Efficient Homomorphic Encryption on Integer Vectors and Its Applications

Hongchao Zhou and Gregory Wornell
Dept. Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139
Email: {hongchao,gww}@mit.edu

***Abstract*—Homomorphic encryption, aimed at enabling computation in the encrypted domain, is becoming important to a wide and growing range of applications, from cloud computing to distributed sensing. In recent years, a number of approaches to fully (or nearly fully) homomorphic encryption have been proposed, but to date the space and time complexity of the associated schemes has precluded their use in practice. In this work, we demonstrate that more practical homomorphic encryption schemes are possible when we require that not all encrypted computations be supported, but rather only those of interest to the target application. More specifically, we develop a homomorphic encryption scheme operating directly on integer vectors that supports three operations of fundamental interest in signal processing applications: addition, linear transformation, and weighted inner products. Moreover, when used in combination, these primatives allow us to efficiently and securely compute arbitrary polynomials. Some practically relevant examples of the computations supported by this framework are described, including feature extraction, recognition, classification, and data aggregation.**

I. Introduction

Homomorphic encryption techniques hold the promise of enabling truly secure ways to gather, share, and process information in distributed settings. For instance, in data cloud applications, users typically want to keep their data private while still allowing remote servers to perform tasks such as filtering or search. As another example, in biometric authentication applications, users want to successfully authenticate with remote servers while maintaining the privacy of their requests. Finally, in remote and distributed sensing applications, there is a need to be able fuse and process acquired data directly in the encrypted domain, sometimes while further disguising the nature of such processing.

The notion of homomorphic encryption—encryption that supports computation on encrypted data—dates back to late 1970's [1]. Later, homomorphic encryption schemes that support either addition or multiplication operations (but not both) were proposed by, e.g., Goldwasser and Micali [2], El-Gamal [3], and Paillier [4]. A key breakthrough was the relatively recent development by Gentry of a fully homomorphic encryption (FHE) scheme [5], which is capable of evaluating an arbitrary function in the encrypted domain. Since then, several variants and improvements on Gentry's method have been developed; see, e.g., [6]–[14]. Among them, the Brakerski-Gentry-Vaikuntanathan (BGV) scheme [9] is among the most promising (somewhat) FHE scheme. However, it is still rather impractical due to its high computational complexity and large communication cost [15].

Several aspects of current homomorphic encryption schemes make them difficult to realize in practice. First, most homomorphic encryption schemes are bit-by-bit encryption schemes, i.e., each ciphertext represents a single bit. The advantage of such a bit-by-bit encryption scheme is that it is easy to be fully homomorphic: if the scheme supports both binary addition and multiplication operations on encrypted bits, then it can compute an arbitrary Boolean function in the encrypted domain. This is because any Boolean function can be represented by a collection of binary addition and multiplication operations. At the same time, however, bit-by-bit encryption makes the practical applications of homomorphic encryption problematic, since it significantly reduces the storage and communication efficiency and increases the computational time. For example, in data-cloud applications, where users store their private data in remote servers, it is unacceptably inefficient to store each single bit with an entire ciphertext of thousands (or more) bits. Furthermore, bit-by-bit encryption schemes require applications to convert computation tasks into binary addition and multiplication operations, which makes the computation more complex—even for simple elementary operations such as the multiplication of two 8-bit integers, its execution demands hundreds of bit operations with multiplication depth of 16 [15]. And we know that, for homomorphic encryption, it is much more difficult to handle multiplications than additions, especially when the multiplication depth is large.

In order to reduce the complexity of homomorphic computation, several ciphertext-packing techniques have been developed to combine multiple ciphertexts into a single ciphertext. In [16], Smart and Vercauteren develop a technique to pack ciphertexts based on polynomial-CRT. In [17], Yasuda, et al., develop a technique to pack ciphertexts based on ideal lattices and apply it to biometrics. In [18], Cheon, et al., extend the scheme of van Dijk, et al., [12] to support encrypting and processing a binary vector. In [11], Naehrig, Lauter, and Vaikuntanathan present a ciphertext-packing technique whose security is based on the Ring Learning With Errors (Ring

This work was supported in part by Draper Laboratory through the UR&D Program and by AFOSR under Grant No. FA9550-11-1-0183.

LWE) problem. A similar ciphertext-packing technique whose security is based on the Learning With Error (LWE) problem, was introduced by Peikert, Vaikuntanathan and Waters (PVW) [19]. Later, Brakerski, Gentry and Halevi [20] showed how to apply the PVW technique to perform SIMD-type (Single Instruction Multiple Data) operations, and their goal was trying to parallelize any given repeated operations, by encrypting multiple bits within the same ciphertext and performing the same operations on these bits. However, these techniques have their application limitations, and they still need to decompose computation tasks into many binary operations. Even with these techniques, the computation and communication requirements are still difficult to accommodate in most practical applications.

In this paper, we approach the problem from a somewhat different perspective. In particular, while almost all the current homomorphic encryption schemes are aimed at enabling arbitrary computation tasks so as to be "universal," we instead restrict our attention to important and broadly useful—but more limited—classes of computation tasks in order to obtain schemes more amenable to implementation in practice. More specifically, we develop an efficient homomorphic encryption scheme that encrypts data directly in the form integer vectors, and supports practically important forms of computation on such data in the encrypted domain. Without the need to decompose the processing of such data into binary operations, the computational complexity is significantly reduced.

The paper is organized as follows. Section II describes the scenario of interest with applications in cloud storage and sensing systems. Section III presents a scheme that encrypts integer vectors directly, as a natural extension of the recently developed homomorphic encryption schemes based on the learning with errors (LWE) formalism. In Section IV, we demonstrate that this scheme supports three types of fundamental operations on integer vectors in the encrypted domain: addition, linear transformation, and weighted inner products. Used in combination, these allow us to efficiently compute an arbitrary polynomial in a secure manner. Section V provides examples of the kinds of practical computations possible with this scheme, including feature extraction, recognition, classification, and data aggregation. Finally, Section VI contains some concluding remarks.

II. Application Scenario

In this section, we introduce a scenario that has important applications in cloud storage and sensing systems. However, before that, we first describe the different but more traditional scenario that has been the focus of much of the homomorphic encryption literature to date. As depicted in Fig. 1, in this traditional scenario there are two parties involved in a computational procedure process. The user (party B) sends an encrypted request $\mathbf{x}$, which is some privacy-sensitive information like biometrics or medical records, to the server (Party A). The server has the processing algorithm f, but cannot or does not want to disclose f to the user due to some

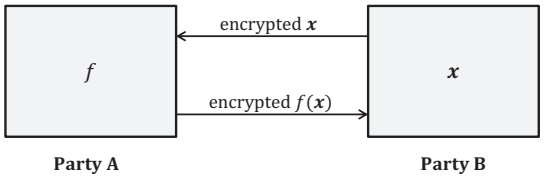

Fig. 1. A homomorphic-encryption scenario widely studied in literatures.

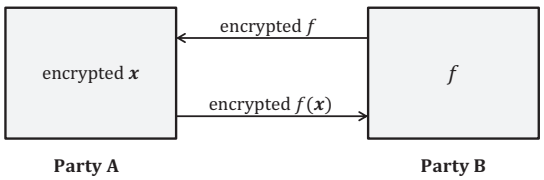

Fig. 2. A homomorphic-encryption scenario considering in the current paper.

computational or privacy reasons. As a result, the server computes $f(\mathbf{x})$ in the encrypted domain, and returns the encrypted answer to the user. This scenario has promising applications in search engines, biometric authentication, privacy-protected face recognition, etc. [21]

By contrast, in this paper our interest is in the scenario depicted in Fig. 2. In this scenario, party A receives or stores the encryption of a large amount of data $\mathbf{x}$, which is confidential and can be decrypted only by party B, e.g., $\mathbf{x}$ is a user's private data stored in data clouds in the encrypted form. Some time, Party B wants to get some information $f(\mathbf{x})$ from $\mathbf{x}$ without revealing the function f to party A, e.g., performing search. A trivial approach is that party A sends the whole encryption of $\mathbf{x}$ back to party B directly, and party B evaluates the function f based on the received encryption of $\mathbf{x}$. However, this approach is not wise in most applications, since the data $\mathbf{x}$ is large and the communication between the two parties is not cheap. Hence, we consider the following approach: party B sends an encrypted version of f to party A, and party A performs certain processes (without knowing $\mathbf{x}$ and f) and returns a small amount of encrypted data back to party B, from which party B further obtains the result $f(\mathbf{x})$.

An important difference between the scenario depicted in Fig. 2 and that in Fig. 1 is that, in the new scenario, the processing function f is known by party B, which also owns the secret key. This property allows us to directly perform operations on integer vectors in the encrypted domain, such as linear transformations and weighted inner products (see Section IV), but these operations are much more difficult to implement in the scenario of Fig. 1, where the processing function f and the secret key are separated. There is another benefit of the property: if $f(\mathbf{x})$ contains some operations that are difficult to realize in the encrypted domain, these operations can be executed at the side of party B. For example, assume that $f(\mathbf{x})$ is a linear classifier such that $f(\mathbf{x}) = 0$ if and only if $\mathbf{w} \cdot \mathbf{x} < t$ for a constant t; otherwise, $f(\mathbf{x}) = 1$. In this case, $\mathbf{w} \cdot \mathbf{x}$ can be computed in the encrypted domain

at the party A side, and the comparison between $\mathbf{w} \cdot \mathbf{x}$ and t can be performed at the party B side.

The scenario of Fig. 2 is applicable to a wide range of contexts involving data clouds and sensor networks. Depending on specific applications, there are a few factors that should be considered. They include: 1) the privacy requirement of $\mathbf{x}$ and f, namely, how secure the encryptions of $\mathbf{x}$ and f are; 2) the storage cost of $\mathbf{x}$, i.e., the number of encrypted bits required to represent $\mathbf{x}$; 3) the computation cost of f, i.e, the amount of computations needed at party A and party B for computing $f(\mathbf{x})$ in the encrypted domain; and 4) the communication cost, i.e., the number of transmitted bits between party A and party B. In what follows, we describe several application examples of the scenario.

A. Encrypted Cloud Storage

A major concern of users when weighing the adoption of cloud data storage solutions, such as provided by companies such as Dropbox, Google, and Microsoft, is the loss of privacy of their data. Homomorphic encryption can help users preserve the privacy while allowing them to perform certain computations on their own data or to retrieve some useful information. Here, we treat party A as the server and party B as the end user, where $\mathbf{x}$ is the user's private data stored at the server in the encrypted form. The user may want to get some information from $\mathbf{x}$, e.g., searching with a private query string.

B. Medical Records Storage

One example of encrypted cloud storage is the private cloud medical records storage system discussed in [11]. All data for a patient's medial records stored in the system is encrypted by the healthcare providers. The patient can share or access the records by sharing the secret key with trusted providers. While the amount of medical records for a patient is usually big, the trusted providers may only want to get some useful information extracted from the patient's records. In this case, the storage system is party A, and the trusted providers are party B in Fig. 2.

C. Sensor Networks

In sensor networks or other sensing systems, especially those related to military applications, sensed data is expected to be protected (encrypted) immediately when it is generated at sensors, and the base station is interested in only a small amount of information from the sensed data, e.g., some features of a sensed image or some statistics about the data. Hence, certain computations are required to perform at the sensors and relay nodes for data processing and aggregation to reduce the communication load. Here, the sensors and relay nodes are the party A, and the base station is the party B. This problem, secure aggregation in sensor networks, has been studied in [22], but their adopted encryption scheme only supports addition operations, which limits its applications in sensor networks. In contrast, the encryption scheme studied in the current paper can support much wider computation tasks.

III. Encryption Scheme

A. Encryption Scheme

The encryption scheme that we study is a natural extension of the PVW scheme [19] from binary vectors to integer vectors. We let $\mathbf{x} \in \mathbb{Z}_p^m$ be the integer vector to encrypt, and it has length m and alphabet size p. Let $\mathbf{c} \in \mathbb{Z}_q^n$ be the ciphertext of $\mathbf{x}$ with length $n > m$ and alphabet size $q \gg p$. Typically, q is super-polynomial in the ciphertext length n. The secret key is a matrix $\mathbf{S} \in \mathbb{Z}_q^{m \times n}$, and it satisfies

$$\mathbf{Sc} = q\mathbf{k} + w\mathbf{x} + \mathbf{e}, \tag{1}$$

for some integer vector $\mathbf{k}$ and noise vector $\mathbf{e}$. Here, w is an integer parameter such that $w > 2|\mathbf{e}|$. We call the maximum absolute value of the entries in a vector $\mathbf{v}$ or a matrix $\mathbf{M}$ as its magnitude, denoted by $|\mathbf{v}|$ or $|\mathbf{M}|$.

The process of encrypting $\mathbf{x}$ is to find a ciphertext $\mathbf{c}$ such that $\mathbf{Sc}$ satisfies (1) for some integer vector $\mathbf{k}$ and noise vector $\mathbf{e}$. For the convenience of description, we first present the decryption process, as follows, and then present the public-key encryption process in Section III-C.

According to (1), decryption of ciphertext $\mathbf{c}$ based on the secret key $\mathbf{S}$ is done by computing

$$\mathbf{x} = \lceil \frac{\mathbf{Sc}}{w} \rfloor_q, \tag{2}$$

where $\lceil a \rfloor_q$ means the nearest integer to a with modulus q. The decryption succeeds if the magnitude of $\mathbf{e}$, i.e., $|\mathbf{e}|$, is smaller than $\frac{w}{2}$. In order to further support computations in the encrypted domain, we assume that both $|\mathbf{S}|$ and $|\mathbf{e}|$ are much smaller than w.

B. Key-Switching Technique

In [7], Brakerski and Vaikuntanathan introduced a very useful re-linearization technique that can switch the secret key in the PVM scheme to any other secret key when they are both vectors. Later, Brakerski, Gentry and Halevi developed a technique to switch two secret keys of matrices [20]. In general, Brakerski and Vaikuntanathan's re-linearization method has two steps, and we apply it to switch a secret key $\mathbf{S} \in \mathbb{Z}_q^{m \times n}$ to another secret key $\mathbf{S}' \in \mathbb{Z}_q^{m \times n'}$ as follows, and meanwhile, we get a new ciphertext $\mathbf{c}'$ that still encrypts the same integer vector $\mathbf{x}$.

1) Step 1: Switch the secret key $\mathbf{S}$ to an intermediate secret key $\mathbf{S}^*$ such that its corresponding new ciphertext $\mathbf{c}^*$ has a much smaller magnitude than $\mathbf{c}$. The idea is to represent each element c_i in $\mathbf{c}$ with a binary vector (its binary representation), hence it results in a new ciphertext $\mathbf{c}^*$ with $|\mathbf{c}^*| = 1$. Assume that $c_i = b_{i0} + b_{i1}2 + ... + b_{i(\ell-1)}2^{\ell-1}$, then by writing each c_i as $[b_{i0}, b_{i1}, ..., b_{i(\ell-1)}]^T$, we get $\mathbf{c}^*$. For instance, given a ciphertext $\mathbf{c} = [1, 5]^T \in \mathbb{Z}_8^2$, we convert it to $\mathbf{c}^* = [1, 0, 0, 1, 0, 1]^T$. Now, we construct a secret key $\mathbf{S}^* \in \mathbb{Z}^{m \times n\ell}$ such that

$$\mathbf{S}^*\mathbf{c}^* = \mathbf{Sc}. \tag{3}$$

This can be done by replacing each element S_{ij} in $\mathbf{S}$ with a vector $[S_{ij}, S_{ij}2, ..., S_{ij}2^{\ell-1}]$. For example, if the original

secret key is $\mathbf{S} = [4, 3; 1, 2] \in \mathbb{Z}_8^{2\times 2}$, then the new secret key is $\mathbf{S}^* = [4, 8, 16, 3, 6, 12; 1, 2, 4, 2, 4, 8]$. It is easy to check that, in the above example, $\mathbf{S}^*\mathbf{c}^* = \mathbf{S}\mathbf{c}$ is satisfied.

2) Step 2: Switch the intermediate secret key $\mathbf{S}^* \in \mathbb{Z}^{m\times n\ell}$ to the desired secret key $\mathbf{S}' \in \mathbb{Z}_q^{m\times n'}$. In order to do this, we construct an integer matrix $\mathbf{M} \in \mathbb{Z}^{n'\times n\ell}$ such that

$$\mathbf{S}'\mathbf{M} = \mathbf{S}^* + \mathbf{E} \mod q \tag{4}$$

for a noise matrix $\mathbf{E}$ with small magnitude. If $\mathbf{S}' = [\mathbf{I}, \mathbf{T}]$ with an identity matrix $\mathbf{I}$, this integer matrix $\mathbf{M}$ can be constructed by

$$\mathbf{M} = \begin{pmatrix} -\mathbf{T}\mathbf{A} + \mathbf{S}^* + \mathbf{E} \\ \mathbf{A} \end{pmatrix} \mod q, \tag{5}$$

where $\mathbf{A} \in \mathbb{Z}_q^{(n'-m)\times n\ell}$ is a random matrix.

Then we define a new ciphertext

$$\mathbf{c}' = \mathbf{M}\mathbf{c}^* \mod q, \tag{6}$$

which can be further written as

$$\mathbf{c}' = q\mathbf{k}^* + \begin{pmatrix} -\mathbf{T}\mathbf{A}\mathbf{c}^* + \mathbf{S}^*\mathbf{c}^* + \mathbf{E}\mathbf{c}^* \\ \mathbf{A}\mathbf{c}^* \end{pmatrix}, \tag{7}$$

where $\mathbf{k}^*$ is an integer vector, and $|\mathbf{k}^*|$ is much smaller than q when $|\mathbf{T}|$ and $|\mathbf{S}|$ are much smaller than q.

For this new group of secret key $\mathbf{S}'$ and ciphertext $\mathbf{c}'$, they satisfy

$$\mathbf{S}'\mathbf{c}' = q\mathbf{S}'\mathbf{k}^* + \mathbf{S}\mathbf{c} + \mathbf{E}\mathbf{e}^* = q\mathbf{k}' + w\mathbf{x} + \mathbf{e}', \tag{8}$$

with

$$\mathbf{k}' = \mathbf{k} + \mathbf{S}'\mathbf{k}^*, \quad \mathbf{e}' = \mathbf{e} + \mathbf{E}\mathbf{c}^*. \tag{9}$$

Both $\mathbf{k}$ and $\mathbf{e}$ are specified in (1), and $\mathbf{E}\mathbf{c}^*$ is the additional noise vector introduced by the key-switching process. Since $|\mathbf{S}'|$, $|\mathbf{k}^*|$, $|\mathbf{e}|$, $|\mathbf{E}|$, and $|\mathbf{c}^*|$ are much smaller than q, $|\mathbf{k}'|$ and $|\mathbf{e}'|$ are also much smaller than q. We can correctly obtain $\mathbf{x}$ by decrypting $\mathbf{c}'$ with the new secret key $\mathbf{S}'$. Therefore, we have successfully switched the secret key $\mathbf{S}$ to a new secret key $\mathbf{S}'$, and meanwhile the original ciphertext $\mathbf{c}$ is converted to a new ciphertext $\mathbf{c}'$.

To implement the key-switching technique in the scenario depicted in Fig. 2. Party B first generates the key-switching matrix $\mathbf{M}$ based on the two secret keys $\mathbf{S}$ and $\mathbf{S}'$, and then sends $\mathbf{M}$ to party A. With the received matrix $\mathbf{M}$, party A computes the new ciphertext $\mathbf{c}'$ based on (6). We see that the key-switching matrix forms a public key. In fact, this key-switching technique plays an important role in both the encryption process and the computation process, which will be further discussed.

C. Security and Encryption

The security of the encryption relies on the security of the key-switching matrix (matrices) $\mathbf{M}$, which is based on the hardness assumption of the extended Learning With Error (LWE) problem: It is difficult to get $\mathbf{S}'$ from $\mathbf{S}$ and $\mathbf{M}$ by solving (4), where both $\mathbf{S}'$ and $\mathbf{E}$ are random matrices with elements independently drawn from a noise distribution χ on $\mathbb{Z}_q$. In fact, this problem is equivalently difficult as the standard LWE problem [20], [23], which tries to get a vector $\mathbf{s}'$ from $\mathbf{s}$ and $\mathbf{M}$ by solving

$$\mathbf{s}'\mathbf{M} = \mathbf{s} + \mathbf{e} \mod q \tag{10}$$

when $\mathbf{s}'$ is a uniform random vector and $\mathbf{e}$ is a noise vector.

The process of encrypting an integer vector $\mathbf{x}$ can be done based on the key-switching technique. Let $\mathbf{I}$ be an $m \times m$ identity matrix, then $\mathbf{I}(w\mathbf{x}) = w\mathbf{x}$, which is actually in the form of (1) with a zero noise vector. In a sense, we can treat $\mathbf{I}$ as a secret key and $w\mathbf{x}$ as a ciphertext, although both are not secret, and hence we can switch $\mathbf{I}$ to a secret key $\mathbf{S}$ with the key-switching technique, and by which we get a new ciphertext $\mathbf{c}$ instead of $w\mathbf{x}$. The ciphertext $\mathbf{c}$ is an encryption of $\mathbf{x}$ based on the secret key $\mathbf{S}$. We see that this is a public-key encryption scheme, since we can use the key-switching matrix $\mathbf{M}$ as the public key to generate the ciphertext $\mathbf{c}$, and this public key $\mathbf{M}$ can be constructed based on the secret key $\mathbf{S}$.

IV. Operations on Encrypted Data

In this section, we describe three types of fundamental operations on integer vectors that can be easily performed based on the encryption scheme described above, including addition, multiplication, and weighted inner products. Many practical tasks can be represented by a combination of these three types of fundamental operations. We demonstrate that given the encrypted integer vectors, any polynomial (within a certain degree) on integers can be computed secretly and efficiently in the scenario of Fig. 2.

A. Three Fundamental Operations

Let $\mathbf{x}_1, \mathbf{x}_2$ be two integer vectors, then the three types of fundamental operations are: (1) addition, i.e, $\mathbf{x}_1 + \mathbf{x}_2$, which requires $\mathbf{x}_1$ and $\mathbf{x}_2$ having the same length; (2) linear transformation, i.e., $\mathbf{G}\mathbf{x}_1$ for an arbitrary matrix $\mathbf{G}$, and (3) weighted inner products, i.e., $\{\mathbf{x}_1^T\mathbf{H}_j\mathbf{x}_2\}$ for a group of weight matrices $\{\mathbf{H}_j\}$. Here, we assume that all the values appeared in these operations are between zero and $\lfloor \frac{q}{w} \rfloor$, i.e., there are no integer overflows in our computation.

Let $\mathbf{c}_1, \mathbf{c}_2$ be the two ciphertexts of the integer vectors $\mathbf{x}_1, \mathbf{x}_2$ with secret keys $\mathbf{S}_1, \mathbf{S}_2$, respectively, and they satisfy

$$\mathbf{S}_i\mathbf{c}_i = q\mathbf{k}_i + w\mathbf{x}_i + \mathbf{e}_i, \tag{11}$$

with $|\mathbf{S}_i|$, $|\mathbf{k}_i|$ and $|\mathbf{e}_i|$ much smaller than q. In what follows, we demonstrate how these three types of fundamental operations work.

1) Addition Operation: The addition operation $\mathbf{x}_1 + \mathbf{x}_2$ is straightforward: if $\mathbf{c}_1$ and $\mathbf{c}_2$ have the same secret key, i.e. $\mathbf{S}_1 = \mathbf{S}_2 = \mathbf{S}$, then $\mathbf{c}' = \mathbf{c}_1 + \mathbf{c}_2 \mod q$ is an encryption of $\mathbf{x}_1 + \mathbf{x}_2$, since

$$\mathbf{S}\mathbf{c}' = q\mathbf{k}' + w(\mathbf{x}_1 + \mathbf{x}_2) + (\mathbf{e}_1 + \mathbf{e}_2), \tag{12}$$

where $\mathbf{k}'$ is an integer vector with a small magnitude. However, $\mathbf{x}_1$ and $\mathbf{x}_2$ may have different secret keys, so the first thing that we need to do is to switch one secret key to the other, e.g., from $\mathbf{S}_1$ to $\mathbf{S} = \mathbf{S}_2$. Note that if $\mathbf{S}_1$ and $\mathbf{S}_2$ are correlated, for the purpose of guaranteeing security, we should create a new

secret key $\mathbf{S}$ and switch both $\mathbf{S}_1$ and $\mathbf{S}_2$ to $\mathbf{S}$. We use $\mathbf{c}'_1$ and $\mathbf{c}'_2$ to denote the ciphertexts obtained after key switching, and they satisfy $\mathbf{S}\mathbf{c}'_i = q\mathbf{k}'_i + w\mathbf{x}_i + \mathbf{e}'_i$ with $|\mathbf{k}'_i|$ and $|\mathbf{e}'_i|$ small. If we let $\mathbf{c}' = \mathbf{c}'_1 + \mathbf{c}'_2 \mod q$, then

$$\mathbf{S}\mathbf{c}' = q\mathbf{k}' + w(\mathbf{x}_1 + \mathbf{x}_2) + \mathbf{e}' \tag{13}$$

with $|\mathbf{k}'| \leq |\mathbf{k}'_1| + |\mathbf{k}'_2|$ and $\mathbf{e}' = \mathbf{e}'_1 + \mathbf{e}'_2$. Hence, $\mathbf{c}'$ is a valid encryption of $\mathbf{x}_1 + \mathbf{x}_2$.

There is a special case: if one wants to compute $\mathbf{x}_1 + \mathbf{a}$ with a constant vector $\mathbf{a}$, which is known by public, then a valid encryption of $\mathbf{x}_1 + \mathbf{a}$ is $\mathbf{c} = \mathbf{c}_1 + w[\mathbf{a}^T, 0, ..., 0]^T \mod q$. It works since in our construction the secret key $\mathbf{S}$ is in the form of $[\mathbf{I}, \mathbf{T}]$ with an identity matrix $\mathbf{I}$, and hence $\mathbf{S}\mathbf{c} = \mathbf{S}\mathbf{c}_1 + w\mathbf{a}$.

2) Linear Transformation: The linear transformation $\mathbf{G}\mathbf{x}_1$ follows the observation that

$$\mathbf{G}\mathbf{S}\mathbf{c}_1 = q\mathbf{G}\mathbf{k}_1 + w\mathbf{G}\mathbf{x}_1 + \mathbf{G}\mathbf{e}_1. \tag{14}$$

So if $|\mathbf{G}|$ is much smaller than q, we can treat $\mathbf{c}' = \mathbf{c}_1$ as the ciphertext of $\mathbf{G}\mathbf{x}_1$, with secret key $\mathbf{S}' = \mathbf{G}\mathbf{S}$.

There is a special case: if one wants to compute $a\mathbf{x}_1$ for a small integer a known by public, then its ciphertext is

$$\mathbf{c} = a\mathbf{c}_1 \mod q. \tag{15}$$

3) Weighted Inner Products: A group of weighted inner products $\{\mathbf{x}_1^T\mathbf{H}_j\mathbf{x}_2\}$ can be computed by applying the technique for multiplication via tensor products in [7]. Note that

$$(\mathbf{S}_1\mathbf{c}_1)^T\mathbf{H}_j(\mathbf{S}_2\mathbf{c}_2) = \mathrm{vec}(\mathbf{S}_1^T\mathbf{H}_j\mathbf{S}_2) \cdot \mathrm{vec}(\mathbf{c}_1\mathbf{c}_2^T), \tag{16}$$

where $\mathrm{vec}(\mathbf{A})$ denote the vector that consists of all the entries in a matrix $\mathbf{A}$. Substituting $\mathbf{S}_i\mathbf{c}_i$ with (11) in the left side of the above equation, we get

$$\begin{aligned}&(\mathbf{S}_1\mathbf{c}_1)^T\mathbf{H}_j(\mathbf{S}_2\mathbf{c}_2)\\ =&q(q\mathbf{k}_1^T\mathbf{H}_j\mathbf{k}_2 + w\mathbf{k}_1^T\mathbf{H}_j\mathbf{x}_2 + w\mathbf{x}_1^T\mathbf{H}_j\mathbf{k}_2)\\ &+ w^2(\mathbf{x}_1^T\mathbf{H}_j\mathbf{x}_2) + (q\mathbf{k}_1^T\mathbf{H}_j\mathbf{e}_2 + q\mathbf{e}_1^T\mathbf{H}_j\mathbf{k}_2\\ &+ w\mathbf{x}_1^T\mathbf{H}_j\mathbf{e}_2 + w\mathbf{e}_1^T\mathbf{H}_j\mathbf{x}_2 + \mathbf{e}_1^T\mathbf{H}_j\mathbf{e}_2).\end{aligned} \tag{17}$$

Let $\mathbf{s}'_j = \mathrm{vec}(\mathbf{S}_1^T\mathbf{H}_j\mathbf{S}_2)^T$ be the jth row of the new secret key $\mathbf{S}'$, and let $\mathbf{c}' = \lceil \frac{\mathrm{vec}(\mathbf{c}_1\mathbf{c}_2^T)}{w} \rfloor_q$ be the new ciphertext.

Assume that $q = wl + r$ with an integer l and a very small remainder r. Since $|\mathbf{k}_1|, |\mathbf{k}_2|, |\mathbf{H}_j|, |\mathbf{e}_1|, |\mathbf{e}_2|$ are much smaller than w, we have

$$\mathbf{s}'_j\mathbf{c}' = qk'_j + w(\mathbf{x}_1^T\mathbf{H}_j\mathbf{x}_2) + e'_j, \tag{18}$$

where k'_j is an integer, and k'_j, e'_j are much smaller than w. By decrypting $\mathbf{c}'$ with the secret key $\mathbf{S}'$, we can compute the group of weighted inner products $\{\mathbf{x}_1^T\mathbf{H}_j\mathbf{x}_2\}$.

So far, the dimension of the ciphertext $\mathbf{c}'$ is big: it is the square of the original dimension n. Fortunately, the key-switching technique can be used to reduce this dimension, by switching the secret key $\mathbf{S}'$ to a new secrete key, and meanwhile, the dimension of the ciphertext is reduced to n. As a result, we get the final ciphertext $\mathbf{M}\lceil \frac{\mathrm{vec}(\mathbf{c}_1\mathbf{c}_2^T)}{w} \rfloor_q$, where $\mathbf{M}$ is the key-switching matrix.

We see that the noise magnitude grows much faster in the weighted-inner-products operations than that in the addition and linear-transformation operations. In addition, to perform a weighted-inner-products operation, it takes $O(n^3)$ times; and as a contrast, it only takes $O(n^2)$ time for an addition or linear-transformation operation. So, in practice, we need to minimize the number (in particular the depth) of weighted-inner-products operations in computation.

B. Polynomial Computation

Based on the three types of fundamental operations above, we can compute an arbitrary polynomial (within a certain degree) on integers efficiently. For instance, in order to compute $x_2^2 - 4x_1x_3$ with each variable an integer of 8 bits, the bit-by-bit encryption schemes require 322 binary addition operations and 302 binary multiplication operations [15], with computation depth 43 and multiplication depth 16. However, with the encryption scheme on integer vectors, this function can be computed with a single weighted-inner-products operation $\mathbf{x}^\mathbf{T}\mathbf{H_1}\mathbf{x}$ by simply setting

$$\mathbf{H_1} = \begin{pmatrix} 0 & 0 & -2 \\ 0 & 1 & 0 \\ -2 & 0 & 0 \end{pmatrix}.$$

In fact, an arbitrary degree-2 polynomial can be computed based on a single weighted-inner-products operation. In order to do this, we need to execute the operation on $[1, \mathbf{x}^T]^T$ instead of $\mathbf{x}$, since every degree-2 polynomial can be represented by $[1, \mathbf{x}^T]\mathbf{H}[1, \mathbf{x}^T]^T$ for some matrix $\mathbf{H}$. If the ciphertext of $\mathbf{x}$ is $\mathbf{c}$ with a secret key $\mathbf{S}$, then the ciphertext of $[1, \mathbf{x}^T]^T$ is $[w, \mathbf{c}^T]^T$ with a secret key

$$\mathbf{S}' = \begin{pmatrix} 1 & 0 \\ 0 & \mathbf{S} \end{pmatrix}. \tag{19}$$

It is also worth mentioning that the structure of the weighted-inner-products operation allows us to parallelize the computation of multiple distinct polynomials, and hence reducing the computational time. For example, if we want to compute $2x_1^3 + x_1x_2x_3 - 2x_3 + 4x_3^3$, we can first compute $[2x_1^2, x_1x_2, -2, 4x_3^2]$ with a weighted-inner-products operation, and based on which we further compute $2x_1^3 + x_1x_2x_3 - 2x_3 + 4x_3^3 = [2x_1^2, x_1x_2, -2, 4x_3^2] \cdot [x_1, x_3, x_3, x_3]$ with another weighted-inner-products operation.

In general, for an arbitrary polynomial of degree $d > 0$, we can compute it with $\lceil \log_2 d \rceil$ weighted-inner-products operations and some number of addition and linear-transformation operations. But, how to divide a computation task into the three types of fundamental operations to minimize the computation and communication cost is still an interesting unsolved problem.

To compute a polynomial in the encrypted domain, party B first generates all the secret keys and key-switching matrices ahead, and then sends the key-switching matrices to party A, as the public key. Based on the key-switching matrices, party A can further evaluate the polynomial. When there is only a single ciphertext stored at or received by party A, this approach is less efficient than transmitting the ciphertext directly to party

B. It becomes very useful when the number of the ciphertexts is large and we have to execute the same computation on these ciphertexts, since we can keep using the same set of key-switching matrices, i.e., the public key.

C. Secrecy Analysis

As described in Section II, in many applications, party B wants to preserve the privacy of the processing function f to party A. Here, we show that the encrypted computation based on the three types of fundamental operations can actually achieve this goal, i.e., the processing function f is **secret** to party A in the computation process. The only information that party A (see Fig. 2) can get is the order of the types of the performed operations, not their exact expressions. For instance, in the example of computing $x_2^2 - 4x_1x_3$ in the previous subsection, party A only knows that the computation is in the form of $\mathbf{x^T H_1 x}$ (without knowing what $\mathbf{H_1}$ is), i.e., the computation function is a homogeneous polynomial of degree 2.

Specifically, for an addition operation $\mathbf{x}_1 + \mathbf{a}$, if $\mathbf{a}$ is an encrypted parameter from party B, then party A cannot determine $\mathbf{a}$; for a linear-transformation operation $\mathbf{G}\mathbf{x}_1$, party A cannot get the transformation matrix $\mathbf{G}$; and similarly, the matrices $\{\mathbf{H}_j\}$ are secret to party A in a weighted-inner-products operation $\{\mathbf{x}_1^T\mathbf{H}_j\mathbf{x}_2\}$. Composing these three types of operations result in a polynomial, and party A knows nothing about the coefficients of the polynomial and the number of terms in the polynomial. For instance, in the example of computing $2x_1^3 + x_1x_2x_3 - 2x_3 + 4x_3^3$ in the previous subsection, party A only knows that the processing function is a degree-3 polynomial, and nothing more, since any degree-3 polynomial can be realized with the same set of fundamental operations by choosing different parameters, i.e., $\mathbf{a}$, $\mathbf{G}$, and $\{\mathbf{H}_j\}$ in the fundamental operations.

From the discussion above, we see that party B preserves the privacy of all the coefficients in the polynomial to party A, and in this sense, the computation is secret. In fact, by adding few redundant operations, we can guarantee that the information known by party A about the polynomial is nothing more than the degree of the polynomial. Assume that the processing function f is a polynomial on the elements of an integer vector $\mathbf{x}$, which is encrypted as a ciphertext $\mathbf{c}$ with a secret key $\mathbf{S}$. According to (19), we can get a new ciphertext $\mathbf{c}'$ that encrypts $[1, \mathbf{x}^T]^T$ with a secret key $\mathbf{S}'$. If we evaluate the polynomial f based on the encryption of $[1, \mathbf{x}^T]^T$ instead of the encryption of $\mathbf{x}$, then we can prove that party A knows at most the degree of the polynomial. For instance, in the first example above, in order to compute $x_2^2 - 4x_1x_3$, we can perform the operation $[1, \mathbf{x}^T]\mathbf{H}_2[1, \mathbf{x}^T]^T$ with

$$\mathbf{H}_2 = \begin{pmatrix} 0 & 0 \\ 0 & \mathbf{H}_1 \end{pmatrix},$$

and in this case party A only knows that the processing function is a degree-2 polynomial – nothing else.

V. Examples of Supported Processing

In this section, we describe several examples of computation tasks based on homomorphic encryption, including feature extraction, recognition, classification, and data aggregation. The computational complexity of these tasks can be significantly reduced with encryptions and computations on integer vectors in the scenario of Fig. 2.

A. Feature Extraction

Let $\mathbf{x} \in \mathbb{Z}^m$ be a long integer vector, such as an image, whose encryption is stored at party A, e.g., data servers and sensor nodes, and only party B has the secret key. Party B is interested in getting the feature of $\mathbf{x}$, i.e., $\mathbf{y} = \mathbf{G}\mathbf{x}$ with a feature matrix $\mathbf{G} \in \mathbb{Z}^{l\times m}$ $(l \ll m)$. But party B does not want to let party A know what feature it is interested in. Our goal is to generate an encryption of the feature $\mathbf{y}$ at party A while keeping the feature matrix $\mathbf{G}$ confidential.

Let $\mathbf{c}$ be the ciphertext of $\mathbf{x}$ with a secret key $\mathbf{S}$. According to the analysis for linear-transformation operations, see (14), $\mathbf{c}$ is also a ciphertext of $\mathbf{y}$ but with another secret key $\mathbf{GS}$. Since $\mathbf{y}$ is much shorter than $\mathbf{x}$, we can use the key-switching technique to reduce the dimension of the ciphertext $\mathbf{c}$. As a result, we switch the secret key $\mathbf{GS}$ to $\mathbf{S}'$, and convert the ciphertext $\mathbf{c}$ to a new ciphertext $\mathbf{c}'$ such that $\mathbf{c}'$ is much shorter than $\mathbf{c}$.

In most applications, the vector $\mathbf{x}$ is very long, and it is represented by multiple ciphertexts $\mathbf{c}_1, \mathbf{c}_2, ..., \mathbf{c}_k$, with $\mathbf{c}_i$ encrypts $\mathbf{x}_i$. The feature of $\mathbf{x}$ is

$$\mathbf{y} = \sum_{i=1}^{k} \mathbf{G}_i\mathbf{x}_i \tag{20}$$

for a collection of feature matrices $\{\mathbf{G_i}\}$. We see that the ciphertext of $\mathbf{G}_i\mathbf{x}_i$ is $\mathbf{c}_i$ with secret key $\mathbf{G}_i\mathbf{S}$. Since all the k ciphertexts have distinct secret keys, in order to continue to perform addition operations over them, we need to switch these secret keys to a common secret key, denoted by $\mathbf{S}'$. With this new secret key, the ciphertext of $\mathbf{G}_i\mathbf{x}_i$ is converted to $\mathbf{c}_i'$ instead of $\mathbf{c}_i$. Finally, we obtain $\mathbf{c}' = \sum_{i=1}^{k} \mathbf{c}_i' \mod q$ as a valid encryption of $\mathbf{y}$, and its secret key is $\mathbf{S}'$.

Let's consider the following concrete example. Assume that $\mathbf{x}$ is an image of 2^{14} pixels with each pixel represented by an integer of 8 bits. It is divided into 2^7 vectors, denoted by $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{2^7}$, each contains 2^7 pixels and is encrypted by a ciphertext of length 2^8. Assume that the length of the feature $\mathbf{y}$ is 10, then we have 2^7 feature matrices with each matrix $\mathbf{G}_i \in \mathbb{Z}_{2^8}^{10\times 2^7}$.

We construct each ciphertext $\mathbf{c}_i$ (that encrypts $\mathbf{x}_i$) with the following parameters: the constant $w = 2^{20}$ and the modulus $q \approx 2^{50}$. Both the secret keys and noise matrices are generated based on a noise distribution χ, which is the uniform distribution on $\mathbb{Z}_4$. After the initial encryption (before performing linear transformations), it has

$$\mathbf{S}\mathbf{c}_i = q\mathbf{k}_i + w\mathbf{x}_i + \mathbf{E}_1\mathbf{x}_i^*, \tag{21}$$

where $\mathbf{S}$ is the secret key, $\mathbf{k}_i$ is an integer vector, $\mathbf{E}_1 \in \mathbb{Z}_4^{2^7 \times 2^{10}}$ is a noise matrix, and $\mathbf{x}_i^* \in \{0,1\}^{2^{10}}$ is the binary representation of $\mathbf{x}_i$. Then we apply linear transformation and key switching, by which, the ciphertext $\mathbf{c}_i$ is converted into a new ciphertext $\mathbf{c}_i'$ of length 2^8 with a new secret key $\mathbf{S}'$. According to the analysis for the key-switching technique,

$$\mathbf{S}'\mathbf{c}_i' = q\mathbf{k}_i' + w\mathbf{G}_i\mathbf{x}_i + (\mathbf{G}_i\mathbf{E}_1\mathbf{x}_i^* + \mathbf{E}_2\mathbf{c}_i^*), \tag{22}$$

where $\mathbf{E}_2 \in \mathbb{Z}_4^{10\times(2^8*50)}$ is a noise matrix and $\mathbf{c}_i^* \in \{0,1\}^{2^8*50}$ is the binary representation of $\mathbf{c}_i$. We see that the noise part is $\mathbf{e}_i = \mathbf{G}_i\mathbf{E}_1\mathbf{x}_i^* + \mathbf{E}_2\mathbf{c}_i^*$, and its magnitude is

$$|\mathbf{e}_i| \leq 2^7 \times 2^{10} \times 4 \times 2^8 + 2^8 \times 50 \times 4 \approx 2^{27}. \tag{23}$$

Although $|\mathbf{e}_i|$ is possible to be larger than w, it is much smaller than $|\mathbf{G}_i\mathbf{x}_i|$. Since we are only interested in the significant bits of the elements in the feature $\mathbf{y}$, the effect of the noise part is actually ignorable. Finally, by adding all the ciphertexts $\mathbf{c}_i'$ with $1 \leq i \leq 2^7$, we get the ciphertext $\mathbf{c}$ for $\mathbf{y}$ such that

$$\mathbf{S}'\mathbf{c} = q\mathbf{k} + w\mathbf{y} + \mathbf{e}, \tag{24}$$

with $\mathbf{k}$ an integer vector, $|\mathbf{y}| \leq 2^{30}$, $|\mathbf{e}| \leq 2^{34}$, and $w|\mathbf{y}| \leq 2^{50} \approx q$. Typically, $|w\mathbf{y}| \gg |\mathbf{e}|$, hence $\mathbf{y}$ can be obtained with sufficient precision by decrypting $\mathbf{c}$.

In this example, it requires roughly 200K bytes space to store a 16 KB image. The communication cost of each key-switching matrix is at most 16 KB (when the random matrix $\mathbf{A}$ in (5) is created with pseudo random bits). Hence, the total communication cost is about 2 MB, which is a reasonable amount of cost for large cloud storage applications, where a user stores, e.g., more than 10 GB of images. In a sense, the transmitted data (key-switching matrices) can be considered as the encryption of the feature matrices, and the communication cost is only 12 times of the total size of the feature matrices. In addition, the computation cost is about 50 times of that in the plaintext domain, which is usually acceptable in practical applications.

B. Recognition

In the scenario of Fig. 2, assume that party A has the encryptions of a collection of integer vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k$. Party B wants to check whether there is a vector $\mathbf{x}_i$ that is similar to a private vector $\mathbf{a}$, i.e., $\|\mathbf{x}_i - \mathbf{a}\| \leq t$ for a threshold t, where $\|\mathbf{x}_i - \mathbf{a}\|$ is the Euclid distance between $\mathbf{x}_i$ and $\mathbf{a}$. The question is how to compute $\|\mathbf{x}_i - \mathbf{a}\|$ in the encrypted domain when both $\mathbf{x}_i$ and $\mathbf{a}$ are encrypted, so that party A only needs to transmit the encryption of $\|\mathbf{x}_i - \mathbf{a}\|$ to party B.

For convenience, we write $\mathbf{x}_i$ as $\mathbf{x}$. Since

$$\|\mathbf{x} - \mathbf{a}\|^2 = \mathbf{x}^T\mathbf{x} + \mathbf{a}^T\mathbf{a} - 2\mathbf{a}^T\mathbf{x}, \tag{25}$$

we can compute $\mathbf{x}^T\mathbf{x}$, $\mathbf{a}^T\mathbf{a}$, and $\mathbf{a}^T\mathbf{x}$, respectively, based on weighted-inner-products operations. The secret key for $\mathbf{x}^T\mathbf{x}$ is $\mathrm{vec}(\mathbf{S^T S})$ and the ciphertext is $\lceil \frac{\mathrm{vec}(\mathbf{c_x c_x}^T)}{w} \rfloor_q$; similarly, the secret key for $\mathbf{a}^T\mathbf{a}$ is $\mathrm{vec}(\mathbf{S^T S})$ and the ciphertext is $\lceil \frac{\mathrm{vec}(\mathbf{c_a c_a}^T)}{w} \rfloor_q$; the secret key for $\mathbf{a}^T\mathbf{x}$ is $\mathrm{vec}(\mathbf{S^T S})$ and the ciphertext is $\lceil \frac{\mathrm{vec}(\mathbf{c_a c_x}^T)}{w} \rfloor_q$. Since they have the same secret key, we can directly perform addition operations on them. As a result, we get the ciphertext of $\|\mathbf{x} - \mathbf{a}\|^2$, which is

$$\mathbf{c}' = \left\lceil \frac{\mathrm{vec}(\mathbf{c_x c_x}^T + \mathbf{c_a c_a}^T - 2\mathbf{c_a c_x}^T)}{w} \right\rfloor_q, \tag{26}$$

and the secret key is $\mathrm{vec}(\mathbf{S^T S})$. Now, the dimension of the ciphertext is still too high, we can further apply the key-switching technique to reduce the dimension.

In what follows, we provide an example: the integer vector $\mathbf{x}$ is in $\mathbb{Z}_{2^6}^{2^7}$, and the length of its ciphertext $\mathbf{c_x}$ is 2^8. The noise distribution χ is the uniform distribution on $\mathbb{Z}_4$. Based on these parameters, the ciphertext $\mathbf{c_x}$ satisfies

$$\mathbf{S}\mathbf{c_x} = q\mathbf{k}_1 + w\mathbf{x} + \mathbf{e}_1, \tag{27}$$

with $|\mathbf{k}_1| \leq 6 * 2^{16}$ and $|\mathbf{e_1}| \leq 6 * 2^9$. Then, we let $w \approx 2^{60}$ and $q = 2^{20}w$, and we compute $\mathbf{x}^T\mathbf{x}$ in the encrypted domain based on a weighted-inner-products operation. As a result, we get a secret key $\mathbf{s}' = \mathrm{vec}(\mathbf{S^T S})$, and the ciphertext of $\mathbf{x}^T\mathbf{x}$ is $\mathbf{c}_\mathbf{x}' = \lceil \frac{\mathrm{vec}(\mathbf{c_x c_x}^T)}{w} \rfloor_q$. According to the analysis for the weighted-inner-products operation, we have

$$\mathbf{s}'\mathbf{c}_\mathbf{x}' = qk_2 + w\mathbf{x}^T\mathbf{x} + e_2, \tag{28}$$

where k_2 is an integer and $|e_2| \leq 2 * 2^{20} * 2^7|\mathbf{k_1}| * |\mathbf{e_1}| \leq 2^{59} < w$. It implies that $\mathbf{x}^T\mathbf{x}$ can be correctly decrypted from $\mathbf{c}_\mathbf{x}'$ based on the secret key $\mathbf{s}'$. The same analysis applies to all the terms in $\|\mathbf{x} - \mathbf{a}\|^2 = \mathbf{x}^T\mathbf{x} + \mathbf{a}^T\mathbf{a} - 2\mathbf{a}^T\mathbf{x}$, and in this case, it is good to set $w \approx 2^{60}$ and $q = 2^{20}w$ with ciphertext length $n = 256$ and integer-vector length $m = 128$. Although there will be an extra noise introduced by the next-step key switching, the magnitude of the extra noise is much smaller than that of e_2, hence ignorable in our analysis.

Our observation is that if we modify the encryption process, then the computation can be further simplified. Instead of encrypting $\mathbf{x}$, we assume that $\mathbf{x}' = (1, \mathbf{x^T x}, \mathbf{x}^T)^T$ is encrypted. If we define $\mathbf{a}' = (\mathbf{a^T a}, 1, -2\mathbf{a}^T)^T$ for party B, then we can get $\|\mathbf{x}-\mathbf{a}\|^2 = \mathbf{a'}^T\mathbf{x}'$. It implies that $\|\mathbf{x}-\mathbf{a}\|^2$ can be computed based on a single linear-transformation operation on $\mathbf{x}'$ with a transform matrix $G = \mathbf{a}'$, where $\mathbf{a}'$ is only known by party B. This method is computationally much more efficient than the method above, as linear-transformation operations are much simpler to implement than weighted-inner-product operations.

C. Classification

Classification has many important applications in data clouds and sensing systems. For example, in email services, a user may want to know whether an encrypted email at server is a spam or not; and in sensor monitoring networks, the base station may want to determine the existence of certain events, such as fires, in a specific area.

A simple and widely-used classifier is a linear classifier: given a vector $\mathbf{x} \in \mathbb{Z}_p^m$, the classifier outputs 0 if and only $\mathbf{w} \cdot \mathbf{x} \leq t$ for a weight vector $\mathbf{w}$ and a threshold t, i.e, it is described by a function

$$f(\mathbf{x}) = \begin{cases} 0 & \text{if } \mathbf{w} \cdot \mathbf{x} \leq t \\ 1 & \text{otherwise.} \end{cases} \tag{29}$$

In order to apply linear classification for spam or event detection, the user or the base station only needs to collect $\mathbf{w} \cdot \mathbf{x}$ instead of the original long vector $\mathbf{x}$. Actually, $\mathbf{w} \cdot \mathbf{x}$ can be computed with a single linear-transformation operation, where $\mathbf{w}^T$ is the transformation matrix from the user or the base station. The computation process is simple and efficient.

Furthermore, we can implement a more sophisticated non-linear classifier by replacing $\mathbf{w} \cdot \mathbf{x}$ with a degree-2 polynomial on $\mathbf{x}$. Such a classifier can be designed based on the support vector machines (SVM), and the corresponding degree-2 polynomial can be evaluated based on a single weighted-inner-products operation.

D. Data Aggregation

With the concerns of data security and privacy, in sensor networks, it is desired to encrypt data immediately at the sensor nodes when the data is generated, and the encryption process is based on a public key broadcasted by the base station. The encrypted data is aggregated from thousands of nodes to the base station via multiple hops. With limited communication bandwidth and energy constraints, data processing is necessary at relay nodes to reduce the amount of transmitted data.

Assume that each sensor node detects an integer value, encrypts it and sends it to the base station. Some relay nodes may receive multiple ciphertexts $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_k$ from different sensor nodes, representing integer values $x_1, x_2, \ldots, x_k$, respectively. The question is that how to pack them together to form a single ciphertext $\mathbf{c}'$, which encrypts $(x_1, x_2, \ldots, x_k)$.

Let's consider a secret key $\mathbf{S}$ and a ciphertext $\mathbf{c}$ constructed as follows:

$$\mathbf{c} = \left\{ \begin{array}{c} \mathbf{c}_1 \\ \mathbf{c}_2 \\ \vdots \\ \mathbf{c}_k \end{array} \right\}, \qquad \mathbf{S} = \left\{ \begin{array}{c} \mathbf{s}, 0, \ldots, 0 \\ 0, \mathbf{s}, \ldots, 0 \\ \vdots \\ 0, 0, \ldots, \mathbf{s} \end{array} \right\}, \tag{30}$$

where $\mathbf{s}$ is the secret key of the original ciphertexts $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_k$. It is easy to see that

$$\mathbf{Sc} = w(x_1, x_2, ..., x_k)^T + \mathbf{e} \mod q \tag{31}$$

with a noise vector $\mathbf{e}$. It is clear to see that $\mathbf{c}$ is a ciphertext of $(x_1, x_2, ..., x_k)^T$, but its dimension is k times of the original dimension n. With the key-switching technique, we can reduce the dimension of the ciphertext by converting $\mathbf{c}$ to a new ciphertext $\mathbf{c}'$. As a result, we have packed a collection of ciphertexts $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{x}_k$ as a single ciphertext $\mathbf{c}'$.

In some other occasions, the base station may be only interested in the statistics of the sensed values, such as their mean, max and distribution. In order to extract these information, we adopt another way of encrypting the sensed values: at a sensor node, instead of encrypting a sensed value x_i, we represent x_i as a binary vector and encrypt this binary vector. Specifically, let x_i be an integer in $\mathbb{Z}_p$, e.g., $p = 128$, we encode x_i into $\mathbf{x}_i \in \{0, 1\}^p$ such that only the $(x_i + 1)$th entry in $\mathbf{x}_i$ is 1 and all other entries are 0s, and then we encrypt $\mathbf{x_i}$ with a ciphertext $\mathbf{c}_i$. For example, if a sensor node detects $x_i = 3 \in \mathbb{Z}_8$, then $\mathbf{x}_i = [0, 0, 0, 1, 0, 0, 0, 0]$. If a relay node receives a collection of ciphertexts, denoted by $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_k$, it simply fuses them by adding them together, i.e., it generates and forwards a new ciphertext

$$\mathbf{c} = \sum_{i=1}^{k} \mathbf{c_i} \mod q. \tag{32}$$

Finally, the base station obtains a ciphertext $\mathbf{c}'$ that encrypts an integer vector $\mathbf{n}'$ with $\mathbf{n}' = \sum_i \mathbf{x}_i$, which represents the frequencies of all the integers in $\mathbb{Z}_p$ detected by the sensor nodes. Based on the frequency vector $\mathbf{n}' = [n_1, n_2, ..., n_p]$, the base station can obtain the statistics of all the sensed values, including the mean, max, variance, etc. For example, the mean is

$$\text{mean}(x) = \frac{\sum_{j=1}^{p} (j-1) n_j}{\sum_{j=1}^{p} n_j}; \tag{33}$$

and the max is

$$\max(x) = \max\{j : 1 \le j \le p, n_j > 0\} - 1. \tag{34}$$

VI. Concluding Remarks

In this paper, we studied a homomorphic encryption scheme on integer vectors, as a natural extension of the recently developed homomorphic encryption schemes based on the learning with errors (LWE) assumption. In contrast to previous work, we focused on a new scenario that has wide applications in data clouds and sensing systems. We demonstrated that, in this scenario, the encryption scheme supports three types of fundamental operations on integer vectors, and based on which we can compute an arbitrary polynomial on integers within a certain degree efficiently and secretely. In addition, we described a few examples of computation tasks, including feature extraction, recognition, classification, and data aggregation. A strong implication of this paper is that although it is difficult to construct universal homomorphic-encryption schemes for general computations in practice, for some specific applications we may find simple homomorphic-encryption schemes with reasonable communication and computation costs.

The work in this paper also brings us many interesting problems that require further studies. For example, how to divide a computation task into the three types of fundamental operations that minimize the overall communication and computation cost? How to reduce the public-key size, i.e., the total size of the key-switching matrices? Can any task be represented as polynomials modulus l for an integer l, and what is the best presentation?

Acknowledgment

The authors thank Venkat Chandar for useful technical discussions.

References

[1] R. L. Rivest, L. Adleman, and M. L. Dertouzos, "On data banks and privacy homomorphisms," in *Foundations of Secure Computation*, R. Demillo, D. Dobkin, A. Jones, and R. Lipton, Eds. New York: Academic, pp. 169–180, 1978.

[2] S. Goldwasser and S. Micali, "Probabilistic encryption and how to play mental poker keeping secret all partial information," in *Proc. STOC*, ACM, pp. 365–377, 1982.

[3] Taher El-Gamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in *Proc. CRYPTO*, pp. 10–18, 1984.

[4] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *EUROCRYPT*, pp. 223–238, 1999.

[5] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. STOC'09*, pp. 169–178.

[6] C. Aguilar-Melchor, P. Gaborit, and J. Herranz, "Additively homomorphic encryption with d-operand multiplications," in *Proc. CRYPTO'10* (LNCS) vol. 6223, pp. 138–154.

[7] Z. Brakerski and V. Vaikuntanathan, "Efficient fully homomorphic encryption from (standard) LWE," in *Proc. 2011 IEEE 52nd Annu. Symp. on Foundations of Computer Science*, pp. 97106.

[8] Z. Brakerski and V. Vaikuntanathan, "Fully homomorphic encryption from ring-LWE and security for key dependent messages," in *Proc. Advances in Cryptology, (CRYPTO 2011)*, vol. 6841, pp. 505–524.

[9] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping, in *Proc. 3rd Innovations in Theoretical Computer Science Conf.*, 2012, pp. 309–325.

[10] J.-S. Coron, A. Mandal, D. Naccache, M. Tibouchi, "Fully homomorphic encryption over the integers with shorter public keys," in *CRYPTO'2011* (LNCS) vol. 6841, pp. 487–504, 2011.

[11] M. Naehrig, K. Lauter, and V. Vaikuntanathan, "Can homomorphic encryption be practical?" in *Proc. ACM Workshop on Cloud Computing Security*, 2011.

[12] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers," in *EUROCRYPT'2010* (LNCS) vol. 6110, pp. 24–43.

[13] C. Gentry, S. Halevi, and V. Vaikuntanathan, "A simple BGN-type cryptosystem from LWE," in *Proc. EUROCRYPT'2010* (LNCS) vol. 6110, pp. 506–522.

[14] C. Gentry, S. Halevi, and N. Smart, "Homomorphic evaluation of the AES circuit," in *Proc. CRYPTO'2012*, (LNCS) vol. 7417, pp. 850–867, 2012.

[15] C. Aguilar-Melchor, S. Fau, C. Fontaine, G. Gogniat, and R. Sirdey, "Recent advances in homomorphic encryption," *IEEE Signal Processing Magazine*, pp. 108–117, March 2013.

[16] N. P. Smart and F. Vercauteren, "Fully homomorphic SIMD operations," *Designs, Codes and Cryptography*, Springer, issn 0925-1022, 2012.

[17] M. Yasuda, T. Shimoyama, J. Kogure, K. Yokoyama, and T. Koshiba, "Packed homomorphic encryption based on ideal lattices and its application to biometrics," *Security Engineering and Intelligence Informatics* (LNCS) vol. 8128, pp. 55–74, 2013.

[18] J. H. Cheon, J.-S. Coron, et al., "Batch fully homomorphic encryption over the integers," in *Advances in Cryptology - EUROCRYPT'2013*, (LNCS) vol. 7881, pp. 315–335, 2013.

[19] C. Peikert, V. Vaikuntanathan, and B. Waters, "A framework for efficient and composable oblivious transfer," In *CRYPTO'08*, (LNCS) vol. 5157, pp. 554–571, 2008.

[20] Z. Brakerski, C. Gentry, and S. Halevi, "Packed ciphertexts in LWE-based homomorphic encryption," in *Public-Key Cryptography - PKC*, (LNCS) vol. 7778, pp. 1–13, 2013.

[21] R. L. Lagendijk, Z. Erkin, and M. Barni, "Encrypted signal processing for privacy protection," *IEEE Signal Processing Magazine*, pp. 82–105, Jan. 2013.

[22] R. Riggio, and S. Sicari, "Secure aggregation in hybrid mesh/sensor networks," in *Proc. International Conference on Ultra Modern Telecommunications & Workshops*, 2009.

[23] B. Applebaum, D. Cash, C. Peikert, and A. Sahai, "Fast cryptographic primitives and circular-secure encryption based on hard learning problems," in *CRYPTO'09*, (LNCS) vol. 5677, pp. 595–618, 2009.