the help of correctness in Eq. (11) and Lemma 2, we can say that homomorphic multiplication of $ct_1(T)$ and $ct_2(\bar{P})$ is comparable to the polynomial multiplication of $Poly_1(T)$ and $Poly_2(\bar{P})$ in the ring R_t. In addition, we already know from the inner product property in Eq. (22), the polynomial multiplication of $Poly_1(T)$ and $Poly_2(\bar{P})$ produces many inner products $\langle T_d, P_y\rangle$ as the coefficients of x^{ly+d}, which equals $\sum_{h=0}^{p_y-1} a_{d+h}b_{y,h}$ for $1 \le y \le k$ and $0 \le d \le l-p_y$. As a result, the proposition holds under the correctness in Eq. (11). □

7. Secure computation of pattern matching with repetitive wildcards

Through Eq. (24), we have already shown the calculation of many SEDs in Eq. (12) using our packing method for the SPM-RW problem. In our symmetric and public-key SPM-RW protocols, Bob calculates Eq. (24) homomorphically over the packed ciphertexts, which is shown in the following subsections.

7.1. Symmetric secure pattern matching with repetitive wildcards

As mentioned in our protocol in Section 4.4.1, symmetric SPM-RW protocol occurs between plaintext T and encrypted pattern $\bar{P}$. To compute the Eq. (24) homomorphically, we construct the ciphertexts of $Poly_2(\bar{P})$, $Poly_2(\bar{P}^2)$, and $Poly_2(V_{|\bar{P}|})$ as $ct_2(\bar{P})$, $ct_2(\bar{P}^2)$, and $ct_2(V_{|\bar{P}|})$ that can be obtained by Eq. (27). In addition, the following theorem requires for this computation.

Theorem 1. *Under the condition Lemma 1, the linear combination of homomorphic operations*

$$Poly_1(T^2) \boxtimes ct_2(V_{|\bar{P}|}) \boxplus Poly_1(V_l) \boxtimes ct_2(\bar{P}^2) \boxplus (-2Poly_1(T) \boxtimes ct_2(\bar{P})) \tag{29}$$

simultaneously computes multiple values of Eq. (12) using our packing method and Eq. (27) for $0 \le d \le (l-p_y)$ *and* $1 \le y \le k$ *on encrypted data in which* $V_{|\bar{P}|}$ *denotes the unit vector* $(1, 1, \ldots, 1)$ *of length* $|\bar{P}|$ *and* V_l *denotes another unit vector* $(1, 1, \ldots, 1)$ *of length* l. *Concretely, the homomorphic operation in Eq. (29) gives a polynomial of* R_t *in which the coefficient of* x^{ly+d} *is equal to* $E_{y,d}$ *in Eq. (12) for each* $0 \le d \le (l-p_y)$ *and* $1 \le y \le k$ *on encrypted data.*

Proof. The property in Eq. (22) and Proposition 1 show that each x^{ly+d}-th coefficient of $Poly_1(T^2) \boxtimes ct_2(V_{|\bar{P}|})$ is equal to the sum $(a_d^2, a_{d+1}^2, \ldots, a_{d+p_y-1}^2) \cdot (1, 1, \ldots, 1)^T = \sum_{h=0}^{p_y-1} a_{d+h}^2$ for $0 \le d \le (l-p_y)$ and $1 \le y \le k$, where A^T denotes the transpose of a vector A. Similarly, we can say that the homomorphic multiplication of $Poly_1(V_l) \boxtimes ct_2(\bar{P}^2)$ and $(-2Poly_1(T) \boxtimes ct_2(\bar{P}))$ computes two polynomials on encrypted data with the x^{ly+d}-th coefficient, which are equal to $\sum_{h=0}^{p_y-1} b_{y,h}^2$ and $-2\sum_{h=0}^{p_y-1} a_{d+h} \cdot b_{y,h}$ respectively for each $0 \le d \le (l-p_y)$ and $1 \le y \le k$. Finally, we can say by the correctness in Eq. (5), it proves that homomorphic operation in Eq. (29) constructs a polynomial of R_t in which the coefficient x^{ly+d} is equal to $E_{y,d}$ in Eq. (12) for each $0 \le d \le (l-p_y)$ and $1 \le y \le k$ on encrypted data. □

7.2. Public-key secure pattern matching with repetitive wildcards

Since the text T and the pattern $\bar{P}$ are encrypted in public-key SPM-RW protocol in Section 4.4.2, we require to state another theorem than Theorem 1. We construct the ciphertexts of $Poly_1(T)$, $Poly_1(T^2)$, $Poly_1(V_l)$, $Poly_2(\bar{P})$, $Poly_2(\bar{P}^2)$, and $Poly_2(V_{|\bar{P}|})$ as $ct_1(T)$, $ct_1(T^2)$, $ct_1(V_l)$, $ct_2(\bar{P})$, $ct_2(\bar{P}^2)$, and $Poly_2(V_{|\bar{P}|})$ respectively with Eq. (28) to calculate Eq. (24) homomorphically.

Theorem 2. *Under the condition Lemma 2, the linear combination of homomorphic operations*

$$ct_1(T^2) \boxtimes ct_2(V_{|\bar{P}|}) \boxplus ct_1(V_l) \boxtimes ct_2(\bar{P}^2) \boxplus (-2ct_1(T) \boxtimes ct_2(\bar{P})) \tag{30}$$

simultaneously computes multiple values of Eq. (12) with the help of Eq. (28) for $0 \le d \le (l-p_y)$ *and* $1 \le y \le k$ *on encrypted data in which* $V_{|\bar{P}|}$ *denotes the unit vector* $(1, 1, \ldots, 1)$ *of length* $|\bar{P}|$ *and* V_l *denotes another unit vector* $(1, 1, \ldots, 1)$ *of length* l. *Precisely, the homomorphic operation in Eq. (30) gives a polynomial of* R_t *in which the coefficient of* x^{ly+d} *is equal to* $E_{y,d}$ *in Eq. (12) for each* $0 \le d \le (l-p_y)$ *and* $1 \le y \le k$ *on encrypted data.*

Proof. The proof of this theorem is nearly same as Theorem 1. The inner product property in Eq. (22) and Proposition 2 show that each x^{ly+d}-th coefficient of $ct_1(T^2) \boxtimes ct_2(V_{|\bar{P}|})$ is equal to the sum $(a_d^2, a_{d+1}^2, \ldots, a_{d+p_y-1}^2) \cdot (1, 1, \ldots, 1)^T = \sum_{h=0}^{p_y-1} a_{d+h}^2$ for $0 \le d \le (l-p_y)$ and $1 \le y \le k$, where A^T denotes the transpose of a vector A. Similarly, we can say that the homomorphic multiplication of $ct_1(V_l) \boxtimes ct_2(\bar{P}^2)$ and $(-2ct_1(T) \boxtimes ct_2(\bar{P}))$ computes two polynomials on encrypted data with the x^{ly+d}-th coefficient, which are equal to $\sum_{h=0}^{p_y-1} b_{y,h}^2$ and $-2\sum_{h=0}^{p_y-1} a_{d+h} \cdot b_{y,h}$ respectively for each $0 \le d \le (l-p_y)$ and $1 \le y \le k$. Finally, we can say by the correctness (11), it proves that homomorphic operation in Eq. (30) produces a polynomial of R_t in which the coefficient x^{ly+d} is equal to $E_{y,d}$ in Eq. (12) for each $0 \le d \le (l-p_y)$ and $1 \le y \le k$ on encrypted data. □

7.3. Random masking

In our SPM-RW protocols, we have already used masking to suppress information leakage. Now we discuss the reasons for information leakage in the protocols and their countermeasures through masking as follows.

7.3.1. Information leakage problem

For the SPM-RW protocols, information leakage problem may occur if Bob sends an encrypted polynomial to Alice without any masking after homomorphic calculation. Here our protocols perform the pattern matching between text T and pattern $\bar{P}$. Both of our protocols calculate the pattern matching in the cloud with the decryption help from Alice for the decision making. For the two protocols of the SPM-RW problem, the cloud calculates the SED $E_{y,d}$ in Eq. (12) by engaging the arithmetic computations in Eqs. (29) and (30) for symmetric and public-key case respectively. From these computations in Eqs. (29) and (30), it is obvious that Bob obtains $ct(E_{y,d})$ using three polynomial multiplications and two additions homomorphically. Moreover, the polynomial multiplication produces a large polynomial; Alice uses a sub-polynomial of the large polynomial to get all $E_{y,d}$ as the coefficients of x^{ly+d}. In our protocols, if Bob would send the encrypted polynomial to Alice after the homomorphic computation without any masking, Alice can get some extra information from that polynomial, whereas she requires only the coefficients of x^{ly+d}.

7.3.2. Countermeasures

To suppress the information leakage problem mentioned in Section 7.3.1, we can use two countermeasures as follows.

Random polynomial addition. In our SPM-RW protocols, we can hide the extra information from leakage to Alice by using some random masks at the cloud (Bob) ends. Bob adds a random polynomial r to $ct(E_{y,d})$ for masking extra information since Alice needs to check only the coefficient of x^{ly+d} from the large polynomial $ct(E_{y,d})$ produced by Bob. The random polynomial in the ring R can be represented by the following equation:

$$r = \sum_{i=0}^{l-1} r_i + \sum_{y=1}^{k} \sum_{j=1}^{p_y-1} r_{(y+1)\cdot l-j} x^{(y+1)\cdot l-j}.$$

We can use the above random mask to hide all coefficient of $ct(E_{y,d})$ except the coefficients x^{ly+d}. Here $ct(E_{y,d})$ consists of three ciphertext components as $ct(E_{y,d}) = (c_0, c_1, c_2)$. Now Bob adds r to the ciphertext as $ct(E'_{y,d}) = ct(E_{y,d}) \boxplus r = (c_0 \boxplus r, c_1, c_2)$. Then the resulting ciphertext $ct(E'_{y,d})$ contains all required information as the coefficients of x^{ly+d} and hide all other coefficients using the random mask r. In this way, we can protect $ct(E_{y,d})$ from leaking any information to Alice except the coefficients of x^{ly+d}.

Random vector multiplication. As we know from our protocols, Alice needs to find out only the coefficients of x^{ly+d} with having "0" from the degree $n-1$ polynomial $ct(E_{y,d})$ produced by Bob after decryption. Here we can hide all the n coefficients except "0" by multiplying a small random $r_i \in \mathbb{Z}_t^*$ with every coefficient through component-wise multiplication for $0 \le i \le n-1$. Now we represent n coefficients of the polynomial $ct(E_{y,d})$ as a vector $\mathbf{C} = (\zeta_0, \zeta_1, \ldots, \zeta_{n-1})$ and the random vector as $\mathbf{R} = (r_0, r_1, \ldots, r_{n-1})$. Then the masking can be represented by the equation as follows:

$$\mathbf{C} \times \mathbf{R} = (\zeta_0 \cdot r_0, \zeta_1 \cdot r_1, \ldots, \zeta_{n-1} \cdot r_{n-1}). \tag{31}$$

To achieve the above masking over the encrypted polynomial, we can use the CRT packing as mentioned in Eq. (14) in Section 4.2.1. By using CRT packing, we can perform the masking of the results for two protocols using the component-wise homomorphic operations over the ciphertexts by the following steps.

1. Bob adds the random mask $R_1 \in \mathbb{Z}_{\mathbf{t}}^n$ to the encrypted result through addition.
2. Bob then returns the encrypted results to Alice, and she then decrypts the randomized results and again packs them according to CRT packing method in Eq. (14).
3. Alice now encrypts them again and sends them to Bob for component-wise masking.
4. After receiving the encrypted results from Alice, Bob removes first random mask R_1 by subtraction with the help of CRT packing.
5. Bob again constructs another random mask in $\mathbb{Z}_{\mathbf{t}}^n$ with the help CRT packing method to allow component-wise masking.
6. Now Bob adds the masking to the result through multiplication and sends the randomized results back to Alice.

Using the above technique, the computation time of masking will be increased slightly due to using new packing. In addition, the multiplication technique reveals the coefficients containing "0" only, which occurs at some coefficients of x^{ly+d}. On the contrary, the random polynomial addition method reveals all SEDs appeared at the coefficients of x^{ly+d}, but the technique does not reveal any information regarding actual data. Therefore, the random vector multiplication approach can hide more information than the random polynomial addition method. However, both methods of masking secure actual data regarding the pattern and the text.

Now we can use any of the above solutions to accomplish the masking mentioned in our SPM-RW protocols in order to suppress the information leakage. However, we follow the random polynomial addition to add masking in the SPM-RW protocols for the efficiency.

7.4. An example of symmetric secure pattern matching

After the computation of many SEDs for each sub-pattern, we give an example to show how the pattern matching is performed on the text by using Algorithm 1 by maintaining the sequence of sub-patterns. The text and the pattern along with sub-pattern matching indices are depicted in the table of computation (ToC) which is shown in Fig. 4. Suppose that Bob has the plaintext T = AAGTGCTGCCAGTCGT such that $|T| = 16$ and Alice has the pattern P = GTGCT*CC*GT*T with $k = 4$. Here we get the sub-patterns as P_1 = (GTGCT), P_2 = (CC), P_3 = (GT), and P_4 = (T). Then the length vector (p_1, p_2, p_3, p_4) of sub-patterns is $(5, 2, 2, 1)$. We get our parameters for the pattern matching as $0 \le d \le (16 - p_y)$ and $1 \le y \le 4$. Furthermore, we represent the alphabet Σ = {A, G, C, T} as $\{1, 2, 3, 4\}$ to encode the text and the pattern using non-binary vectors. By encoding algorithm, the text T can be represented as T = (1124234233124324). We also encode all the sub-patterns as P_1 = (24234), P_2 = (33), P_3 = (24), and P_4 = (4). Next, we concatenate all the sub-patterns as $\bar{P} = \{(24234), (33), (24), (4)\}$. By using our packing method in Eq. (16), Bob will have the text $Poly_1(T)$ and $Poly_1(T^2)$. Next, Alice encrypts all sub-patterns using her key into a single polynomial by $Poly_2(\bar{P})$ and $Poly_2(\bar{P}^2)$. Finally, she encrypts $Poly_2(\bar{P})$ and $Poly_2(\bar{P}^2)$ as $ct_2(\bar{P})$ and $ct_2(\bar{P}^2)$ respectively and sends them to Bob for pattern matching computation. Bob accomplishes the pattern matching according to Theorem 1 to compute $E_{y,d}$ in Eq. (12). He also adds some random masks to the polynomial coefficients with the degree other than $ly + d$. Then Bob returns the result to Alice in encrypted form. Alice then decrypts the result and finds out the coefficients of x^{ly+d} and determines the indices where each sub-pattern matches the actual text using Algorithm 1 as shown in Fig. 4.

Here sub-patterns P_1, P_2, and P_3 match at the indices 2, 8, and 11 respectively. However, sub-pattern P_4 matches at the four indices 3, 6, 12, and 15. Now Alice considers only that index which is greater than the sum of the matched index of P_3 and $p_3 - 1$ ($> 11 + 2 - 1 = 12$). Hence, the index will be 15. Finally, she computes the gaps between sub-patterns and matches the wildcards between sub-patterns. In this way, Alice finds her desired result using our protocol. Moreover, our algorithm does not match a pattern like "CC*GT*GTGCT*T" with given plaintext T though every sub-pattern exists in the plaintext. This mismatch happens because all sub-patterns do not occur sequentially in the text. Therefore, this algorithm accepts only sub-strings with the same order as in the main string. Furthermore, this algorithm determines only the single occurrence of the pattern P in the text T to reduce the complexity of computation.

Text = AAGTGCTGCCAGTCGT, Pattern = GTGCT*CC*GT*T
X = Sub-Pattern Matches Index

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Text	A	A	G	T	G	C	T	G	C	C	A	G	T	C	G	T
Pattern			G	T	G	C	T	-	C	C	-	G	T	-	-	T

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Subpattern-1			x													
Subpattern-2									x							
Subpattern-3												x				
Subpattern-4				x		x							x			x

Fig. 4. Table of computation for finding pattern matching results.

8. Secure computation of pattern matching with compound wildcards

For the sake of our computation in Eq. (26) over the encrypted text **T** and the encrypted pattern **P**, we need to define another theorem than Theorem 2 for the SPM-CW protocol as mentioned in Section 5.3. By using Eq. (28), the packed ciphertexts of $Poly_1(\mathbf{T})$, $Poly_1(\mathbf{T}^2)$, $Poly_1(\mathbf{V}_{|\mathbf{T}|})$, $Poly_2(\mathbf{P})$, $Poly_2(\mathbf{P}^2)$, and $Poly_2(\mathbf{P}^3)$ are obtained as $ct_1(\mathbf{T})$, $ct_1(\mathbf{T}^2)$, $ct_1(\mathbf{V}_{|\mathbf{T}|})$, $ct_2(\mathbf{P})$, $ct_2(\mathbf{P}^2)$, and $ct_2(\mathbf{P}^3)$ respectively.

Theorem 3. *Under the condition Lemma 2, the linear combination of homomorphic operations*

$$ct_1(\mathbf{V}_{|\mathbf{T}|}) \boxtimes ct_2(\mathbf{P}^3) \boxplus ct_1(\mathbf{T}^2) \boxtimes ct_2(\mathbf{P}) \boxplus (-2ct_1(\mathbf{T}) \boxtimes ct_2(\mathbf{P}^2)) \tag{32}$$

simultaneously computes multiple values of Eq. (25) with the help of Eq. (28) for $0 \le d \le \varphi - \vartheta$ *and* $1 \le \iota \le \gamma$ *on the encrypted data, where* $\mathbf{V}_{|\mathbf{T}|}$ *denotes the unit vector* $(1, 1, \ldots, 1)$ *of length* $|\mathbf{T}|$. *Concretely, the homomorphic operation in Eq. (32) gives a polynomial of* R_t *in which the coefficient of* $x^{\varphi\cdot\iota+d}$ *is equal to* $E_{\iota,d}$ *in Eq. (25) for each* $0 \le d \le \varphi - \vartheta$ *and* $1 \le \iota \le \gamma$ *on encrypted data.*

Proof. We can prove this theorem, which is nearly similar to Theorem 2. The inner product property in Eq. (22) and Proposition 2 show that each $x^{\varphi\cdot\iota+d}$-th coefficient of $ct_1(\mathbf{V}_{|\mathbf{T}|}) \boxtimes ct_2(\mathbf{P}^3)$ is equal to the sum $(\mathbf{p}_0^3, \mathbf{p}_1^3, \ldots, \mathbf{p}_{\vartheta-1}^3) \cdot (1, 1, \ldots, 1)^T = \sum_{h=0}^{\vartheta-1} \mathbf{p}_h^3$, where A^T denotes the transpose of a vector A. Similarly, we can say that the homomorphic multiplication of $ct_1(\mathbf{T}^2) \boxtimes ct_2(\mathbf{P})$ and $(-2ct_1(\mathbf{T}) \boxtimes ct_2(\mathbf{P}^2))$ computes two polynomials on encrypted data with the $x^{\varphi\cdot\iota+d}$-th coefficient, which are equal to $\sum_{h=0}^{\vartheta-1} a_{\iota,d+h}^2 \cdot \mathbf{p}_h$ and $-2\sum_{h=0}^{\vartheta-1} a_{\iota,d+h} \cdot \mathbf{p}_h^2$ respectively for each $0 \le d \le (\varphi - \vartheta)$ and $1 \le \iota \le \gamma$. Finally, we can say by the correctness in Eq. (11), it proves that homomorphic operation in Eq. (32) produces a polynomial of R_t in which the coefficient $x^{\varphi\cdot\iota+d}$ is equal to $E_{\iota,d}$ in Eq. (25) for each $0 \le d \le (\varphi - \vartheta)$ and $1 \le \iota \le \gamma$ on encrypted data. □

Here we get each $E_{\iota,d}$ as the coefficient of $x^{\varphi\cdot\iota+d}$ of the large polynomial in Eq. (32). For our pattern matching, we constructed two patterns $\mathcal{P}_1$ and $\mathcal{P}_2$ from the pattern $\mathcal{P}$. In our SPM-CW protocol, Alice will get two pattern matching results $ct(\Delta_1)$ and $ct(\Delta_2)$ for her two queries $\mathcal{P}_1$ and $\mathcal{P}_2$ according to Eq. (32). Alice then decrypts the results $ct(\Delta_1)$ and $ct(\Delta_2)$ using her key to obtain Δ_1 and Δ_2. She uses Algorithm 2 to calculate the final result of pattern matching from the set difference of two results obtained from Δ_1 and Δ_2.

8.1. Random masking

In our SPM-CW protocol, we have already used masking to suppress information leakage. Now we discuss the reasons for information leakage and its countermeasure through masking as follows.

8.1.1. Information leakage problem

The information leakage problem may occur in our SPM-CW protocol if Bob sends the encrypted polynomial to Alice without any masking after pattern matching computation. In this protocol, we use the modified Euclidean distance-based pattern matching technique in which a pattern can be found in the text if the distance between them is "0". Since we use the double-query technique as mentioned in Section 5.1, Bob calculates Δ_1 and Δ_2 for matching the pattern $\mathcal{P}_1$ and $\mathcal{P}_2$ with the text according to Eq. (32), which includes three polynomial multiplications and two additions. According to Eq. (32), Bob produces two large polynomials $ct(\Delta_1)$ and $ct(\Delta_2)$ which are sent to Alice for decryption to decide pattern matching. According to inner product property in Eq. (22), Alice needs a sub-polynomial to get the result as the coefficients of $x^{\varphi\cdot\iota+d}$. Therefore, Alice can get more information than she requires for the pattern matching if Bob does not use masking.

8.1.2. Countermeasure

To provide additional security due to the information leakage as mentioned in Section 8.1.1, we used the technique of random vector multiplication of Section 7.3.2 using CRT packing that is already mentioned in our SPM-CW protocol.

Remark 4. To achieve efficiency, we follow the semi-honest model for our protocols. To get maliciously secure protocols, we can convert them using existing compilers [33,34] which will cause a certain cutback in their performances.

9. Performance analysis

We discuss the performances of the SPM-RW and SPM-CW protocols along with their security in the following subsections.

9.1. Security weaknesses and countermeasures

Here we show the probable security weaknesses of our protocols and their possible countermeasures.

9.1.1. Security weaknesses

We have already said that our protocols are secure in the semi-honest model. Here we mean that Bob tries to learn information from the protocols but follows the protocols. We consider two weaknesses for our protocols. Firstly, in the asymmetric versions of the SPM-RW and SPM-CW protocols, Charlie cannot disclose his data to Alice but sends his data to Bob using Alice's public-key. Moreover, Bob being a semi-honest party can collude with Alice to bypass the computation of the SPM to Alice. Now, both Alice and Bob can be benefited from the collusion. For instance, Bob can save its computing power, and Alice can get access to the data that she has no permission to access. Secondly, our protocols are unable to prevent the more real-world attacks by the malicious adversaries because we follow the semi-honest model for protocols' security.

9.1.2. Countermeasures against the weaknesses

To solve the first weakness, we can add a trusted third party (TTP) as a service provider with our asymmetric protocol scenario. Furthermore, TTP in our protocol will be responsible to generate keys, provide the keys to Alice and Bob, decrypt the results after computation by Bob, and send the results to Alice. Here Charlie and Bob will have the same role, but Alice will lose his power of key generation and decryption. After the addition of TTP in our protocols, Bob tries to collude with Alice but fails because Alice has no secret key to decrypt Charlie's data. To solve the second weakness, we can convert the protocols to become maliciously secure using some existing compilers [33,34]. In the second case, we need to compromise the performances of the protocols due to adding malicious security to them.

Table 1
Performance comparison for SPM-RW protocols.

Encryption type	Parameters	Yasuda et al.'s method [6]	Our method
Both	No. of homomorphic multiplications	$3k$	3
	Cost of homomorphic multiplication	$\mathcal{O}(3k(\lceil l/n \rceil \cdot \lceil \lvert\bar{P}\rvert/n \rceil))$	$\mathcal{O}(3(\lceil l/n \rceil \cdot \lceil \lvert\bar{P}\rvert/n \rceil))$
Symmetric	Communication complexity	$\mathcal{O}(kC_1 + C_2)$	$\mathcal{O}(C_1 + C_2)$
Public-Key	Communication complexity	$\mathcal{O}(kC_1 + C_3)$	$\mathcal{O}(C_1 + C_3)$

n = lattice dimension;
$|\bar{P}|$ = total length of sub-patterns, k = number of sub-patterns in a query.
C_1 = communication cost for each query and its answer between Alice and Bob.
C_2 = communication cost for sending the text from Alice to Bob.
C_3 = communication cost for sending the text from Charlie to Bob.

9.2. Theoretical evaluation

To present the efficiency of our packing method, we evaluate the performance of the SPM-RW protocols theoretically, which are shown in Table 1. Let us consider a performance parameter "number of homomorphic multiplications" due to cryptographic perspective and "cost of homomorphic multiplication" to differentiate our method and Yasuda et al.'s method [6] to calculate pattern matching for our protocols. Moreover, recall the same text T and pattern $\bar{P}$ with k sub-patterns. In addition, let C_1 be the communication cost for each query and its answer between Alice and Bob. Furthermore, C_2 defines the communication cost for sending the text from Alice to Bob for the SPM-RW protocol using symmetric encryption. Also, C_3 defines the communication cost for sending the text from Charlie to Bob for the SPM-RW protocol using the public-key encryption. Now we compare our method and the method in [6] to evaluate the SPM-RW protocols with the symmetric and public-key encryption as shown in Table 1. Now we show the comparison between Yasuda et al.'s method [6] and our method regarding the cost of homomorphic multiplications and the communication complexity to solve the SPM-RW problem. For both symmetric and public-key case, the cost of homomorphic multiplication using Yasuda et al.'s method is $\mathcal{O}(3k(\lceil l/n \rceil \cdot \lceil |\bar{P}|/n \rceil))$, whereas the cost of homomorphic multiplication using our method is $\mathcal{O}(3(\lceil l/n \rceil \cdot \lceil |\bar{P}|/n \rceil))$. In case of symmetric SPM-RW protocol, the communication complexity of Yasuda et al.'s method is $\mathcal{O}(kC_1 + C_2)$, whereas the communication complexity of our method is $\mathcal{O}(C_1 + C_2)$. In case of public-key SPM-RW protocol, the communication complexity of Yasuda et al.'s method is $\mathcal{O}(kC_1 + C_3)$, whereas the communication complexity of our method is $\mathcal{O}(C_1 + C_3)$. As a result, we have been able to reduce the communication complexity of [6] by a factor near about k. Due to using our packing method, we have been able to handle k sub-patterns in three multiplications instead of $3k$ multiplications if we follow the method in [6]. Therefore, a simplification of multiplication is done here.

Besides, we show the functional difference between the SPM-RW protocol and Yasuda et al.'s protocol [6] as shown in Table 2. Here the pattern in our SPM-RW protocols contains repetitive wildcards, whereas pattern in Yasuda et al.'s protocol contains a single wildcard. Furthermore, a wildcard in the pattern replaces with one letter in the text using Yasuda et al.'s protocol [6], whereas the same replaces with zero or many letters in the text using our protocols as shown in Table 2. Moreover, our protocols suppress the information leakage, which is absent in Yasuda et al.'s protocol [6].

Remark 5. We observed that our method has a computation overhead of doing large polynomial multiplication of dimension n, whereas we require a sub-polynomial of the multiplication result to find the required inner products according to Eq. (22). The overhead is unavoidable in our method due to using polynomial ring-based SwHE for the protocols' security.

Table 2
Functional difference between SPM-RW protocols and Yasuda et al.'s protocol [6].

Parameters	Yasuda et al.'s protocol [6]	SPM-RW protocols
Nature of pattern	Single wildcard	Repetitive wildcards
Text replacing behavior by a wildcard	One letter	Zero or many letters
Information leakage	Yes	No

9.3. Security level

Before we estimate the values of the parameters, we consider the security of our schemes from decoding attack [35] and distinguishing attack [36]. According to Lindner and Peikert [35], we consider every parameter setting to provide more than 80-bit security level to secure our protocols against the distinguishing attack and more powerful decoding attack in which the adversarial advantage $\epsilon = 2^{-64}$. In addition, Chen and Nguyen [37] estimated in lattice-based cryptographic schemes that it is required to have the root Hermite factor $\delta < 1.0050$ to achieve an 80-bit security level. As discussed in [21], the running time t_{adv} is defined as $\log(t_{adv}) = 1.8/\log(\delta) - 110$ in which the root Hermite factor δ is expressed as $c \cdot q/\sigma = 2^{2\sqrt{n \cdot \log(q) \cdot \log(\delta)}}$.

9.4. Parameter settings

Here we provide the parameter settings of the SPM-RW protocols for searching DNA sequences and the SPM-CW protocol for processing PDBQ.

9.4.1. Secure pattern matching with repetitive wildcards

In our SPM-RW protocols, we set the parameters (n, q, t, σ, k) to perform the experiments. We encoded the DNA alphabet $\Sigma = \{\mathtt{A}, \mathtt{G}, \mathtt{C}, \mathtt{T}\}$ as $\Sigma = \{1, 2, 3, 4\}$ for simplifying our pattern matching computation. For our experiments, we implemented the SPM-RW protocols using our method and Yasuda et al.'s methods [6] in C programming language with Pari C library (version 2.7.5) [38] and ran on a computer with Intel® Core i7-4790 CPU with 3.60 GHz and 8 GB RAM. We compiled our C code using gcc 5.4.0 in Linux environment.

As mentioned in Section 4.2.4, since $n \geq (k+1)l$ and degree of second polynomial of our packing method increases with the increase of l, we perform our experiments by setting the value of n and l as small as possible to achieve a better performance than the existing method. Moreover, we found in [2] that the value of n should be at least 2048 to make the encryption secure. Then we consider $n = 2048$ for all experiments to achieve more than 80-bit security level for the experiments to protect our protocols from some distinguishing attacks. We also keep same lattice dimension for the experiments using Yasuda et al.'s method [6]. We have chosen the values of our required parameters (q, t, σ, k) carefully to comply with our method. We set $\sigma = 8$ and vary other parameters (t, q, k) for our experimental evaluations of symmetric and public-key case.

In the symmetric case, according to work in [6], the value of t should satisfy $t \geq 2^8 n$ to avoid carry operation since the SED requires two additions as mentioned in Eq. (29). Moreover, $q > 2^{11} nt\sigma = 2^{11} \cdot 2^{11} \cdot 2^{19} \cdot 2^3 = 2^{44}$ for the ciphertext space R_q. Therefore, we set $q = 45$ bits for the results shown in Table 3. In the public-key case, we set $t = 2048$ and $q > 16n^2t^2\sigma^4 = 2^4 \cdot 2^{22} \cdot 2^{22} \cdot 2^{12} = 2^{60}$ according to work in [2]. Therefore, we set $q = 61$ bits for the results shown in Table 4.

To evaluate the SPM-RW protocols, we considered several texts of the lengths from 128 to 512. For $l = 512$, we took 2 (resp., 3) sub-patterns of length 6 (resp., 9) characters. Moreover, we took 5 (resp., 7) sub-patterns of length 15 (resp., 21) characters for $l = 256$. Also, we took 11 (resp., 15) sub-patterns of length 33 (resp., 45) characters for $l = 128$. Furthermore, we limit the value of l from 128 to 512 to keep the lattice dimension $n = 2048$ to realize the benefits of our scheme. In addition, we believe that high lattice dimension and more text length can be supported if the experiments are conducted in real cloud server where the ciphertext sending cost through a network is more realized.

Table 3
Experimental comparison for symmetric SPM-RW protocol.

Text length (l)	No. of sub-patterns (k)	Pattern length $\|\bar{P}\|$	Total time (ms[a])		Time of Hom.Op.[b] (ms[a])		Improvement factor (Hom.Op.[b])	Security level $\log(t_{Adv})$
			Yasuda et al.'s method [6]	Our method	Yasuda et al.'s method [6]	Our method		
512	2	6	125	62	110	60	**1.83**	234
	3	9	204	78	141	63	**2.24**	
256	5	15	203	46	170	31	**5.48**	
	7	21	312	47	210	31	**6.77**	
128	11	33	328	31	235	31	**7.58**	
	15	45	453	31	297	31	**9.58**	

[a]ms: milliseconds.
[b]Hom.Op: Homomorphic operations.

9.4.2. Secure pattern matching with compound wildcards

We set $(n, t, q, \sigma) = (2048, 262144, 75 \text{ bits}, 8)$ for the experiments of processing PDBQ regarding the SPM-CW protocol. Moreover, we used the encoding of input alphabet $\Sigma = \{a - z\}$ as the integers in $\mathbb{Z}-\{0\}$ and the wildcard "$" is encoded as zero. Here we implemented our protocol in C++ programming language using Pari C library (version 2.9.1) [38] and ran on a computer with Intel® Core i5-4590 CPU@3.30 GHz and 4 GB RAM in Linux environment.

9.5. Experiments

Here we show the experimental results regarding the SPM-RW protocols for searching DNA sequences and the SPM-CW protocol for processing PDBQ.

9.5.1. Results of secure pattern matching with repetitive wildcards

In the settings mentioned earlier, we implemented both of our protocols using our method and Yasuda et al.'s method [6] to solve the SPM-RW problem and compare their performances as shown in Tables 3 and 4. Here we computed the total time for both methods in milliseconds (ms) required by the protocols including key generation, encryption, query transmission, pattern matching, result transmission, and decryption. In addition, we show the time of homomorphic operations in milliseconds, that is required by Bob in the cloud for both protocols. Furthermore, we considered six different settings (l, k) for the experiments of both protocols such that they satisfy $n \geq (k+1)l$. We demonstrate the comparative experimental results of the SPM-RW protocol using the symmetric encryption in Table 3. When we set $l = 512$, $k = 2$, and $|\bar{P}| = 6$, our method took about 62 ms (60 ms for homomorphic operations), whereas the method of Yasuda et al. took 125 ms (110 ms for homomorphic operations) for total pattern matching. In addition, our method took about 78 ms (63 ms for homomorphic operations), whereas the method of Yasuda et al. took 204 ms (141 ms for homomorphic operations) for pattern matching with the settings $l = 512$, $k = 3$, and $|\bar{P}| = 9$. Moreover, we present the experimental results for the settings $(l = 256, k = 5, |\bar{P}| = 15)$, $(l = 256, k = 7, |\bar{P}| = 21)$, $(l = 128, k = 11, |\bar{P}| = 33)$, and $(l = 128, k = 15, |\bar{P}| = 45)$ in Table 3. For every setting, we show the performance improvement factors of the homomorphic operations between our method and Yasuda et al.'s method. Improvement factors are 1.83, 2.24, 5.48, 6.77, 7.58, and 9.58 when the values of k are 2, 3, 5, 7, 11, and 15 respectively. We observe that the factors are mostly near about k highlighted by boldface column in Table 3 except the cases $k > 10$. Therefore, our method is about k times faster than that of Yasuda et al. [6] for performing homomorphic operations. For our protocol, we measured the timings of pattern encryption, key generation, and decryption at the client which are very small (sometimes ≈ 0 ms) as compared to pattern matching time of our method. Consequently, we do not mention the timings separately in our experimental results. Furthermore, we need to achieve more than 80-bit security for protecting the protocols from some distinguishing attacks. As shown in Table 3, we achieve the security level of 234 bits for our parameter settings and root Hermite factor $\delta = 1.00363 \leq 1.0050$. Therefore, our settings are able to provide security from some distinguishing attacks.

In Table 4, we show the comparative results of the SPM-RW protocol using the public-key encryption. The total time taken by Yasuda et al.'s method [6] was 126 ms (62 ms for homomorphic operations), whereas our method took only 78 ms (31 ms for homomorphic operations) to match 2 sub-patterns of length 6 with our text length 512 as shown in Table 4. For the settings $l = 512$, $k = 3$, and $|\bar{P}| = 9$, the method of Yasuda et al. took 187 ms (95 ms for homomorphic operations), whereas our method took 78 ms (31 ms for homomorphic operations). In addition, we show more results for the settings $(l = 256, k = 5, |\bar{P}| = 15)$, $(l = 256, k = 7, |\bar{P}| = 21)$, $(l = 128, k = 11, |\bar{P}| = 33)$, and $(l = 128, k = 15, |\bar{P}| = 45)$ in Table 4. For every setting, the performance improvement factor of homomorphic operations is shown by the boldface column in Table 4. Improvement factors are 2, 3.06, 4.55, 6.88, 10.58, and 12.66 when the values of k are set to 2, 3, 5, 7, 11, and 15 respectively. It is easy to see that the factors are mostly near around k. Therefore, our method is near about k times as fast as the method of Yasuda et al. [6] for performing homomorphic operations. As shown in Table 4, we achieve the security level of 140 bits for our different parameter settings along with root Hermite factor $\delta = 1.00498 \leq 1.0050$. As a result, our protocol is safe from some distinguishing attacks. For this protocol, we measured the encryption time of text which varies between 15 and 16 ms. Here we also measured the times taken by key generation and decryption for every parameter setting of our method which are near about 0 ms for most of the cases, that is not mentioned in experimental results.

Remark 6. We observe that both of our SPM-RW protocols are near about k-times as fast as the method in [6] for performing homomorphic operations. In addition, our code is not fully optimized, and the system has a low configuration as compared to a current high-performance machine. Consequently, an optimized code running on a highly configured machine can produce better results than our results.

9.5.2. Results of secure pattern matching with compound wildcards

To evaluate the SPM-CW protocol, we compare our results with that Kim et al. [5] for processing PDBQ with compound wildcards. We took the same database settings as Kim et al. took. Here we match a single pattern of length 10 with compound wildcards with a set of texts. Therefore, we made three datasets of size 51, 40, and 32 in which the length of each text is 35, 45, and 55 bytes respectively. The performances of the SPM-CW protocol for PDBQ processing are shown in Table 5. For instance, to find a pattern with compound wildcards in the 51 texts of length 35 bytes each, we required 2.590 s, whereas Kim et al. took 100.91 s for doing the same pattern matching. Furthermore, to handle a data set of 40 texts with 45 bytes each, our method required 2.605 s, whereas Kim et al. required 120.96 s for the same pattern matching with the same settings. Also, to process a data set of 32 texts with 55 bytes each, our method required 2.605 s, whereas Kim et al. required 153.21 s for the same settings. Besides, we show amortized

Table 4
Experimental comparison for public-key SPM-RW protocol.

Text length (l)	No. of sub-patterns (k)	Pattern length $\lvert\bar{P}\rvert$	Total time (ms[a])		Time of Hom.Op.[b] (ms[a])		Improvement factor (Hom.Op.[b])	Security level $\log(t_{Adv})$
			Yasuda et al.'s method [6]	Our method	Yasuda et al.'s method [6]	Our method		
512	2	6	126	78	62	31	**2.00**	140
	3	9	187	78	95	31	**3.06**	
256	5	15	297	62	141	31	**4.55**	
	7	21	407	63	220	32	**6.88**	
128	11	33	641	62	328	31	**10.58**	
	15	45	860	62	405	32	**12.66**	

[a]ms: milliseconds.
[b]Hom.Op: Homomorphic operations.

Table 5
Experimental comparison for public-key SPM-CW protocol for processing PDBQ.

String length (Bytes)	Number of string	Total time (s)		Amortized time (s)		Security level	
		Kim et al. [5][a]	Our method[b]	Kim et al. [5][a]	Our method[b]	Kim et al. [5]	Our method
35	51	100.91	2.590	1.98	0.051		
45	40	120.96	2.605	3.02	0.065	83-bit	92-bit
55	32	153.21	2.605	4.78	0.081		

[a]Experiment platform: a local server equipped with Intel® Xeon E5-2620 CPU@2.40 GHz with 24 threads and 64 GB RAM.
[b]Experiment platform: a desktop with Intel® Core i5-4590 CPU@3.30 GHz and 4 GB RAM.

time[2] for the comparison. For our three settings as mentioned earlier, the amortized time of our method is as low as 0.051 s and as high as 0.081 s for the SPM-CW protocol, whereas the amortized time of the method of Kim et al. required as low as 1.98 s and as high as 4.78 s for the pattern matching. In addition, we achieve a better security level of 92 bits than that of Kim et al. of 83 bits as shown in Table 5. From these performances, it is obvious that our method is practical to solve the SPM-CW problem.[3]

10. Conclusions

In this paper, we have discussed two problems of pattern matching: the SPM-RW problem and the SPM-CW problem. For the SPM-RW problem, we propose two efficient protocols using symmetric and public-key SwHE respectively in the semi-honest model and a packing method to achieve the efficiency of the two protocols. In addition, we applied our packing method in pattern matching to non-binary vectors. The packing method is also applicable to a binary vector with multiple queries. Here our packing method enables us to perform homomorphic multiplications about k times faster than that in [6]. Through our protocols, we have been able to show the repetitive-wildcards pattern matching in a precise manner than [6]. For the SPM-CW problem, we propose another efficient protocol using the double-query technique with public-key SwHE in the semi-honest model. For the SPM-CW protocol, our experiments for the PDBQ processing exhibit the performances that are practically usable. Furthermore, our packing method is not only applicable to the SPM but also applicable to other secure computation fields. We surmise that our experimental results and packing method will inspire future researchers to perform the large polynomial computation in a few multiplications. In addition, we consider only three types of wildcard symbols "$", "*", and "!" for our pattern matching. If we want to support other symbols, it will increase time complexity of the method. There is a scope to improve our technique of the SPM-RW protocols in large settings, whereas we have shown our experimental in small settings. Note that our protocols do not provide security against malicious adversaries dealt with the more real-world scenario, which we keep as our future work.

[2] Amortized time is defined by the time required for a single pattern matching per record in the database.

[3] Here we compare the two performances in dissimilar configuration of two PCs between two experiments, in which our PC has lower configuration than that of Kim et al. [5] (See Table 5 for the details). Their test platform is a local server with LAN and all the algorithms are performed on the server [39]. Moreover, they did not consider the time required to send ciphertexts through a network [39] similar to our experiments.

CRediT authorship contribution statement

Tushar Kanti Saha: Conceptualization, Methodology, Software, Formal analysis, Investigation, Resources, Data curation, Validation, Writing - original draft, Visualization. **Deevashwer Rathee:** Software, Formal analysis, Investigation, Data curation. **Takeshi Koshiba:** Supervision, Writing - review & editing, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is supported in part by JSPS Grant-in-Aids for Scientific Research (A) JP16H01705 and for Scientific Research (B) JP17H01695. The authors would like to thank Masaya Yasuda and many anonymous reviewers for their helpful comments which helps us to improve the presentation of the paper.

References

[1] Clancy TC, Kiyavash N, Lin DJ. Secure smartcard based fingerprint authentication. In: Proceedings of the 2003 ACM SIGMM workshop on biometrics methods and applications. ACM Press; 2003, p. 45–52.
[2] Yasuda M, Shimoyama T, Kogure J, Yokoyama K, Koshiba T. Secure pattern matching using somewhat homomorphic encryption. In: ACM workshop on cloud computing security workshop, CCSW 2013. ACM Press; 2013, p. 65–76.
[3] Troncoso-Pastoriza JR, Katzenbeisser S, Celik M. Privacy preserving error resilient DNA searching through oblivious automata. In: Proceedings of the 14th ACM conference on computer and communications security. ACM Press; 2007, p. 519–28.
[4] Defrawy EK, Faber S. Blindfolded Data Search via Secure Pattern Matching. Computer 2013;46(12):68–75, IEEE.
[5] Kim M, Lee HT, Ling S, Tan BHM, Wang H. Private compound wildcard queries using fully homomorphic encryption. IEEE Trans Dependable Secure Comput 2017;16(5):743–56. http://dx.doi.org/10.1109/TDSC.2017.2763593.

[6] Yasuda M, Shimoyama T, Kogure J, Yokoyama K, Koshiba T. Privacy-preserving wildcards pattern matching using symmetric somewhat homomorphic encryption. In: Susilo W, Mu Y, editors. ACISP 2014. LNCS, vol. 5844, Switzerland: Springer; 2014, p. 338–53.
[7] Baron J, Defrawy EK, Minkovich K, Ostrovsky R, Tressler E. 5pm: Secure pattern matching. In: Security and cryptography for networks 2012. LNCS, vol. 7485, Heidelberg: Springer; 2012, p. 222–40.
[8] Blanton M, Aliasgari M. Secure outsourcing of DNA searching via finite automata. In: Foresti S, Jajodia S, editors. Data and applications security XXIV. LNCS, vol. 6166, Heidelberg: Springer; 2010, p. 49–64.
[9] Hazay C, Toft T. Computationally secure pattern matching in the presence of malicious adversaries. J Cryptol 2014;27(2):358–95.
[10] Frikken KB. Practical private DNA string searching and matching through efficient oblivious automata evaluation. In: Data and applications security 2009. LNCS, vol. 5645, Heidelberg: Springer; 2009, p. 81–94.
[11] Chase M, Shen E. Substring-searchable symmetric encryption. PoPETs 2015;2015(2):263–81.
[12] Faber S, Jarecki S, Krawczyk H, Nguyen Q, Rosu M, Steiner M. Rich queries on encrypted data: Beyond exact matches. In: European symposium on research in computer security —ESORICS 2015 part II. LNCS, vol. 9327, Cham: Springer; 2015, p. 123–45.
[13] Goldwasser S, Micali S. Probabilistic encryption & how to play mental poker keeping secret all partial information. In: Proceedings of the fourteenth annual ACM symposium on theory of computing. ACM Press; 1982, p. 365–77.
[14] ElGamal T. A public key cryptosystem and a signature scheme based on discrete logarithms. In: Advances in cryptology, vol. 196, Heidelberg: Springer; 1985, p. 10–8.
[15] Cohen JD, Fischer MJ. A robust and verifiable cryptographically secure election scheme. In: 26th annual symposium on foundations of computer science, 1985. IEEE; 1985, p. 372–82.
[16] Paillier P. Public-key cryptosystems based on composite degree residuosity classes. In: Advances in cryptology—EUROCRYPT'99. LNCS, vol. 1592, Heidelberg: Springer; 1999, p. 223–38.
[17] Boneh D, Goh EJ, Nissim K. Evaluating 2-DNF formulas on ciphertexts. In: Theory of cryptography—TCC 2005. LCNS, vol. 3378, Springer; 2005, p. 325–41.
[18] Gentry C. Fully homomorphic encryption using ideal lattices. In: Symposium on theory of computing — STOC 2009. ACM Press; 2009, p. 169–78.
[19] Hu Y. Improving the efficiency of homomorphic encryption schemes (PhD diss.), Worcester Polytechnic Institute, Massachusetts; 2013.
[20] Brakerski Z, Vaikuntanathan V. Fully homomorphic encryption from Ring-LWE and security for key dependent messages. In: Rogaway P, editor. CRYPTO 2011. LNCS, vol. 6841, Heidelberg: Springer; 2011, p. 505–24.
[21] Lauter K, Naehrig M, Vaikuntanathan V. Can homomorphic encryption be practical?. In: ACM workshop on cloud computing security workshop, CCSW 2011. ACM Press; 2011, p. 113–24.
[22] Yasuda M, Shimoyama T, Kogure J, Yokoyama K, Koshiba T. Secure statistical analysis using RLWE-based homomorphic encryption. In: Information security and privacy. Switzerland: Springer; 2015, p. 471–87.
[23] Saha TK, Koshiba T. Outsourcing private equality tests to the cloud. J Inf Secur Appl 2018;43:83–98.
[24] Saha TK, Koshiba T. An enhancement of privacy-preserving wildcards pattern matching. In: F. Cuppens, Wang L, Cuppens-Boulahia N, Tawbi N, Garcia-Alfaro J, editors. Foundations and practice of security. FPS 2016. LNCS, vol. 10128, Cham: Springer; 2017, p. 145–60.
[25] Yao AC. Protocols for secure computations. In: 23rd annual symposium on foundations of computer science. IEEE; 1982, p. 160–4.
[26] Jha S, Kruger L, Shmatikov V. Towards practical privacy for genomic computation. In: IEEE symposium on security and privacy. IEEE; 2008, p. 216–30.
[27] Katz J, Malka L. Secure text processing with applications to Private DNA Matching. In: Proceedings of the 17th ACM conference on computer and communications security. ACM Press; 2010, p. 485–92.
[28] Beck M, Kerschbaum F. Approximate two-party privacy-preserving string matching with linear complexity. In: IEEE international congress on big data. IEEE; 2013, p. 31–7.
[29] Lyubashevsky V, Peikert C, Regev O. On ideal lattices and learning with errors over rings. In: Gilbert H, editor. EUROCRYPT 2010. LNCS, vol. 6110, Heidelberg: Springer; 2010, p. 1–23.
[30] Yasuda M, Shimoyama T, Kogure J, Yokoyama K, Koshiba T. Practical packing method in somewhat homomorphic encryption. In: DPM 2013 and SETOP 2013. LNCS, vol. 8247, Heidelberg: Springer; 2014, p. 34–50.
[31] Smart NP, Vercauteren F. Fully Homomorphic SIMD operations. Des Codes Cryptogr 2014;71(1):57–81. http://dx.doi.org/10.1007/s10623-012-9720-4.
[32] Rivest RL, Adleman L, Dertouzos ML. On data banks and privacy homomorphism. In: Foundations of secure computation. Academia Press; 1978, p. 169–77.
[33] Goldreich O, Micali S, Wigderson A. How to play any mental game. In: Proceedings of the nineteenth annual ACM symposium on theory of computing. ACM Press; 1987, p. 218–29.
[34] Ishai Y, Prabhakaran M, Sahai A. Founding cryptography on oblivious transfer – efficiently. In: Advances in cryptology – CRYPTO 2008. CRYPTO 2008. LNCS, vol. 5157, Heidelberg: Springer; 2008, p. 572–91.
[35] Lindner R, Peikert C. Better key sizes (and attacks) for LWE-based encryption. In: Topics in cryptology — CT-RSA 2011, vol. 6558, Heidelberg: Springer; 2011, p. 319–39.
[36] Micciancio D, Regev O. Lattice-based cryptography. In: Post-quantum cryptography. Heidelberg: Springer; 2009, p. 147–91.
[37] Chen Y, Nguyen PQ. BKZ 2.0: Better lattice security estimates. In: Advances in cryptology — ASIACRYPT 2011. LNCS, vol. 7073, Heidelberg: Springer; 2011, p. 1–20.
[38] PARI/GP version 291. The PARI Group; Univ. Bordeaux. 2018, Available from http://pari.math.u-bordeaux.fr/.
[39] Tan BHM. Private communication.