ELECTRONIC APPARATUS AND PERFORMING OPERATIONS ON HOMOMORPHIC CIPHERTEXT AND CONTROL METHOD THEREOF

Abstract

Provided are an electronic apparatus and a control method thereof. The apparatus includes: a memory for storing at least one instruction; and a processor, wherein the processor may be configured to acquire the homomorphic ciphertext by using a Residue Number System (RNS) modulus including a plurality of moduli each having a size corresponding to a machine word size, and perform an operation on the homomorphic ciphertext by using rational rescaling.

Background/Summary

TECHNICAL FIELD

[0001] The present disclosure relates to an electronic apparatus and a control method thereof, and more particularly, to an electronic apparatus for performing operations on a homomorphic ciphertext and a control method thereof.

BACKGROUND ART

[0002] As communication technology advances and electronic apparatuses become more widespread, continuous efforts are being made to ensure secure communication between the electronic apparatuses. Accordingly, encryption and decryption technology are used in most communication environments.

[0003] If a message encrypted by the encryption technology is transmitted to the other party, the other party is required to perform decryption to use the message. In this case, the other party may waste resources and time in a process of decrypting encrypted data. In addition, the message may be easily leaked to a third party if the third party hacks the message while the other party temporarily decrypts the message for operation.

[0004] To solve these problems, homomorphic encryption methods are being studied. Homomorphic encryption may acquire the same result as an encrypted value acquired after performing an operation on a plaintext, even if the operation is performed on a ciphertext itself acquired without decrypting encrypted information. Therefore, various operations may be performed without decrypting the ciphertext.

[0005] In particular, Cheon-Kim-Kim-Song (CKKS) scheme is a Fully Homomorphic Encryption (FHE) scheme that allows approximate computation of real-valued encrypted data. In CKKS encoding, the message is multiplied by a scaling factor and rounded to ensure precision. The scaling factor is squared during a multiplication process, and rescaling is thus required to reduce the scaling factor again. This process not only reduces the scaling factor but also reduces a ciphertext modulus, and bootstrapping is thus required to recover the modulus after multiple multiplications.

[0006] The implementation of the CKKS scheme adopts a Residue Number System (RNS) within CKKS (RNS-CKKS) for efficiency. In RNS-CKKS, the ciphertext modulus is composed of a product of Number Theoretic Transform (NTT) primes, which are referred to as RNS moduli. By Chinese Remainder Theorem (CRT), a modulus operation expressed as $Q=\pi q_i$ may be decomposed into operations on $\hat{q}_i$, i.e., NTT primes.

However, the rescaling is allowed only for the NTT primes that divide the ciphertext modulus, this prime needs to be as large as the scaling factor. The NTT primes may be much smaller than a word size of the electronic apparatus, and a relationship between the modulus and the scaling factor may thus lead to computational inefficiency.

[0007] Most implementations select a scaling factor that is as small as possible to reduce modulus waste. The modulus is a limited resource and thus has a great impact on computational efficiency. In particular, it is important to efficiently distribute a limited modulus budget among various functions. Bootstrapping may consume a huge amount of modulus, and the amount of modulus left for homomorphic computation may thus usually be smaller. Therefore, saving modulus consumption is often the top priority, and the scaling factor is selected as small as possible. Meanwhile, the number of RNS moduli needs to be minimized for the efficiency. The ciphertext may be represented, calculated, and stored using each RNS modulus, computational and space complexity may be proportional to the number of RNS moduli included in the ciphertext modulus. In this regard, the efficiency may be maximized if the RNS modulus is selected as close as possible to the machine word size. However, this selection may conflict with the scaling factor minimization strategy described above. As a result, existing efficient implementations of CKKS may select one of the following two options: [0008] Option 1) Minimize the number of RNS moduli by filling a given maximum modulus budget using word-sized NTT primes of the electronic apparatus; and [0009] Option 2) Maximizing multiplication depth by filling the given maximum modulus budget using scaling factor-sized NTT primes.

[0010] Comparison of the two options above may be described as a tradeoff between flexibility and the efficiency. Option 1 may cover a wider range of applications with a single parameter set than Option 2, while being less computationally efficient. Option 2 requires regenerating a public parameter and an evaluation key depending on a circuit if a larger scaling factor is required.

[0011] In conclusion, given the conflict described above, it is beneficial to break the relationship between the RNS modulus and the scaling factor. If the RNS modulus and the scaling factor are selected independently, an overall computation speed may be improved by selecting a word-size RNS modulus and an application-dependent scaling factor.

DISCLOSURE

Technical Solution

[0012] According to an embodiment of the present disclosure, provided is an electronic apparatus including: a memory for storing at least one instruction; and a processor, wherein the processor is configured to acquire a homomorphic ciphertext by using a Residue Number System (RNS) modulus including a plurality of moduli each having a size corresponding to a machine word size, and perform an operation on the homomorphic ciphertext by using rational rescaling.

[0013] The plurality of moduli may include sprout moduli composed of a product of primes each having a size less than or equal to the machine word size.

[0014] The processor may be configured to generate an intermediate modulus by resurrecting the sprout modulus, and acquire an output modulus by rescaling the intermediate modulus, the intermediate modulus being a multiple of the output modulus.

[0015] The processor may be configured to perform a key switching operation on the homomorphic ciphertext by using the sprout modulus.

[0016] In case that the sprout modulus is expressed as a product of g_1, which is a 31-bit Number Theoretic Transform (NTT) prime, g_2, which is a 16-bit NTT prime or in the form of $2^{16}+1$, and g_3, which is a value in the form of 2, the processor may be configured to process a product of g_1 and g_2 by using composite number-based NTT (Composite NTT) and process g_3 by using embedded NTT to thus integrate their results into G_3 if the machine word size is greater than 38 bits and less than or equal to 64 bits, and process g_3 by using the embedded NTT to thus acquire G_3 if the machine word size is less than or equal to 38 bits.

[0017] Two NTT processors may be used if the machine word size is greater than 38 bits and less than or equal to 64 bits, and one NTT processor may be used if the machine word size is less than or equal to 38 bits.

[0018] The processor may be configured to generate an intermediate modulus by upscaling the RNS modulus, perform a key switching operation on the intermediate modulus, and perform rescaling on the intermediate modulus on which the key switching operation is performed.

[0019] The processor may be configured to adjust the plurality of homomorphic ciphertexts having different moduli and scaling factors by using the rational rescaling.

[0020] According to an embodiment of the present disclosure, provided is a control method of an electronic apparatus, the method including: acquiring a homomorphic ciphertext by using a Residue Number System (RNS) modulus including a plurality of moduli each having a size corresponding to a machine word size included in the electronic apparatus; and performing an operation on the homomorphic ciphertext by using rational rescaling.

[0021] The plurality of moduli may include sprout moduli composed of a product of primes each having a size less than or equal to the machine word size.

[0022] The performing may include generating an intermediate modulus by resurrecting the sprout modulus, and acquiring an output modulus by rescaling the intermediate modulus, the intermediate modulus being a multiple of the output modulus.

[0023] In the performing, a key switching operation may be performed on the homomorphic ciphertext by using the sprout modulus.

[0024] In case that the sprout modulus is expressed as a product of g_1, which is a 31-bit Number Theoretic Transform (NTT) prime, g_2, which is a 16-bit NTT prime or in the form of $2^{16}+1$, and g_3, which is a value in the form of 2, in the performing, a product of g_1 and g_2 may be processed by using Composite NTT, and g_3 may be processed by using embedded NTT to thus integrate their results into G_3 if the machine word size is greater than 38 bits and less than or equal to 64 bits, and g_3 may be processed by using the embedded NTT to thus acquire G_3 if the machine word size is less than or equal to 38 bits.

[0025] Two NTT processors may be used by the electronic apparatus if the machine word size is greater than 38 bits and less than or equal to 64 bits, and one NTT processor may be used by the electronic apparatus if the machine word size is less than or equal to 38 bits.

[0026] The performing may include generating an intermediate modulus by upscaling the RNS modulus, performing a key switching operation on the intermediate modulus, and performing rescaling on the intermediate modulus on which the key switching operation is performed.

[0027] The method may further include adjusting the plurality of homomorphic ciphertexts having different moduli and scaling factors by using the rational rescaling.

Description

DESCRIPTION OF DRAWINGS

[0028] FIG. **1** is a diagram for describing a structure of a network system according to an embodiment of the present disclosure.

[0029] FIG. **2** is a block diagram showing a configuration of a computing device according to an embodiment of the present disclosure.

[0030] FIG. **3** is a diagram showing a conventional Residue Number System (RNS) modulus and a modulus according to an embodiment of the present disclosure.

[0031] FIGS. **4**A and **4**B are diagrams for describing rational rescaling according to an embodiment of the present disclosure.

[0032] FIGS. **5** and **6** are diagrams for describing a conventional key switching operation.

[0033] FIG. **7** is a diagram for describing an embodiment using a temporary ciphertext modulus chain according to an embodiment of the present disclosure.

[0034] FIGS. **8** to **11** are diagrams for describing an embodiment using a sprout modulus according to an embodiment of the present disclosure.

[0035] FIGS. **12** to **17** are diagrams for describing various algorithms for performing operations on the homomorphic ciphertext according to an embodiment of the present disclosure.

BEST MODEL

[0036] Hereinafter, the present disclosure is described in detail with reference to the accompanying drawings. Encryption/decryption may be applied as necessary to a process of transmitting information (or data) that is performed in the present disclosure, and an expression describing the process of transmitting the information (or data) in the present disclosure and the claims should be interpreted as including all cases of the encryption/decryption even if not separately mentioned. In the present disclosure, an expression such as "transmission (transfer) from A to B" or "reception from A to B" may include transmission (transfer) or reception while having another medium included in the middle, and may not necessarily express only the direct transmission (transfer) or reception from A to B.

[0037] In describing the present disclosure, a sequence of each step should be understood as non-restrictive unless a preceding step in the sequence of each step needs to logically and temporally precede a subsequent step. That is, except for the above exceptional case, the essence of the present disclosure is not affected even if a process described as the subsequent step is performed before a process described as the preceding step, and the scope of the present disclosure should also be defined regardless of the sequences of the steps. In addition, in this specification, "A or B" may be defined to indicate not only selectively indicating either A or B, but also including both A and B. In addition, a term "including" in the present disclosure may encompass a concept of further including other components in addition to components listed as being included.

[0038] The present disclosure only describes essential components necessary for describing the present disclosure, and does not mention components unrelated to the essence of the present disclosure. In addition, it should not be interpreted as an exclusive concept that the present disclosure includes only the mentioned components, and should be interpreted as a non-exclusive concept that the present disclosure may include other components as well.

[0039] In addition, in the present disclosure, a "value" may be defined as a concept that includes a vector as well as a scalar value. In addition, in the present disclosure, an expression such as "calculate" or "compute" may be replaced with an expression that generates a result of the corresponding calculation or computation. In addition, unless otherwise indicated, an operation on a ciphertext described below refers to a homomorphic operation. For example, addition on homomorphic ciphertexts indicates homomorphic addition on two homomorphic ciphertexts.

[0040] Mathematical operations and calculations in each step of the present disclosure described below may be implemented as computer operations by a known coding method and/or coding designed to be appropriate for the present disclosure to perform the corresponding operations or calculations.

[0041] Specific equations described below are illustratively provided among possible alternatives, and the scope of the present disclosure should not be construed as being limited to the equations mentioned in the present disclosure.

[0042] For convenience of description, the present disclosure defines the following notations. [0043] $a \leftarrow D$: Select an element a based on distribution D. [0044] s**1**, s**2** $\in$ R: S**1**, S**2**: Each of S**1** and S**2** is an element belonging to a set R. [0045] mod(q): Perform a modulus operation with an element q. [0046] $\llcorner$ .Math.$\urcorner$: Round an internal value.

[0047] Hereinafter, various embodiments of the present disclosure are described in detail with reference to the accompanying drawings.

[0048] FIG. **1** is a diagram for describing a structure of a network system according to an embodiment of the present disclosure.

[0049] Referring to FIG. **1**, the network system may include a plurality of electronic apparatuses **100-1** to **100-*n***, a first server device **200**, and a second server device **300**, and the respective components may be connected to each other via a network **10**.

[0050] The network **10** may be implemented as any of various forms of wired/wireless communication networks, a broadcast communication network, an optical communication network, a cloud communication network or the like, and the respective devices may be connected to each other without a separate medium, such as wireless fidelity (Wi-Fi), Bluetooth, or Near Field Communication (NFC).

[0051] FIG. **1** shows that the plurality of electronic apparatuses **100-1** to **100-*n*** are provided. However, the plurality of electronic apparatuses are not necessarily required to be used, and a single apparatus may be used instead. As an example, the electronic apparatuses **100-1** to **100-*n*** may be implemented in various forms of apparatuses such as smartphones, tablets, game players, personal computers (PCs), laptop PCs, home servers, or kiosks, and may also be implemented in the form of home appliances using Internet of Things (IoT) functions.

[0052] A user may input various information by using the electronic apparatuses **100-1** to **100-*n*** that the user uses. The input information may be stored in the electronic apparatuses **100-1** to **100-*n*** themselves, or may also be transmitted to and stored in an external device for reasons such as storage capacity and security. As shown in FIG. **1**, the first server device **200** may serve to store such information, and the second server device **300** may serve to utilize some or all of the information stored in the first server device **200**.

[0053] Each of the electronic apparatuses **100-1** to **100-*n*** may homomorphically encrypt the input information and transmit a homomorphic ciphertext to the first server device **200**.

[0054] Each of the electronic apparatuses **100-1** to **100-*n*** may include an error, i.e., encryption noise calculated in a process of performing homomorphic encryption, in the ciphertext. In detail, the homomorphic ciphertext generated by each of the electronic apparatuses **100-1** to **100-*n*** may be generated in a form in which a result value including a message and an error value is restored if the homomorphic ciphertext is decrypted later utilizing a secret key.

[0055] As an example, the homomorphic ciphertext generated by each of the electronic apparatuses **100-1** to **100-*n*** may be generated in a form that satisfies the following property if decrypted utilizing the secret key.

[00001] $\text{Dec}(\text{ct}, \text{sk}) = \text{.Math. ct}, \text{sk .Math.} = M + e (\text{mod} q)$ [Equation1]

[0056] Here, <and > indicate dot product operation (or usual inner product), ct indicates the ciphertext, sk indicates the secret key, M indicates a plaintext message, e indicates the encryption error value, and mod q indicates a modulus of the ciphertext. q needs to be selected to be larger than a result value M multiplied by a scaling factor Δ to the message. If an absolute value of an error value e is sufficiently smaller than M, a decrypted value M+e of the ciphertext may be a value that may replace an original message by the same precision in a significant figure operation. Among decrypted data, the error may be disposed on the least significant bit (LSB) side, and M may be disposed on the next least significant bit side.

[0057] If a size of the message is too small or too large, the size may be adjusted using the scaling factor. If the scaling factor is used, not only a message in an integer form but also a message in a real number form may be encrypted, and its usability may thus be greatly increased. In addition, the size of the message may be adjusted utilizing the scaling factor to thus also adjust a size of an effective region, that is, a region where the messages exist in the ciphertext after the operation is performed.

[0058] In some embodiments, the modulus q of the ciphertext may be set and used in various forms. As an example, the modulus of the ciphertext may be set in a form of an exponential power $q = \Delta^L$ of the scaling factor Δ. If Δ is 2, the modulus may be set to a value such as $q = 2^{10}$.

[0059] In addition, the homomorphic ciphertext according to the present disclosure is described assuming that fixed point-numbers are used. However, the homomorphic ciphertext may also be applied even to a case where floating-point numbers are used.

[0060] The first server device **200** may store the received homomorphic ciphertext in a ciphertext state without decrypting the ciphertext.

[0061] The second server device **300** may request a specific processing result for the homomorphic ciphertext from the first server device **200**. The first server device **200** may perform a specific operation based on the request from the second server device **300** and then transmit its result to the second server device **300**.

[0062] As an example, if ciphertexts ct**1** and ct**2** transmitted from the two electronic apparatuses **100-1** and **100-2** are stored in the first server device **200**, the second server device **300** may request the first server device **200** for a value acquired by combining information provided by the two electronic apparatuses **100-1** and **100-2**. The first server device **200** may perform an operation for combining the two ciphertexts based on the request

and then transmit a result value ct**1**+ct**2** to the second server device **300**.

[0063] Due to a property of the homomorphic ciphertext, the first server device **200** may perform the operation without decrypting the ciphertext, and the result value may also be generated in a ciphertext form. In the present disclosure, the result value acquired using the operation is referred to as an operation result ciphertext.

[0064] The first server device **200** may transmit the operation result ciphertext to the second server device **300**. The second server device **300** may decrypt the received operation result ciphertext to thus acquire the operation result value of data included in each homomorphic ciphertext.

[0065] Meanwhile, the electronic apparatus **100** may acquire the homomorphic ciphertext by using a Residue Number System (RNS) modulus including the plurality of moduli each having a size corresponding to a word size of the electronic apparatus **100**, and use rational rescaling, thereby performing an operation on the homomorphic ciphertext. In one or more embodiments, the plurality of moduli may include sprout moduli composed of a product of primes each having a size less than or equal to the word size, and the electronic apparatus **100** may perform various operations (e.g., key switching operation) on the homomorphic ciphertext by using the sprout modulus. In one or more embodiments, the electronic apparatus **100** may generate an intermediate modulus by upscaling the RNS modulus, perform the key switching operation on the intermediate modulus, and perform rescaling on the intermediate modulus on which the key switching operation is performed, thereby performing the key switching operation on the homomorphic ciphertext.

[0066] In this way, the electronic apparatus **100** may perform efficient multiplication operations while minimizing the number of RNS moduli, thereby enabling a faster operation on the homomorphic ciphertext.

[0067] Meanwhile, FIG. **1** shows a case where the encryption is performed by the first electronic apparatus and the second electronic apparatus, and the second server device performs the decryption, and the present disclosure is not necessarily limited thereto.

[0068] FIG. **2** is a block diagram showing a configuration of a computing device according to an embodiment of the present disclosure.

[0069] In detail, in the system shown in FIG. **1**, not only the apparatus that performs the homomorphic encryption, such as the first electronic apparatus or the second electronic apparatus, but also the device that operates the homomorphic ciphertext, such as the first server device, or the device that decrypts the homomorphic ciphertext, such as the second server device, may be referred to as the electronic apparatus. Such an electronic apparatus may be any of various devices such as a personal computer (PC), a laptop, a smartphone, a tablet, or a server.

[0070] Referring to FIG. **2**, an electronic apparatus **400** may include a communication device **410**, a memory **420**, a display **430**, a manipulation input device **440**, and a processor **450**.

[0071] Meanwhile, the electronic apparatus **400** may have a machine word size of N bits. Here, the machine word size may indicate the number of bits of data that the processor **450** may process at a single time. Here, the machine word size may be related to the number of bits in a hardware architecture of the electronic apparatus **400**. As an example, the machine word size may be related to the size of the registers, the size of an address bus, the size of a data bus, or the like.

[0072] In one or more embodiments, the electronic apparatus **400** may have a 32-bit or 64-bit hardware architecture. Here, the electronic apparatus **100** having the 32-bit hardware architecture may generally have a 32-bit machine word size, which is only an embodiment, and may have a machine word size of less than or equal to 32 bits (e.g., 16 bits or 8 bits) depending on the specific operation. In addition, the electronic apparatus **100** having the 64-bit hardware architecture may generally have a 64-bit machine word size, which is only an embodiment, and may have a machine word size of less than or equal to 64 bits (e.g., 32 bits, 16 bits, or 8 bits) depending on the specific operation.

[0073] Meanwhile, according to an embodiment of the present disclosure, the electronic apparatus **100** may have the 64-bit word size.

[0074] The communication device **410** may connect the electronic apparatus **400** to the external device (not shown), and may be connected to the external device not only via a Local Area Network (LAN) and the Internet, but also via a Universal Serial Bus (USB) port or a wireless communication port (e.g., Wi-Fi 802.11a/b/g/n, NFC, or Bluetooth). The communication device **410** may also be referred to as a transceiver.

[0075] The communication device **410** may receive a public key from the external device, and transmit the public key generated by the electronic apparatus **400** itself to the external device.

[0076] In addition, the communication device **410** may receive the message from the external device, and transmit the generated homomorphic ciphertext to the external device.

[0077] In addition, the communication device **410** may receive various parameters required for generating the ciphertext from the external device. Meanwhile, in implementation, the various parameters may be directly input from the user through the manipulation input device **440** described below.

[0078] In addition, the communication device **410** may receive a request for the operation on the homomorphic ciphertext from the external device and transmit its computed result to the external device. Here, the communication device **410** may receive the homomorphic ciphertext having a M*N-bit word size from the external device. For example, the communication device **410** may receive the homomorphic ciphertext having the 64-bit word size from the external device.

[0079] The memory **420** is a component for storing an operation system (0/S) for operating the electronic apparatus **400**, various software, data, or the like. The memory **420** may be implemented in any of various forms such as a Random-Access Memory (RAM), a Read-Only Memory (ROM), a flash memory, a hard disk drive (HDD), an external memory, or a memory card, and is not limited to any one thereof.

[0080] The memory **420** may store the message to be encrypted. Here, the message may include various credit information, personal information, or the like quoted by the user, and may also be information related to a usage history, such as location information, internet usage time information, or the like used by the electronic apparatus **400**.

[0081] In addition, the memory **420** may store the public key. If the electronic apparatus **400** is a device that directly generates the public key, the electronic apparatus **400** may store not only the secret key, but also various parameters required for generating the public key and the secret key.

[0082] In addition, the memory **420** may store the homomorphic ciphertext generated in a process described below. In addition, the memory **420** may also store the homomorphic ciphertext transmitted from the external device. In addition, the memory **420** may also store the operation result ciphertext, which is a result of an operation process described below.

[0083] In addition, the memory **420** may store the scaling factor for performing the operation on the homomorphic ciphertext. Here, the scaling factor may be one of values used to convert the homomorphic ciphertext to plaintext. In addition, homomorphic encryption may perform computations on the homomorphic ciphertext, and the scaling factor may be one of important parameters used to perform mathematical operations on the message in its homomorphic ciphertext state. Here, the scaling factor may be referred to by any of various terms such as the parameter.

[0084] The display **430** may display a user interface window for selection of functions supported by the electronic apparatus **400**. In detail, the display **430** may display the user interface window for the selection of various functions provided by the electronic apparatus **400**. The display **430** may be a monitor such as a liquid crystal display (LCD), or an organic light-emitting diode (OLED) display, and may also be implemented as a touchscreen capable of simultaneously performing a function of the manipulation input device **440** described below.

[0085] The display **430** may display a message requesting input of the parameter required for generating the secret key or the public key. In addition, the display **430** may display a message for selecting a message as an encryption target. Meanwhile, in implementation, the encryption target may be selected directly by the user or automatically selected. That is, the personal information or the like to be encrypted may be set automatically even if the user does not directly select the message.

[0086] The manipulation input device **440** may receive a function selection command and a control command for a corresponding function of the electronic apparatus **400** from the user. In detail, the manipulation input device **440** may receive the parameter required for generating the secret key or the public key from the user. In addition, the manipulation input device **440** may receive the message to be encrypted from the user.

[0087] The processor **450** may control each component included in the electronic apparatus **400**. The processor **450** may be configured as a single device, such as a central processing unit (CPU) or an application-specific integrated circuit (ASIC), or may be configured as a plurality of devices, such as central processing units (CPUs) and graphics processing units (GPUs).

[0088] Here, the processor **450** may have the machine word size that may process 64 bits of data at a single time.

[0089] The processor **450** may store the message to be transmitted in the memory **420** if the corresponding message is input. The processor **450** may homomorphically encrypt the message by using various set values and programs stored in the memory **420**. In this case, the public key may be used.

[0090] The processor **450** may generate and use the public key required to perform the encryption on its own, or may receive the public key from the external device and use the same. As an example, the second server device **300** performing the decryption may distribute the public key to other devices.

[0091] If the processor **450** generates the key on its own, the processor **450** may generate the public key by using a Ring-Learning With Errors (Ring-LWE) scheme. To describe in detail, the processor **450** may first set the various parameters and rings and store the same in the memory **420**. An example of the parameter may include a length of plaintext message bits, a size of the public key or the secret key, or the like.

[0092] The ring may be expressed by the following equation.

[00002] $R = Z_q[X] / f(x)$ [Equation2]

[0093] Here, R indicates the ring, Z.sub.q indicates a coefficient, and f(x) indicates an Nth polynomial.

[0094] The Ring indicates a set of polynomials having predetermined coefficients, and indicates a set in which addition and multiplication are defined between elements and which is closed under the addition and multiplication. The Ring may be referred to as the ring.

[0095] As an example, the ring indicates a set of the Nth polynomials having the coefficient Z.sub.q. In detail, if n is Φ(N), N indicates a polynomial which may be calculated as the remainder of dividing the polynomial by an Nth cyclotomic polynomial. (f(x)) indicates ideal of Z.sub.q[x] generated by f(x). The Euler totient function Φ(N) indicates the number of natural numbers coprime to N and smaller than N. If ΦN(x) is defined by the Nth cyclotomic polynomial, the ring may also be expressed by Equation 3 below.

[00003] $R = Z_q[X] / \Phi_N(x)$ [Equation3]

[0096] A secret key sk may be expressed as follows.

[0097] Meanwhile, the ring in Equation 3 described above has complex numbers in a plaintext space. Meanwhile, only a set whose plaintext space consists of real numbers among the sets of rings described above may be used to improve an operational speed of the homomorphic ciphertext.

[0098] If the ring is set up, the processor **450** may calculate the secret key sk from the ring.

[00004] $sk \leftarrow (1, s(x)), s(x) \in R$ [Equation4]

[0099] Here, s(x) indicates a random polynomial generated using a small coefficient.

[0100] In addition, the processor **450** may calculate a first random polynomial a(x) from the ring. The first random polynomial may be expressed as follows.

[00005] $a(x) \leftarrow R$ [Equation5]

[0101] In addition, the processor **450** may calculate the error. In detail, the processor **450** may extract the error from a discrete Gaussian distribution or a distribution having a statistical distance close thereto. This error may be expressed as follows.

[00006] $e(x) \leftarrow D^n_q$ [Equation6]

[0102] If even the error is calculated, the processor **450** may calculate a second random polynomial by performing the modulus operation on the error in the first random polynomial and the secret key. The second random polynomial may be expressed as follows.

[00007] $b(x) = -a(x)s(x) + e(x)(\mod q)$ [Equation7]

[0103] Finally, a public key pk may be set up to include the first random polynomial and the second random polynomial as follows.

[00008] $pk = (b(x), a(x))$ [Equation8]

[0104] The above-described key generation method is only an example, the present disclosure is not necessarily limited thereto, and the public key and the secret key may also be generated using another method.

[0105] Meanwhile, the processor **450** may control the communication device **410** to transmit the public key to other devices if the public key is generated.

[0106] In addition, the processor **450** may generate the homomorphic ciphertext for the message. In detail, the processor **450** may generate the homomorphic ciphertext by applying the previously generated public key to the message. Here, the processor **450** may generate the ciphertext to have a length corresponding to a size of the scaling factor Δ.

[0107] In addition, the processor **450** may store the homomorphic ciphertext in the memory **420** if the homomorphic ciphertext is generated, or control the communication device **410** to transmit the homomorphic ciphertext to another device based on a user request or a predetermined default command.

[0108] Meanwhile, according to an embodiment of the present disclosure, packing may be performed. If the packing is used in the homomorphic encryption, the plurality of messages may be encrypted as a single ciphertext. In this case, if the electronic apparatus **400** performs the operations between the respective ciphertexts, the operations on the plurality of messages may be processed in parallel as a result, thereby greatly reducing an operation burden.

[0109] In detail, if the message includes a plurality of message vectors, the processor **450** may also convert the plurality of message vectors into a polynomial that may be encrypted in parallel, then multiply the polynomial by the scaling factor, and use the public key to homomorphically encrypt the same. Accordingly, the processor **450** may generate the ciphertext in which the plurality of message vectors are packed.

[0110] In addition, if the homomorphic ciphertext is required to be decrypted, the processor **450** may apply the secret key to the homomorphic ciphertext to thus generate a polynomial-shaped decrypted text, and decode the polynomial-shaped decrypted text to thus generate the message. Here, the generated message may include the error as mentioned in Equation 1 described above.

[0111] In addition, the processor **450** may perform the operation on the ciphertext. In detail, the processor **450** may perform the operation such as addition or multiplication on the homomorphic ciphertext while maintaining its encrypted state. In detail, the processor **450** may process each of the homomorphic ciphertexts to be used in the operation by using a first function, perform the operation such as the addition or multiplication between the homomorphic ciphertexts processed using the first function, and process the homomorphic ciphertexts, on which the operation is performed, by using a second function, which is an inverse function of the first function. These first-function processing and second-function processing may utilize a linear transformation technique in a bootstrapping process described below.

[0112] Meanwhile, the electronic apparatus **400** may detect the data in the effective region from operation result data if the operation is completed. In detail, the electronic apparatus **400** may perform rounding processing on the operation result data to detect the data in the effective region. The rounding processing indicates rounding off of the message while the message is encrypted, and may also be referred to as the rescaling. In detail, the electronic apparatus **400** may remove a noise region by multiplying each component of the ciphertext by Δ.sup.−1, which is an inverse of the scaling factor, and rounding off the same. The noise region may be determined to correspond to the size of the scaling factor. As a result, the electronic apparatus **400** may detect the message in the effective region excluding the noise region. The rounding processing may be performed while the message is encrypted, an additional error may thus occur. However, a size of the error may be sufficiently small and thus ignored.

[0113] In addition, the electronic apparatus **400** may perform a bootstrapping operation on the ciphertext if an approximate message weight in the operation result ciphertext exceeds a threshold. In detail, the electronic apparatus **400** may generate the homomorphic ciphertext whose plaintext space is expanded by expanding a modulus of the operation result ciphertext, performing a first linear transformation on the homomorphic ciphertext whose modulus is expanded into a polynomial form, performing an approximate arithmetic on a first homomorphic ciphertext, which is converted into the polynomial form, by using a function set to approximate a modulated range of the plaintext, performing a second linear transformation on a second homomorphic ciphertext, on which the approximate arithmetic is performed, into a form of the homomorphic ciphertext, and performing a subtraction operation by subtracting the second homomorphic ciphertext, on which the second linear transformation is performed, from the homomorphic ciphertext whose modulus is expanded.

[0114] As described above, the electronic apparatus **400** may utilize the scaling factor Δ in a process of encrypting the plaintext message into the homomorphic ciphertext or in a process of operating the homomorphic ciphertext.

[0115] Meanwhile, in an embodiment of the present disclosure, a relationship between the ciphertext modulus and the scaling factor may be separated. In particular, this separation may be designed to maintain the same number of switching keys while making the most of the computation and memory budget of the electronic apparatus **400** by using the RNS moduli that include the plurality of moduli each corresponding to the machine word size.

[0116] According to an embodiment of the present disclosure, the electronic apparatus **400** may acquire the homomorphic ciphertext by using the Residue Number System (RNS) modulus including the plurality of moduli each having a size corresponding to the word size of the processor **450**. Here, the RNS modulus may be a value that defines an entire modulus space used in the operation of homomorphic encryption/the homomorphic encryption operation, and as described below, a RNS modulus Q may usually be expressed as a product of the plurality of moduli q.sub.1, q.sub.2, . . . , and q.sub.k.

[00009] $Q = q_1 \text{ .Math. } q_2 \text{ .Math. .Math. .Math. } q_k$ [Equation9]

[0117] According to an embodiment of the present disclosure, all operations may be performed in a Residue Number System (RNS) format within Cheon-Kim-Kim-Song (CKKS, RNS-CKKS). That is, all the homomorphic ciphertexts or keys may be decomposed using the RNS moduli, and each component may be computed based on the machine word size. Therefore, if the number of RNS components representing the homomorphic ciphertext may be reduced, the overall Fully Homomorphic Encryption (FHE) computation may be directly speeded up by the reduction ratio based on the ciphertext modulus. In particular, the number of operations such as (i) Number Theoretic Transform (NTT), Fast Basis Conversion (FBC), tensor operations, and external products may follow an approximate reduction ratio even after considering an additional cost. The FBC, tensor multiplication, and external products may be reduced by the same ratio. In particular, in an embodiment of the present disclosure, the plurality of moduli may use primes (word size primes) each corresponding to the machine word size. However, this configuration may limit available multiplicative depth compared to a case where a modulus value is set to be approximately equal to the scaling factor, i.e., A. The scaling factor Δ may typically vary between 20 and 120 depending on message precision (or target message precision).

[0118] FIG. **3** is a diagram showing a conventional RNS modulus and a modulus according to an embodiment of the present disclosure.

[0119] A conventional RNS modulus **310** may be composed of 14 moduli. Here, the plurality of moduli may have a plurality of sizes to perform various operations. For example, the plurality of moduli may include moduli for scaling the ciphertext (For CtS), moduli for evaluating the operations on the ciphertext (Kor EvalMod), moduli for general operations (For general computation), and moduli for converting the ciphertext (For StC). The conventional RNS modulus **310** may be composed of many moduli, and accordingly, there may be limitations in the size and complexity of the computation, which are relatively large and slow.

[0120] An RNS modulus **320** according to an embodiment of the present disclosure may include the plurality of moduli each corresponding to the machine word size. As an example, the RNS modulus **320** according to an embodiment of the present disclosure may be composed of 10 moduli, which is a smaller number than the conventional RNS modulus **310**. In particular, if a size of the moduli corresponds to the machine word size (e.g., 64 bits), hardware efficiency may be maximized, thus enabling the faster operation (e.g., 1.4 times).

[0121] In an embodiment of the present disclosure, a method may be used to rescale the plurality of moduli by a smaller scaling factor while using primes each corresponding to the machine word size. To this end, according to an embodiment of the present disclosure, the electronic apparatus **100** may perform the operation on the homomorphic ciphertext by using the rational rescaling. That is, the electronic apparatus **100** may use the rational rescaling, which is a tool to switch the ciphertext modulus into a non-divisor modulus. Here, the rational rescaling may refer to rescaling in which Q′ does not necessarily have to be a value divisible by Q if the modulus Q is rescaled to Q′. As an example, as shown in FIG. **4**A, the rational-scaling factor Δ may be used if a modulus Q **411** is scaled to a modulus Q′ **412**.

[0122] A rational rescaling procedure may rescale the input ciphertext by a rational number Q/Q′∈ custom-character, in the modulus Q, to convert the output ciphertext into modulus Q/(Q/Q′)=Q′, and may also rescale the scaling factor.

[0123] For given polynomials α∈R.sub.Q and Q′/Q, the rational rescaling is defined as the rescaling by a rational coefficient 'Q/Q', which may be computed by Equation 10 below.

[00010] $RS_{Q/Q'}(a) = RS_S(\text{Inv - } RS_R(a) \in \mathcal{R}_{\text{lcm}(Q, Q')}) \in \mathcal{R}_{Q'}$ [Equation10]

[0124] Here, R=1 cm(Q, Q′)/Q∈ custom-character, custom-character=1 cm(Q, Q′)∈ custom-character and 1 cm indicates the least common multiple.

[0125] An RS symbol may represent both the rational rescaling and integer rescaling (i.e., conventional rescaling by integers). The rational rescaling may be seamlessly extended to polynomials or ciphertext vectors.

[0126] It needs to be noted that inverse rescaling during the rational rescaling serves a completely different function from that performed during the key switching.

[0127] Theorem **1** below shows that a rational rescaling error is of the same format as those of modulus reduction (ModDown) and rescaling (RS) errors, and that the rational rescaling error is proportional to the number of RNS moduli removed. Therefore, it is important to minimize a RNS modulus factor of 1 cm(Q, Q′)/Q′ to reduce the error.

[Theorem **1**] (Accuracy of Rational Rescaling and Rational Rescale Correctness)

[0128] For given ciphertexts ct∈R.sub.Q.sup.2 and Q′|Q, [<RS.sub.Q/Q′(ct), sk>], Q′=Q′/Q.Math.[<ct, sk>]Q+e.sub.res may be satisfied.

[0129] Here, e.sub.res may be a rescaling error satisfying the following condition.

[00011]? $\text{.Math. ? .Math. ?} \le \frac{\ell}{2} \text{ .Math. (.Math. } s \text{ .Math. ? + 1)?}$ indicates text missing or illegible when filed

[0130] Here, 1 indicates the number of RNS blocks included in 1 cm(Q, Q′)/Q′, and S indicates the secret key.

[0131] Theorem **1** shows that the rational rescaling maintains its accuracy, and the error e.sub.res is constrained in proportion to 1, a size of the secret key ‖S‖i and a predetermined value.

[0132] FIG. **4**B is a diagram for describing the rational rescaling according to an embodiment of the present disclosure.

[0133] First, an initial modulus **421** may be (ct; Δ.sup.2) mod Q. Here, ct indicates the ciphertext and Δ.sup.2 indicates a scaling factor. An intermediate modulus **422** may be acquired by multiplying the initial modulus **421** by R. Here, R may be 1 cm(Q, Q′)/Q. The intermediate modulus **422** may be (R.Math.ct; RΔ.sup.2) mod 1 cm(Q, Q′). A final modulus **423** may be acquired by dividing the intermediate modulus **422** by R′. Here, R′ may be 1 cm(Q, Q′)/Q′. The final modulus **423** may be (ct′; Δ′:=RΔ.sup.2/R′) mod Q′.

[0134] That is, the rational rescaling is a technique for rescaling the modulus of the ciphertext from Q to Q′ by using the rational numbers, and in this process, the accuracy and efficiency of the ciphertext operation may be improved by utilizing 1 cm(Q, Q′) based on the least common multiple.

[0135] FIGS. **5** and **6** are diagrams for describing a conventional key switching operation. The key switching operation is a technique for converting the ciphertext by using another secret key, and is one of the important operations in the homomorphic encryption (HE).

[0136] A modulus of the ciphertext ct is defined as Q, and a switching key swk may be defined as mod Q.sub.swk=PQ.

[0137] As shown in FIG. **5**, the key switching operation includes the following processes: 1) modulus expansion (ModUp), 2) tensor operation, and 3) modulus reduction (ModDown). 1) Modulus expansion (ModUp) is a process of expanding the original ciphertext modulus Q to a larger modulus of mod Q.sub.swk=PQ. The extended ciphertext may perform more the operations by including information of an added dimension. 2) Tensor operation may apply the tensor operation using the switching key to the extended ciphertext. Accordingly, a process of applying a new key may be performed. 3) Modulus reduction (ModDown) may again reduce the ciphertext modulus Q, thereby finally acquiring the ciphertext in which the key is switched.

[0138] Meanwhile, the rational rescaling is a process of converting the ciphertext including the message after the tensor operation on the scaling factor from Δ.sup.2 to Δ.sup.2.Math.Q′/Q≈Δ. This method is different from an existing method Δ.sup.2/ql≈Δ. In particular, if the scaling factor is adjusted by utilizing the rational rescaling, the resulting ciphertext modulus Q may not be a divisor of a switching key modulus PQ. In this case, an existing key switching procedure is unable to be applied because the modulus of the ciphertext is modified, which may not be compatible with an existing gadget decomposition method.

[0139] Q needs to be a divisor of Q.sub.swk to perform the key switching. However, more moduli may be needed to cover all the possible moduli Q. This need may result in a limit to an increase in the moduli. In detail, as shown in **610** of FIG. **6**, a setting Q|Q.sub.swk may be defined to hold for all the possible Q, in which case, moduli of various sizes are required. As shown in **620** of FIG. **6**, a plurality of key sets may be used. However, in this case, the number of switching keys may be increased, and a communication cost may be increased if the keys are transmitted. If a specific circuit is not predetermined, a very large number of keys need to be transmitted for general circuit evaluation, which is inefficient. In addition, in Fully Homomorphic Encryption (FHE), a size of the switching key may range from hundreds of megabytes (MB) to several gigabytes (GB), and preparing the plurality of keys may be burdensome.

[0140] According to an embodiment of the present disclosure, to address the above-described issue, the processor **450** may perform the key switching operation by using the intermediate modulus.

[0141] FIG. **7** is a diagram for describing a process of performing the key switching operation by using the intermediate modulus according to an embodiment of the present disclosure.

[0142] First, the processor **450** may upscale an RNS modulus **710** to generate an intermediate modulus **720**. Here, the intermediate modulus may be a divisor of the switching key modulus. In addition, the processor **450** may perform the key switching operation on the intermediate modulus **720**. In addition, the processor **450** may perform the rescaling on the intermediate modulus **420** on which the key switching operation is performed to acquire an output modulus **730**.

[0143] In detail, given the ciphertext acquired by encrypting the message Δm+e in modulus Q, the intermediate modulus Q.sub.int is Q.sub.int≥Q, and one of moduli satisfying Q.sub.int|PQ may be selected. In addition, the modulus of the ciphertext may be converted into the intermediate modulus Q.sub.int, and then the key switching may be performed thereon. This process is similar to the rational rescaling, which may be operated by first converting the ciphertext into the intermediate modulus and then applying the key switching thereto.

[0144] The ciphertext converted as a result of the key switching may be encrypted using a new secret key, and the message may be adjusted as in Formula 11 below.

[00012] $(\frac{Q_{\text{int}}}{Q})$ .Math. $(m + e)$ + ??indicates text missing or illegible when filed

[0145] Here, e′ indicates an error that occurs during rescaling and key switching processes.

[0146] In addition, the processor **450** may perform the rescaling process to switch the modulus back to the output modulus Q. In this process, the error may be maintained within an original error value and a rescaled error range.

[0147] Extending the above method, the switching key may be prepared in modulus Q.sub.swk, which is mostly composed of the moduli having the machine word size. Depending on the scaling factor, the rational rescaling may be performed on the ciphertext modulus from Q to: Q′≈Q/Δ, and during this process, the modulus of the ciphertext may be temporarily converted into Q.sub.int|Q.sub.swk to perform the key switching.

[0148] Meanwhile, a process of temporarily converting the modulus Q to the intermediate modulus Q.sub.int may cause an additional NTT operation. This problem may also occur even in case of coupling this process with the ModUp operation. In detail, the ModUp operation may require the same number of NTT operations for coefficient input and NTT-evaluated format input. The reason is that although some NTT-evaluated inputs may be reused, maintaining the result in an NTT-evaluated state still requires the same operations. Therefore, performing the temporary modulus switching requires additional (l+1) NTT operations, where l+1 may indicate the number of moduli included in the modulus Q.

[0149] In addition, the temporary modulus switching and the rational rescaling may incur an additional Hadamard multiplication cost. This cost may be quite large, especially if the factors of the two moduli do not overlap much.

[0150] In detail, assuming that the number of factors of Q is ![](custom-character=l.sub.1+l.sub.2 and the number of factors of Q.sub.int is l′=l.sub.2+l.sub.3, then exactly l.sub.2 factors may overlap. In this case, a cost incurred by the conversion from Q to Q.sub.int may be increased as shown in Formula 12 below.

[00013] $O(N(\ell + (\ell - l_2)$ .Math. $l^{'}))$ [Formula12]

[0151] Here, n indicates a ring dimension.

[0152] In an embodiment of the present disclosure, the processor **450** may perform a modulus resurrection technique to avoid generating more switching keys or incurring additional key switching costs. In detail, this method may be a way to reuse some factors of the ciphertext modulus. A basic condition of the moduli included in the RNS modulus is that these moduli need to be relatively prime. While satisfying this condition, the processor **450** may resurrect and reuse some elements of the top ciphertext modulus previously scaled and then lost. In this way, the plurality of moduli included in the RNS modulus may still maintain a prime relationship with each other, while the ciphertext modulus may be maintained as the divisor of the switching key modulus. This method may be a method of fusing the temporary ciphertext modulus chain described above with an existing ciphertext modulus chain.

[0153] In an embodiment of the present disclosure, a part of the ciphertext modulus that is specifically resurrected may be referred to as the sprout modulus. The sprout modulus may be flexible and have a small size. For all possible sprouts, the present disclosure defines their common multiple, which may be referred to as a top sprout r.sub.top. That is, the present disclosure may set the sprout r.sub.top having the maximum size and select the sprout modulus from divisors of the sprout having the maximum size. Each ciphertext modulus may be expressed as a product of the sprout modulus and different word-sized NTT primes, i.e., unit moduli. Accordingly, the ciphertext modulus including the sprout modulus may be referred to as a grafted modulus.

[0154] In detail, the plurality of moduli included in the RNS modulus according to an embodiment of the present disclosure may include the sprout modulus composed of a product of primes whose size is less than or equal to the machine word size. That is, the RNS modulus Q may be expressed by Equation 13 below.

[00014] $Q = q_0 \times$ .Math. $\times q_l \times r$ [Equation13]

[0155] Here, q.sub.0 to q.sub.1 indicate the plurality of moduli each having the size corresponding to the machine word size, and r indicates the sprout modulus. For example, if the plurality of moduli are 60-bit primes, r may be expressed as one 30- to 59-bit prime or one 30-bit prime×one 31- to 59-bit prime, depending on its size. In addition, for any 60k+s bit modulus Q, the sprout modulus r may be k 60-bit primes if s=0, may be

expressed as k-1 60-bit primes, one 30-bit prime, and one (s+30)-bit prime if (1≤s≤29), and may be expressed as a product of k 60-bit primes and one s-bit prime if !(30≤s≤59).

[0156] According to an embodiment of the present disclosure, the sprout modulus may be designed to be used flexibly by separating a specific part, as shown in FIG. **8**. For example, the sprout modulus may be expressed by Equation 14 below.

[00015] $r = r_1 \ .\text{Math.}\ r_2 \ .\text{Math.}\ r_3$ [Equation13]

[0157] Here, each of r.sub.1, r.sub.2, and r.sub.3 may be a 20-bit NTT prime. This structure may make Q=0, 20, 40 mod 60 possible. That is, this structure may make an operation possible where the remainder is set to a specific value (0, 20, 40). Here, q.sub.i may be 2.sup.60.

[0158] As another example, if the sprout modulus is expressed as r=2.sup.60 and q.sub.i is 2.sup.60, the modulus Q may be freely selected to any bit. That is, a remainder operation in a binary mode may be made possible.

[0159] FIG. **9** is a diagram showing a scenario in which the modulus including the sprout modulus is reduced according to an embodiment of the present disclosure. According to an embodiment of the present disclosure, each multiplication operation may progressively consume the moduli by the scaling factor while maintaining as many machine-sized moduli as possible. In addition, the number of non-empty gadget blocks in an entire process may be optimized.

[0160] In addition, the processor **450** may generate the intermediate modulus by resurrecting the sprout modulus and acquire the output modulus by rescaling the intermediate modulus. Here, the intermediate modulus may be a multiple of the output modulus. In detail, as shown in FIG. **10**, a sprout modulus part (triangular part) included in an input modulus **1010** may be resurrected to generate an intermediate modulus **1020**. In addition, the intermediate modulus **1020** may be adjusted to be a multiple of an output modulus **1030**. In this way, the processor **450** may perform the rescaling using an existing method. For example, if the sprout modulus is r=r.sub.1.Math.r.sub.2.Math.r.sub.3, and each of r.sub.1, r.sub.2, and r.sub.3 are the 20-bit NTT primes, the input modulus **1010** may be q.sub.0q.sub.1q.sub.2q.sub.3r.sub.1 and the scaling target may be 2.sup.40. The input modulus **1010** may be q.sub.0q.sub.1q.sub.2q.sub.3r.sub.1≈2.sup.60, and the processor **450** may multiply the input modulus **1010** by r.sub.2 to generate q.sub.0q.sub.1q.sub.2q.sub.3r.sub.1r.sub.2≈2.sup.280, which is the intermediate modulus **1020**. In addition, the processor **450** may acquire q.sub.0q.sub.1q.sub.2r.sub.1r.sub.2≈2.sup.220, which is the output modulus **1030**, by dividing the intermediate modulus **1020** by q.sub.3 to adjust (rescale) the modulus. In this way, flexible rescaling may be possible while maintaining the modulus having the machine word size.

[0161] According to an embodiment of the present disclosure, the processor **450** may perform the key switching operation on the homomorphic ciphertext by using the sprout modulus. In one or more embodiments, the processor **450** may perform the key switching operation by using the sprout modulus through the process shown in FIG. **11**. In detail, the processor **450** may acquire an intermediate modulus **1120** by performing inverse rational rescaling (i.e., inverse conversion of the rational rescaling) on an input modulus **1110**. In addition, the processor **450** may acquire an expanded modulus **1130** by performing the ModUp operation on the intermediate modulus **1120**. In addition, the processor **450** may perform the key switching operation on the expanded modulus **1130** and then perform the ModDown operation to thus acquire an output modulus **1140**. Through this process, unnecessary operations may be reduced while performing the key switching, and efficiency may be increased by utilizing an existing sprout modulus.

[0162] To describe the present disclosure in more detail, the maximum ciphertext modulus may be defined as Q.sub.max=q.sub.0 . . . Q.sub.L−1.Math.r.sub.top, where q.sub.i's may be the unit modulus having a size similar to a machine word size (w=2.sup.w). In addition, each ciphertext modulus Q may be expressed as Q=q.sub.0 . . . q.sub.l−1.Math.r including some of the sprout moduli r. The switching key modulus may be expressed as PQ.sub.max, where P may generally be set to P=p.sub.0 . . . p.sub.K−1. Here, the respective p.sub.i are the unit moduli which are coprime and satisfy gcd(P, Q.sub.max)=1, and P and Q.sub.max may thus satisfy their coprime relationship. Each sprout modulus r is the divisor of the top sprout modulus r.sub.top, and thus may satisfy r|r.sub.top. This allows the following Formula 14 to be established.

[00016] $Q \ .\text{Math.}\ Q_{\max} \ .\text{Math.}\ PQ_{\max}$ [Formula14]

[0163] Therefore, the key switching operation may be performed using the same number of switching keys as the key switching using the existing method. In particular, the sprout modulus r.sub.top having the maximum size may be selected to be a size similar to the machine word size w=2.sup.w. Below is an example of a sprout modulus.

[0164] (Example 1) Assuming that the unit moduli q.sub.i's are 60-bit NTT primes and the top sprout modulus is defined as r.sub.top=r.sub.0r.sub.1. Here, each of r.sub.0 and r.sub.1 may be a 30-bit NTT prime.

[0165] Each sprout modulus r may be expressed as r∈{1, r.sub.0, r.sub.0r.sub.1}. Here, r.sub.0r.sub.1 may only appear in a maximum modulus Q.sub.max.

[0166] If the scaling is performed in modulus Q=q.sub.0 . . . q.sub.l−1.Math.r, the scaling may be continuously performed in units of approximately 30 bits. For example, if r includes r.sub.i, the rescaling may be performed using r.sub.1, and if r=1, the rational rescaling may be performed using q.sub.l/r.sub.1≈2.sup.30.

[0167] A multiplication algorithm resulting from this process may be specified in Algorithm **1** as shown in FIG. **12**. According to an embodiment of the present disclosure, there are two major differences compared to an existing RNS-CKKS multiplication scheme. The first difference is that the ciphertext modulus is modified to the grafted modulus including the sprout modulus, and the second difference is that the rescaling is replaced with the rational rescaling using the existing method, which may be converted into the appropriate modulus Q′. Therefore, in the present disclosure, only a small number of unit moduli may be removed in a rational rescaling process, and the size of the error may be maintained at almost the same level as the error that occurs in an existing rescaling method. A detailed key switching procedure may refer Algorithm **2** shown in FIG. **13**, and a method of selecting the output modulus Q′ may refer Algorithm **4** shown in FIG. **15**.

[0168] The output ciphertext modulus Q′ needs to be approximately Q/Δ to maintain the scaling factor Δ. Therefore, the sprout modulus that is generally selected may vary depending on a rescaling factor to be used. For example, the sprouts used in the above example may be designed to only allow the rescaling in multiples of ω/2=30 bits. This configuration may result in limiting the scaling factor that may be used in case of evaluating a circuit. Therefore, it is important to select the sprout modulus that is appropriate for the given circuit. The present disclosure describes below in detail a universal sprout, which is general and may be used for general purposes.

[0169] The processor **450** may perform the key switching by using the sprout modulus.

[0170] In particular, the key switching procedure and a relinearization procedure may be performed in the same manner as an existing non-grafted RNS-CKKS scheme. The reason is that all the ciphertext moduli Q are maintained as divisors of a switching key modulus PQ.sub.max. In the present disclosure, the maximum ciphertext modulus Q.sub.max may be decomposed into d.sub.num blocks, each having a similar size, such as Q.sub.0, . . . , and Q.sub.dnum−1. These blocks may be referred to as gadget blocks, and here be set to satisfy P≥max.sub.iQ.sub.i.

[0171] The sprout modulus may be disposed in a single gadget block. In addition, efficiency of the key switching may vary depending on the gadget decomposition method. For example, the sprout modulus may be disposed in the top gadget block Q.sub.dnum−1. If conditions αi≤ ◻custom-character<α(i+1)−1 and i<dnum−1 are satisfied, the number of gadgets for a ciphertext modulus Q=r.Math.q.sub.0q.sub.1 . . . ◻custom-character may be greater than the optimal value of d= ⌈◻custom-character+1)/α⌉ . Rearranging these conditions may be as shown in Equation 15 below.

[00017]

$Q = q_0 q_1 \ .\text{Math.}\ q_{\ell - 1} \ .\text{Math.}\ r = (q_0 \ .\text{Math.}\ q_{\ -1}) \ .\text{Math.}\ (q_{\ (d-1)} \ .\text{Math.}\ q_{\ell - 1}) \ .\text{Math.}\ r = Q_0 \ .\text{Math.}\ Q'_{d-1} \ .\text{Math.}\ Q'_{\text{dnum} - 1},$ [Equation15]

[0172] Here, Q′.sub.d−1 indicates a divisor of Q.sub.d−1, and may satisfy Q′.sub.dnum−1=r. The number of gadget blocks in the ciphertext modulus may directly affect the key switching costs. This effect may be increased almost linearly relative to the number of gadget blocks. In the above case,

the gadget block Q_{dnum-1} may only be used to include the sprout modulus. Therefore, a total number of blocks used may be increased from an existing d to d+1. As a result, a key switching speed may be slowed by approximately (d+1)/d times. In particular, if the modulus Q is small, this ratio may become large to be non-negligible.

[0173] To achieve the optimal efficiency, the sprout modulus may be disposed at the bottom gadget block to minimize the number of non-empty gadget blocks. This process may be performed as shown in Equation 16 below.

[00018] $Q_0 = r_{top}$ .Math. q_0 .Math. q_{-2}, [Equation16] $Q_i = q_{i-1} q_i$ .Math. $q_{(i+1)-2}$ for $1 \le i \le$ dnum - 1

[0174] Here, L+1=dnum.Math.α.

[0175] A grafted ciphertext modulus including the sprout modulus may be expressed by Equation 17 as follows.

[00019]

$$Q = r \text{ .Math. } q_0 q_1 \text{ .Math. } q_{\ell-1} = (r \text{ .Math. } q_0 \text{ .Math. } q_{-2}) \text{ .Math. } (q_{-1} \text{ .Math. } q_{-2}) \text{ .Math. } (q_{(d-1)-1} \text{ .Math. } q_{\ell-1}) = Q_0' \text{ .Math. } Q_1 \text{ .Math. } Q_{d-2} \text{ .[}$$

?indicates text missing or illegible when filed

[0176] Here, 1≤⬤custom-character≤L, Q.sub.0′ indicates a divisor of Q′, Q′.sub.d−1 indicates a divisor of Q.sub.d−1, and d=|(⬤custom-character +1)/α|. This condition may minimize the number of gadgets for ⬤custom-charactereach ciphertext modulus Q.

[0177] The key switching procedure, which follows the key switching procedure of the existing RNS-CKKS and applies the grafted ciphertext modulus including the sprout modulus, may be described in Algorithm **2** in FIG. **13**. In addition, accuracy of the key switching in the grafted ciphertext modulus may be ensured by accuracy of the rational rescaling.

[0178] According to an embodiment of the present disclosure, the processor **450** may adjust the plurality of homomorphic ciphertexts having different moduli and scaling factors by using the rational rescaling. In detail, modulus adjustment may be a concept corresponding to level adjustment in RNS-CKKS. Previous RNS-CKKS implementations use a fixed scaling factor for each multiplicative depth, and current RNS-CKKS implementations do not use a fixed scaling factor for each modulus. According to an embodiment of the present disclosure, before performing the addition or multiplication, the processor **450** may adjust two ciphertexts having different moduli and scaling factors.

[0179] It may be assumed that a ciphertext ct(b, a)∈R.sub.Q.sup.2 satisfies Equation 18 below.

[00020] $b + a$ .Math. $s =$ .Math. $m + e(\bmod Q)$ [Equation18]

[0180] Here, Δ indicates the scaling factor, and Q may be composed of the plurality of unit moduli q.sub.i and the sprout modulus r.

[0181] This ciphertext may be converted into the ciphertext having the smaller modulus Q′ and a new scaling factor Δ′. To this end, a level adjustment technique may be applied to perform an adjustment operation. This configuration may be confirmed through Algorithm **3** shown in FIG. **14**. This adjustment process may be coupled within the circuit, similar to an existing level adjustment technique. This process may be referred to as circuit optimization.

[0182] In the present disclosure, the processor **450** may first select an appropriate intermediate modulus Q.sub.mid. This intermediate modulus is a divisor of Q and may satisfy conditions of Q≥Q.sub.mid>Q′ and Q.sub.mid>Q′.Math.Δ. The processor **450** may then reduce the ciphertext modulus to Q.sub.mid by removing some components. The processor **450** may then convert the reduced ciphertext modulus to the final modulus Q′ by multiplying the same by an integer constant and then performing the rational rescaling on its result.

[0183] An output ciphertext ct′=(b′, a′)∈R.sub.Q.sup.2 may satisfy b′+a′.Math.s=Δ′.Math.m+e.sub.adj.

[0184] Here, an adjusted error e.sub.adj may be configured as shown in Equation 19 below.

[00021] $e_{adj} = \frac{'e}{} + \frac{Q'\delta\Delta\, m}{Q_{mid}} + \frac{Q'e}{Q_{mid}} + e_{res}$ [Equation 19]

[0185] Here, δ indicates a rounding error that occurs from the integer constant, |δ|≤½, and e.sub.res may be an error that occurs in the rational rescaling process (rescaling error).

[0186] In the present disclosure, e.sub.adj may be maintained to a sufficiently small value by satisfying a condition Q.sub.mid>Q′.Math.Δ.

[0187] As mentioned above, it is important to select the appropriate sprout modulus in case of evaluating the specific circuit. That is, the given sprout modulus may not be used in another circuit, and may not be universally applicable.

[0188] The sprout modulus used in Example 1 above may only be rescaled to the multiples of 30 bits. Another method may utilize three 20-bit NTT primes, or to use 10-, 20-, or 30-bit NTT primes, thereby setting the rescaling to be possible for only multiples of 10 or 20 bits.

[0189] The following description describes a universal sprout that may be used independently of the scaling factor or the circuit. First, the present disclosure proposes the definition of the universal sprout and three candidates. Second, the present disclosure introduces a method for utilizing the RNS modulus if the sprout includes a power-of-two factor. Third, the present disclosure proposes a method for efficiently implementing the universal sprout in a machine architecture.

[0190] To overcome a limitation of the scaling factor caused by the selection of the sprout modulus, the present disclosure introduces the universal sprouts that may be used for any scaling factor. The description starts with the simplest example. This example may be viewed as a hybrid implementation combining a binary modulus and RNS-based CKKS.

[0191] (Example 2) In one or more embodiments, q.sub.i's may consist of nine 60-bit NTT primes, and the top sprout modulus r.sub.top may be set to 2.sup.60. Accordingly, each sprout modulus r may be expressed as a power-of-two integer that divides r.sub.top=2.sup.60. That is, each ciphertext modulus close to an integer bit size may be expressed by Equation 20 below.

[00022] $Q = q_0$ .Math. $q_{\ell-1}$ .Math. $2 \approx (60$ .Math. $\ell +$) - bit modulus [Equation 20]

[0192] Here, the maximum modulus (Top Modulus) Q.sub.max may be expressed as in Equation 21 below.

[00023] $Q_{max} = q_0$ .Math. q_{L-1} .Math. 2^{60} [Equation 21]

[0193] If the ciphertext modulus is approximately (60.Math.⬤custom-character+γ) bits, the rational rescaling may be performed using ⬤custom-character. . . ⬤custom-character.Math.2.sup.(γ−γ′) to convert the same into a new ciphertext modulus of the same number of bits. This rational rescaling may be performed regardless of γ≥γ′. For example, to rescale the ciphertext modulus from Q=q.sub.0 . . . ⬤custom-character.Math.2.sup.35 to approximately 36 bits, the rational rescaling may be performed using ⬤custom-character/2.sup.24≈2.sup.36, thereby acquiring a new ciphertext modulus Q′=q.sub.0 . . . ⬤custom-character.Math.2.sup.59.

[0194] As shown in Example 2 above, if a sprout (i.e., divisor of r.sub.top) may approximately represent all bit lengths from 1 bit to ω bits, this sprout may be referred to as the universal sprout in the present disclosure. The universal sprout allows the rescaling to any bit length and may be used independently of the circuit.

[0195] However, it may be inefficient to directly handle 2.sup.60 as the modulus in a 64-bit system. To deal with this inefficiency, it is necessary to embed ⬤custom-character/2.sup.60⬤custom-character by using a larger modulus, or use a real/complex-based Discrete Fourier Transform (DFT) instead of the NTT. However, this method requires much higher precision than a fixed-point operation, and may thus be difficult to implement and inefficient. In general, handling 2.sup.60 as the modulus may be approximately 3 times more expensive than handling 60-bit NTT primes.

[0196] Therefore, the description introduces more practical and universal sprouts in Example 3 and Example 4 below. Such a sprout may be seen as an extension of power-of-two sprouts in Example 2. Even including some power-of-two factors, such a sprout is much smaller than the machine word size, which may be handled much more easily. A cost of processing the sprout may be approximately twice as expensive as a cost of processing the word-sized NTT prime.

[0197] (Example 3) If q.sub.i's are 61-bit NTT primes and r.sub.top=2.sup.15.Math.r.sub.1.Math.r.sub.2, where r.sub.1 may be a 16-bit NTT prime and r.sub.2 may be the 30-bit NTT prime. In general, r.sub.1 and r.sub.2 may be selected as follows.

[00024] $r_1 = 2^{16} + 1, r_2 = 2^{30} - 2^{18} + 1$ [0198] r.sub.1 and r.sub.2 may be NTT primes appropriate for a ring dimension N≤2.sup.15.

[0199] Each sprout r may be expressed by Equation 22 below.

[00025] $r = 2$ .Math. $r_1^{\ 1}$ .Math. $r_2^{\ 2}$ [Equation 22]

[0200] Here, 0≤α≤15 and β.sub.1∈{0, 1}.

[0201] The sprouts may represent all bit lengths from 1 bit to 60 bits, {2.sup.1, . . . , 2.sup.15, r.sub.1, r.sub.1.Math.2.sup.1, . . . , r.sub.1.Math.2.sup.13, r.sub.2, r.sub.2.Math.2.sup.1, . . . , r.sub.2.Math.r.sub.1.Math.2.sup.15} may be the universal sprouts.

[0202] In the previous example, the rational rescaling may be performed for any bit length. For example, assuming that r=2.sup.13 and r.sub.2≈2.sup.43 in the ciphertext modulus Q=q.sub.0 . . . ▨custom-character.Math.r, r is divided by r.sub.2/2 to acquire the ciphertext modulus Q′ in Equation 23 below to perform 15-bit rescaling.

[00026] $Q' = q_0$ .Math. $q_{\ell-1}$ .Math. r' $r' = \frac{2^{13} \text{.Math. } r_2}{r_2/2} \approx 2^{14}$ [Equation 23]

[0203] To additionally perform 34-bit rescaling, r′ may be divided by 2.sup.3.Math.▨custom-character/r.sub.2 to acquire the ciphertext modulus Q″ in Equation 24 below.

[00027] $Q'' = q_0$ .Math. $q_{\ell-2}$ .Math. r'' $r'' = \frac{q_\ell \text{.Math. } 2^{14}}{2^3 \text{.Math. } q_\ell/r_2} = 2^{11}$ .Math. $r_2 \approx 2^{41}$ [Equation 24]

[0204] This equation may indicate that the rational rescaling of larger sizes is also possible.

[0205] (Example 4) If q.sub.i's are the 61-bit NTT primes and r.sub.top=2.sup.19.Math.r.sub.1.Math.r.sub.2, where r.sub.1 may be a 19-bit NTT prime and r.sub.2 may be a 23-bit NTT prime. In general, r.sub.1 and r.sub.2 may be selected as follows. [0206] r.sub.1=786,433, r.sub.2=5,767,169

[0207] r.sub.1 and r.sub.2 may be NTT primes appropriate for a ring dimension N≤2.sup.17.

[0208] Such a sprout may not be the universal sprout because its log value does not include a decimal point such as log.sub.2r.sub.1≈19.584 or log.sub.2 r.sub.2≈22.459. However, such a sprout is close to 261, and may still be usable in a similar way. For example, each sprout r may be expressed by Equation 25 below.

[00028] $r = 2$ .Math. $r_1^{\ 1}$ .Math. $r_2^{\ 2}$ [Equation 25]

[0209] Here, 0≤α≤20 and β.sub.i∈{0,1}. An appropriate sprout may be found within up to 0.5 bits of an actual bit length. That is, the rational rescaling may be performed in the same way as the previous example, and the rescaling of various sizes may be performed by utilizing the flexibility of the sprout.

[0210] Theorem **2** below summarizes a rescalable condition related to sprout selection. If the sprout is the universal sprout, the rational rescaling may be performed using any bit size.

[0211] [Theorem **2**] It is assumed that the moduli q.sub.i of the ciphertext modulus satisfy the following range.

[00029] $q_i \in [2\ (1-\ ), 2\ (1+\ )]$

[0212] Here, η>0 and |i∈⌈L−1⌈.

[0213] In addition, it is assumed that the maximum spout moduli are given as follows.

[00030] $r_{top} = r_0$?.Math. $r_?$?indicates text missing or illegible when filed

[0214] It is assumed that r|r.sub.top exists for every positive integer γ≤w, and thus satisfies the following.

[00031] $r \in 2$ .Math. $[1-\ , 1+\ ]$

Here, ϵ>0.

[0215] Here, for any ciphertext modulus Q, the rational rescaling may be performed on the ciphertext by using the sprout modulus, as shown in Formula 26 below.

[00032] 2 .Math. $1 \pm (n\ \ + 2\ ) + O(\ ^2 + \ ^2)$ [Formula 26]

[0216] Here, n=⌈δ/w⌈, and the rescaling may be performed for all positive integers δ<log.sub.2Q.

[0217] The sprout selected in each of Examples 2 and 3 is the universal sprout, which satisfies the assumption of Theorem **2** and may satisfy ϵ<2.sup.−13. In addition, sufficiently many 61-bit primes may exist within the following range.

[00033] $[2\ (1 - 2^{-20}), 2\ (1 + 2^{-20})]$

[0218] Therefore, such a sprout has universal rescalability. That is, the rational rescaling may be performed on the ciphertext by using 2.sup.δ.Math.(1±2.sup.−12) for all the positive integers δ. Here, δ is a natural number smaller than a current ciphertext modulus, and the moduli may be grafted together with the sprout.

[0219] In more general, the universal sprout may be used independently of the parameter selection, such as the operation, the message precision, or the ring dimension. If the unit moduli that follow the assumption of Theorem **2** is selected, the grafted ciphertext modulus Q′ output from Algorithm **1** in FIG. **12** may be selected together with the universal sprout. Universality may be ensured, and |log.sub.2 r′−log.sub.2Δ| may thus maintain the sufficiently small value.

[0220] Finally, although the sprout used in Example 4 is not the universal sprout, this sprout may be utilized because the sprout may approximately represent all integer bits or integers within ±0.5 bits.

[0221] As described in Example 2, the most intuitive way to set the sprout modulus r.sub.top may be to select a power-of-two machine word size. A more general universal sprout introduced in each of Examples 2 to 4 described below may also need to include some power-of-two factors to achieve the universality. In this case, the existing CKKS scheme using power-of-two moduli may be seamlessly utilized in the Residue Number System (RNS) scheme.

[0222] To this end, processes involving the power-of-two moduli may need to be considered. In particular, the processes may include the following procedures involving modulus-changing operations:

[0223] Fast Basis Conversion (FBC), Inverse Rational Rescale (or the inverse rescaling, Inv-RS), modulus expansion (ModUp), modulus reduction (ModDown), and Rational Rescale (or the rational rescaling, RS).

[0224] These processes may be designed to be performed only between bases that are originally relatively prime. Therefore, in the next step, the description describes a method for implementing the RNS operation including the power-of-two factor, which is possible even in the grafted RNS-CKKS scheme by expanding the existing algorithm. The present disclosure starts from a case where the expansion inherently occurs.

[0225] First, Fast Basis Conversion (FBC) may be defined and proven based on Chinese Remainder Theorem (CRT) between pairwise relatively

prime factors of the input modulus and between factors of P. That is, P and Q do not necessarily need to be relatively prime. More precisely, it may be assumed that P and Q may be factorized as described above. For example, the modulus Q may be expressed as follows.

[00034] $Q = q_0$ .Math. $q_{\ell-1}$ .Math. 2

[0226] Here, if FBC is applied to a polynomial $\alpha \in R.sub.Q$, a CRT factor of each P may be computed as in Formula 2.sup.7.

[00035] $\underset{j \in [\ell+1]}{.Math.}$ $[(a \mod q_j) .Math. \hat{q}_j^{-1}]q_j \mod q_j$ [Formula 27]

[0227] Here, {tilde over (q)}j=Q/q.sub.j, which may be set to ☐custom-character:=2.sup.β.

[0228] Second, the other four modulus conversion procedures may also be defined and proven based on Chinese Remainder Theorem (CRT). However, in this case, these procedures may be defined based on the pairwise relatively prime factors in a product of the input modulus and the output modulus. If the power-of-two factor is not changed during the modulus conversion process, the factors included in all the operations may still remain the pairwise relatively prime. Therefore, the factors may be operated in the same way as the existing method.

[0229] If the above conditions are met, the following modulus conversion operations may be seamlessly extended: [0230] Inverse Rational Rescale (or inverse rational rescaling, Inv-RS) [0231] Modulus Expansion (ModUp) [0232] Modulus Reduction (ModDown) [0233] Rational Rescale (or rational rescaling, RS) (if performed together with the integer factor)

[0234] These operations may be performed between the input modulus Q and the output modulus Q′, and need to satisfy the following conditions.

[00036] $\mathrm{ord}_2 Q = \mathrm{ord}_2 Q^2$

[0235] That is, the power-of-two order of Q and Q may need to be the same.

[0236] However, in the grafted RNS-CKKS scheme, Inv-RS, ModDown, and RS may be used between moduli having different power-of-two factors. In the present disclosure, it may be noted that the ModUp operation is always applied after Inv-RS. This sequence may serve to adjust the power-of-two factor to the same value as the switching key modulus, and fill the gadgets. This process is described in Algorithm **2** in FIG. **13**.

[0237] The rational rescaling having integer coefficients may be a specific case of ModDown. Therefore, in the present disclosure, Inv-RS and ModDown may be focused on. Algorithm **5** in FIG. **16** and Algorithm **6** in FIG. **17** specify a method for performing the polynomial conversion on the intermediate modulus by using the above efficient extension method.

[0238] In Algorithm **5** in FIG. **16**, the intermediate polynomial needs to satisfy Equation 28 below.

[00037] $b \equiv 2$ .Math. $a \bmod(\tilde{Q}$ .Math. $2^+)$ [Equation 28]

[0239] Here, {tilde over (Q)} indicates the intermediate modulus, and β and γ indicate power-of-two adjustment factors. An Inv-RS operation having an integer coefficient {tilde over (R)} may then be applied. {tilde over (R)}, which is an odd number, makes this application performable, and an output result may be expressed by Equation 29 below.

[00038] $c \equiv \tilde{R}$ .Math. $(2$ .Math. $a) \equiv R$ .Math. $a \bmod(Q_R)$ [Equation 29]

[0240] In this way, an output having a desired modulus Q.sub.R may be finally acquired using Inv-RS.

[0241] In Algorithm **6** in FIG. **17**, an intermediate polynomial b needs to satisfy Equation 30 below.

[00039] $.Math. b - \tilde{R}^{-1}$ .Math. a .Math. $_\infty \le \frac{k}{2}$ [Equation 30]

[0242] Here, K indicates the number of factored factors of ☐custom-character. In addition, finally, a 2.sup.β modulus operation may be replaced by a γ bit shift.

[0243] A final output c may be computed as shown in Equation 31 below.

[00040] $c = \frac{b - [b]_2}{2}$ [Equation 31]

[0244] Therefore, Equation 32 below may be satisfied.

[00041] $.Math. c - 2^-$ .Math. b .Math. $_\infty \le \frac{1}{2}$ [Equation 32]

[0245] This equation may lead to a relationship as shown in Equation 33 below.

[00042] $.Math. c - \tilde{R}^{-1}$ .Math. a .Math. $_\infty \le \frac{1}{2} + 2^-$ .Math. $\frac{k}{2}$ [Equation 33]

[0246] A desired result may be acquired by satisfying a condition as shown in Equation 33.

[0247] As a result, the present disclosure may perform the key switching in Algorithm **2** and the homomorphic multiplication in Algorithm **1** by using the universal sprout.

[0248] The key switching procedure may be slightly modified for the power-of-two factors. The key switching in Algorithm **2** may omit the Inv-RS operation. That is, for a power-of-two modulus part, the key switching may be performed on 2.sup.ord.sup.2.sup.(Q) by using modulus reduction. This configuration may indicate that the key switching is performed on the modulus 2.sup.ord.sup.2.sup.(Q) instead of 2.sup.ord.sup.2.sup. (Q.sup.max.sup.) without additional Inv-RS.

[0249] The following description discusses hardware-level arithmetic strategies and mainly focuses on a standard 64-bit processor. That is, the following description analyzes a method for efficiently implement the above-described methods. For simplicity, the description limits the discussion to a case of a single-threaded CPU.

[0250] In case of using a universal sprout appropriate for the standard 64-bit processor, a basic approach may require three 64-bit moduli to process polynomial multiplication using the NTT. To confirm this configuration, it may be assumed that r is a product of all moduli included in the sprout, and r<2.sup.64. To process the polynomial multiplication on the modulus r, a larger modulus p that includes the modulus r may then be utilized as follows. If the NTT is performed on the modulus p by setting modulus p>Nr.sup.2 the polynomial multiplication may be emulated on modulus r. That is, the NTT operation on p may emulate the polynomial multiplication performed on r, and thus there is no need to perform the modular reduction on p. In this case, r is close to 2.sup.64, and p thus generally needs to be larger than 2.sup.128. Therefore, three 64-bit NTT primes may be required.

[0251] However, the present disclosure proposes a method of using two NTT moduli instead of three NTT moduli in a 64-bit processor environment. To make this possible, the NTT that utilizes composite numbers (Composite Number NTT) may be used instead of primes. The composite number-based NTT is a method of configuring the NTT moduli by using the composite numbers instead of the primes, which may perform the same operation while using fewer NTT modules than the existing method.

[0252] (Example 5) In this Example, r.sub.top=2.sup.15.Math.r.sub.1.Math.r.sub.2, as shown in Example 3, where r.sub.1 may be a 16-bit NTT prime and r.sub.2 may be the 30-bit NTT prime. Its core idea is that modulus r.sub.1r.sub.2≈2.sup.46 may be handled using composite number-based NTT (Composite NTT). Both r.sub.1 and r.sub.2 are the NTT primes, and the 2Nth primitive roots of unity for r.sub.1r.sub.2 may thus be easily found. Therefore, 2.sup.15 modulus operation may be emulated by setting one NTT modulus as r.sub.1r.sub.2, setting the other NTT modulus as g>N.Math.2.sup.36, and performing g modulus operation. This method allows the operation to be performed using only two NTT moduli instead of the existing three NTT moduli.

[0253] This strategy may also be compatible with the key switching based on Ring Learning With Errors (RLWE). If the sprout modulus includes only r.sub.1 or r.sub.2 at any moment, the sprout modulus may be readily included (embedded) as r.sub.1r.sub.2 moduli without additional conversion. This process may be defined while maintaining compatibility with the NTT. That is, an embedding operation Embed r.sub.1 . . .

r.sub.1r.sub.2 may be defined by Equation 34 below.

[00043] $\text{Embed}_{r_1 - r_1 r_2} \circ \text{NTT}_{r_1} = \text{NTT}_{r_1 r_2} \circ \text{Embed}_{r_1 - r_1 r_2}$ [Equation34]

[0254] In the above equation, the embedding operation needs to be established in R.sub.r1.

[0255] For example, a simple method for defining embedding is as follows. The embedding may be performed by adding zero to the r.sub.2 modulus, or by applying Chinese Remainder Theorem (CRT) later. In this way, in the r.sub.1 modulus, the left and right terms correspond to NTTr.sub.1, and in the r.sub.2 modulus, the left and right terms may always be equal to zero.

[0256] If the sprout modulus is sufficiently small, a single word-sized NTT may be used to perform the NTT operation. In particular, the following two types of sprouts suggested in Example 3 may support the single-word operation. For the sprout without r.sub.2 (a first type), g may be set sufficiently large, and the 2.sup.15 modulus operation may thus be performed without overflow in the modulus g. Therefore, a single-word composite-number-based NTT may be performed using g.Math.r. For a sprout that includes sufficiently small power-of-two factors (a second type), for example, if the sprout is 2.sup.t.Math.r.sub.1.Math.r.sub.2 and t≤15, a 2.sup.t modulus may be included in g′>2.sup.2t.Math.N. Therefore, the entire sprout may be included in g′.Math.r.sub.1−r.sub.2. If g′.Math.r.sub.1.Math.r.sub.2.sup.26, the single-word composite-number-based NTT may be performed.

[0257] The present disclosure may use another word size than a standard 64-bit word size. However, a basic strategy may be almost the same as a standard 64-bit approach. That is, as many NTT moduli as possible may be packed into the composite number-based NTTs (Composite NTTs) and the remaining moduli may be embedded into larger moduli.

[0258] (Example 6) For a hardware architecture where the word size is 2.sup.ω bits (where ω≥7, i.e., 128 bits or more), the strategy used in a 64-bit case may be easily generalized. That is, the top sprout r.sub.top may be selected as follows.

[00044]$r_{top} = 2^{15} .\text{Math. } r_1 .\text{Math. } r_2 .\text{Math. } r_{\log_2(\ )} - 4$

[0259] Here, for 1≤i≤log.sub.2(ω)−4, r.sub.i indicates a 2.sup.i+3-bit sized prime.

[0260] This sprout may perform the operation by using two word sizes, as in the standard 64-bit case. That is, the power-of-two factor (power-of-two part) may be embedded into a larger prime, and the remaining factors may be processed using the composite-based NTT (Composite NTT).

[0261] (Example 7) A strategy for a 32-bit word size is as follows. First, the top sprout r.sub.top may be selected as follows.

[00045]$r_{top} - 2^{15} .\text{Math. } r$

[0262] Here, r=2.sup.16+1 indicates a 16-bit NTT prime.

[0263] 2.sup.15 may be embedded into two large word-sized NTT primes g.sub.1 and g.sub.2. That is, g.sub.1 and g.sub.2 are sufficiently large word-sized NTT primes, and need to satisfy the following condition.

[00046]$2^{30} .\text{Math. } N < g_1 g_2$

[0264] In this case, the sprout operation may be processed using three words g.sub.1, g.sub.2, and r.

[0265] If the power-of-two part 2.sup.t is sufficiently small to thus hold 2.sup.2t.Math.N<g.sub.1, and 2.sup.t may then be included in one modulo g.sub.1. That is, the operation may be performed using only two words, g.sub.1 and r.

[0266] Grafting may address specific constraints of the existing implementations by decoupling the scaling factor from the RNS modulus. Grafting may be effective in both low-precision and high-precision scenarios. The reason is that the size of the scaling factor is often not a multiple of the machine word size. In addition, the existing method requires finding a specific small NTT prime, while Grafting does not. Grafting may solve the problem in some cases where no NTT prime of the corresponding size exists. In addition, Grafting may also assist in accelerating the composite rescaling (Composite Rescaling) or tuple-CKKS multiplications. Grafting does not require pre-reserving any fixed RNS moduli to maintain a specific precision. Due to these characteristics, Grafting is hardware- and compiler-friendly, and may be used flexibly in various circuits because its computation is not fixed depending on the selection of the homomorphic operation.

[0267] The following description describes various application cases that utilize Grafting. Grafting freely sets the ciphertext modulus, thus enabling any circuit to be computed using arbitrary precision. This computation may be performed independently of the selection of the ciphertext modulus. In particular, the present disclosure describes a method for optimizing a CKKS bootstrapping technique. The first method is a technique for shortening execution time (run-time) by freely changing the output modulus, while the second method is a technique for reducing modulus consumption. In addition, the present disclosure also describes acceleration of the tuple-CKKS multiplication. In particular, the present disclosure may present a method for reanalyzing the tuple-CKKS multiplication and utilizing Grafting to thus improve operation performance.

[0268] In general, CKKS bootstrapping may be used to maximize a multiplicative budget to enable additional operations to be performed. A bootstrapping process may include homomorphic decoding of the ciphertext, modulus raising, re-encoding of the ciphertext, and modular reduction to a base modulus. In this process, the respective operations may be referred to as an operation to convert the ciphertext to slot representation (SlotToCoeff), an operation to increase a modulus size (ModRaise), an operation to convert the slot representation back to coefficient representation (CoeffToSlot), and an operation to evaluate the modulus operation by using polynomial approximation (EvalMod). A given input ciphertext ct∈R.sup.2.sub.m6, may basically have the message m in its encrypted state, a new ciphertext ct′∈R.sup.2.sub.Q.sub.max using the modulus Q.sub.max has a message m+q.sub.01, in its encrypted state, and an operation using a modulus Mod q.sub.0 that is performed based on q.sub.0 may then be evaluated by using the polynomial approximation.

EvalMod with Adaptive Precision

[0269] In an EvalMod process, reduction of the modulus Mod q.sub.0 may be approximated by a sine function near zero, which may be decomposed into Cosine Approximation, Double Angle Formula, and Arcsine Approximation.

[0270] The present disclosure may utilize Chebyshev approximation of a cosine function. For example, the present disclosure may use Chebyshev coefficients (Chebyshev Basis) expressed as a 63-degree polynomial. As a polynomial degree is increased, the coefficients tend to get smaller. Therefore, the modulus consumption may be reduced by scaling the coefficients less precisely while maintaining final precision. For example, a circuit for evaluating the polynomial may be segmented into two subordinate circuits using different modulus chains. [0271] 1) Compute the first half of the polynomial (degrees zero to 31).fwdarw.Consume five multiplication depths while maintaining desired precision. [0272] 2) Compute the second half of the polynomial (degrees 32 to 63).fwdarw.Consume six multiplication depths while using smaller coefficients, thus performing the computation with lower precision.

[0273] The second half requires more multiplication depths, thereby reducing the modulus consumption in an entire polynomial approximation process. For example, if five depths (Δ=42 bits)+six depths (Δ=37 bits) are performed in parallel, a total modulus consumption is max(5×42, 6×37)=222 bits, thereby saving a 30-bit modulus compared to the existing method (6×42=252 bits). However, two subcircuits need to compute Chebyshev Bases, respectively, thereby slightly increasing a cosine function evaluation cost. This method may also be extended to a general Chebyshev Approximation evaluation method, thereby further reducing the modulus consumption.

Bootstrapping with Flexible Output Modulus.

[0274] In general, the bootstrapping aims to maximize the multiplicative budget for future computations. However, the maximum budget may not be necessary depending on the circuit being evaluated. For example, high modulus computation may be more expensive for a circuit that includes shallow multiplication depth and wide parallel computation. For example, if a neural network layer is computed, this layer may involve large matrix multiplication followed by polynomial evaluation (activation function approximation). Depending on a matrix size, it may be more efficient to evaluate matrix multiplication by using the smallest modulus and then refresh the modulus to evaluate the rest of the circuit, such as by using the activation function. That is, it may be more efficient to consider only a few multiplication steps and set a smaller modulus instead of maintaining the

maximum modulus in case of evaluating the matrix multiplication. However, this method may not be easily implemented using original bootstrappable CKKS parameters. The precision in a bootstrapping sub-procedure needs to be larger than that of other multiplications for the same output precision. Therefore, the scaling factor may be selected differently to optimize the implementation. This selection restricts the speedup of the bootstrapping procedure by performing the modulus reduction to any desired size. However, these constraints no longer apply to the Grafting scenario, and the output modulus may be flexibly selected, which may lead to the speedups proportional to the modulus size.

[0275] Prior Art **1**: Cheon, J. H., Cho, W., Kim, J., & Stehle, D. Homomorphic multiple precision multiplication for CKKS and reduced modulus consumption, in W. Meng, C. D. Jensen, C. Cremers, & E. Kirda (Eds.), Proceedings of the ACM Conference on Computer and Communications Security (CCS) (pp. 696-710), ACM Press (November 2023). Prior Art **1** introduces a new multiplication algorithm for CKKS and reduces the modulus consumption in each homomorphic multiplication. Gradually, their algorithms may achieve similar throughput or even better latency in case of switching to smaller rings. However, in many cases, the reduced modulus consumption does not directly lead to improved efficiency. The reason is that the performance of computing q is roughly the same as long as q fits the machine word size. Grafting may bridge a gap between expectations and reality. That is, tuple multiplication does not significantly reduce the throughput compared to an original (single) multiplication.

[0276] Hereinafter, the compatibility of tuple multiplication in Grafting and its efficiency may be verified. The present disclosure follows the notation used in Prior Art **1**. CT indicates a ciphertext tuple, ![custom-character]custom-character indicates the ciphertext modulus a level ![custom-character]custom-character, .Math. indicates CKKS tensor operation, Relin indicates CKKS relinearization, RSq indicates the rescaling by q, and DCP and RCB indicate the decomposition of the CKKS ciphertext into quotients and remainders and their reassembly. For simplicity, the present disclosure focuses on pair multiplication, and general tuple multiplication may be verified in almost the same way.

[0277] The sole difficulty in applying the concept of compatible Grafting is defining pair rescaling if a rescaling factor ![custom-character]custom-character/![custom-character]custom-character does not belong to Z. This difficulty may occur in a Graphing framework of the present disclosure. The present disclosure defines a generalized version of the pair rescaling as follows.

[0278] Definition 2 (Pair Rescaling, Rational Numbers) CT=(ĉt, čt)∈R.sub.Q(l).sup.2×R.sub.Q(l).sup.2 may be a ciphertext pair. In ![custom-character]custom-character/![custom-character]custom-character=![custom-character]custom-character/![custom-character]custom-character, ![custom-character]custom-character, ![custom-character]custom-character∈![custom-character]custom-character>0 are coprime natural numbers. The rescaling of CT is defined as ![custom-character]custom-character×![custom-character]custom-character, and follows Equation 35 below.

[00047] $RS^2_{\ell/\ell} := (RS_\ell(_\ell .Math. \hat{ct}), RS_\ell(q_{div}\ _\ell .Math. \hat{ct} + _\ell .Math. \hat{ct}) - q_{div} .Math. RS_\ell(_\ell .Math. \hat{ct}))$ [Equation35]

[0279] In addition, the present disclosure provides a generalized version of Lemma **4.6** of Prior Art **1** in Lemma **1** below, which may show the accuracy of the pair multiplication after changing a definition of the pair rescaling.

[0280] Lemma **1**. CT=(ĉt, čt)∈![custom-character]custom-character×![custom-character]custom-character may be the ciphertext pair. In ![custom-character]custom-character/![custom-character]custom-character= ![custom-character]custom-character/![custom-character]custom-character, ![custom-character]custom-character, ![custom-character]custom-character∈![custom-character]custom-character>0 are the coprime natural numbers. sk=(1, s)∈R.sup.2 may be the secret key having Hamming weight h. In addition, a quantity according to Formula 36 below has an infinity norm≤(h+1)/2.

[00048] $[(RCB_{q_{div}}(RS^2_{\ell/\ell}(CT))) .Math. sk]_{Q^{(\ell-1)}} - \frac{\ell}{\ell}[(RCB_{q_{div}}(CT)) .Math. sk]_{Q^{(\ell)}}$ [Formula36]

[0281] Theorem **3** below is a theorem similar to Theorem **4.8** of Prior Art **1**, which ensures the accuracy of the pair multiplication. The accuracy of the pair multiplication is proven in Lemma **1**. Mult.sup.2 may be defined as a composition of tensor, relinearization, and rescaling.

[0282] [Theorem **3**] Let CT=(ĉt.sub.1, čt.sub.1), CT.sub.2=(ĉt.sub.2, čt.sub.2)∈![custom-character]custom-character×![custom-character]custom-character may be the ciphertext pair. In ![custom-character]custom-character/![custom-character]custom-character=![custom-character]custom-character/![custom-character]custom-character, (![custom-character]custom-character, ![custom-character]custom-character)=1, and sk=(1, s)∈R.sup.2 may be the secret key having the Hamming weight h. It may be assumed that ‖Dec(ĉt.sub.i)‖∞≤{circumflex over (M)} and ‖1Dec(čt.sub.i)‖∞Ṁ hold for all i∈{1, 2} and {circumflex over (M)}, Ṁ satisfy Formula 37 below.

[00049] $N(\hat{M}_{q_{div}} + \hat{M})^2 + E_{Relin} + h < Q^{(\ell)} / 2.$ [Formula37]

[0283] In addition, a quantity according to Formula 38 below has a value having an infinity norm≤(NM.sup.2/q.sub.div+E.sub.Relin, +h)![custom-character]custom-character/![custom-character]custom-character+(h+1)/2.

[00050][Formula38][(RCB_{q_{div}} (Mult^2 (CT_1, CT_2))) .Math. sk]_{Q^{(\ell-1)}} - \frac{\ell}{\ell} [(RCB_{q_{div}} (CT_1) .Math. sk) .Math. (RCB_{q_{div}} (CT_2) .Math. sk)]_{Q^{(\ell)}}

[0284] Efficiency. Next, the description describes efficiency improvement if Grafting is applied to double-CKKS. α=(log.sub.2(q.sub.div), b=⌞log.sub.2(![custom-character]custom-character/![custom-character]custom-character)⌝ may be a size of a division prime and a size of a rescaling factor, respectively. Here, b may be selected slightly larger than a. In case of comparing Mult with Mult.sup.2, the a+b bit RLWE multiplication in Mult may be compared to two b bit multiplications in Mult.sup.2. Assuming that a+b bit computation is (a+b)/b times more expensive than b bit computation, Mult.sup.2 needs to have asymptotically the same throughput as Mult. However, the actual implementation is affected by the machine word size, and in the worst case, Prior Art **1** estimates that Mult.sup.2 may be twice as slow as Mult in terms of the throughput. Table 2 of Prior Art **1** provides parameters for increasing an isomorphic capacity compared to an existing CKKS parameter using 57-bit primes. A modulus used in Prior Art **1** may be composed of two 61-bit primes, eighteen 38-bit primes, and three 23-bit primes. Grafting allows the use of 15 unit moduli, each having approximately 61 bits. This configuration allows the use of 14 dnums instead of 22 dnums, and 15 RNS moduli instead of 23 RNS moduli. As a result, double-CKKS using Grafting may be expected to show a performance improvement of approximately 22/14×23/15≈2.41 times. The present disclosure notes that the efficiency improvement may be even more significant for lower precisions or fewer slots. In this case, a smaller division prime and a smaller rescaling factor may be needed. For example, log.sub.2(q.sub.div)≤10 and log.sub.2(![custom-character]custom-character/![custom-character]custom-character)≤20 may be used for precision ≤15-bit, which may lead to ≥3× performance improvement compared to pure double-CKKS implementation.

[0285] Meanwhile, the RNS in the homomorphic encryption is a method proposed to increase efficiency of the homomorphic operation. Grafting may solve inefficiency, which is caused by applying the RNS to CKKS, i.e., inefficiency caused by using more NTT processors than the theoretically optimal number, by presenting a modulus composition referred to as the sprout as a part of the entire RNS modulus. However, there may be inefficiencies in configuring the sprout itself although this method greatly reduces the total number of the NTT processors. For example, the sprout is not the NTT prime. Accordingly, the sprout is unable to use the NTT processor that exactly fits its size, and needs to use a larger one.

[0286] Therefore, in performing various homomorphic operations, it is necessary to reduce the number of NTT processors required for the sprout to thus achieve efficiency gains. Here, the NTT processor may efficiently perform polynomial multiplication over a ring R_q=Z_q[x]/(x∧N+1) for the NTT prime q (or pseudo NTT prime, which is a prime that is not fully split and therefore does not have the 2Nth roots of unity, and has Nth roots of unity) by using the NTT (or pseudo NTT). When split into multiple qs in this way, assume that one NTT processor is required for each q. A possible size of q needs to be less than 2.sup.(w−2). Here, w indicates the machine word size. For example, the 64-bit processor requires q to be less than 2.sup.62.

[0287] The NTT processor based on the sprout in the 64-bit processor may have the following two basic configuration operations. The first operation is Composite NTT, where for two NTT primes q.sub.1 and q.sub.2 with q.sub.1*q.sub.2<2.sup.62, 2Nth roots of unity mod q.sub.1 and mod q.sub.2 are processed as values in mod q.sub.1*q.sub.2, and the computation is performed as the NTT for q.sub.1*q.sub.2. The second operation is embedded NTT, where for a non-NTT prime q with q.sub.2*N/2*dnum<2.sup.64, an NTT prime Q is selected with q.sup.2*N/2*dnum≤Q<2.sup.64, polynomial multiplication over R_Q is performed using NTT mod Q, and a result of the polynomial multiplication in R_Q is then recovered using a mod q operation.

[0288] According to an embodiment of the present disclosure, if sprout r=g.sub.1*g.sub.2*g.sub.3, g.sub.1=31-bit NTT prime, g.sub.2=16-bit NTT

prime (or 2.sup.16+1), and g.sub.3=2.sup.15. In this case, the sprout is not the NTT prime, and thus requires three word-sized NTT processors. Alternatively, if some sprouts remain, the electronic apparatus **100** may use two NTT processors if (128-logN)/2=56 bits or less. If some sprouts remain, the electronic apparatus **100** may use only one NTT processor if (64-logN)/2=24 bits or less. For reference, the modulus change may be as follows. The number of bits may be in parentheses.

[0289] Sprout r=g.sub.1*g.sub.2*g.sub.3(62)).fwdarw.g.sub.1*g.sub.2*2.sup.14(61).fwdarw. . . . .fwdarw.g.sub.1*g.sub.2(47).fwdarw.g.sub.1*g.sub.3(46).fwdarw.g.sub.1*2.sup.14(45).fwdarw. . . . .fwdarw.g.sub.1(31).fwdarw.g.sub.2*2.sup.14(30).fwdarw. . . . .fwdarw.g.sub.2(16).fwdarw.g.sub.3(15).fwdarw. . . . .fwdarw.2.sup.1(1).

[0290] According to an embodiment of the present disclosure, the number of the word-sized NTT processors may be reduced to two. In detail, in case that the sprout modulus is expressed as a product of g.sub.1, which is the 31-bit Number Theoretic Transform (NTT) prime, g.sub.2, which is a 16-bit NTT prime or in the form of a 2.sup.16+1, and G.sub.3, which is a value in the form of 2, the processor **450** may process a product of g.sub.1 and g.sub.2 by using Composite NTT and process g.sub.3 by using the embedded NTT to thus integrate their results into G.sub.3 if the machine word size is greater than 38 bits and less than or equal to 64 bits. That is, the electronic apparatus **100** may use two NTT processors if the machine word size is greater than 38 bits and less than or equal to 64 bits. The processor **450** may process g.sub.3 by using the embedded NTT to thus acquire G.sub.3 if the machine word size is less than or equal to 38 bits. Here, the electronic apparatus **400** may use one NTT processor if the machine word size is less than or equal to 38 bits. However, the configuration method and the required number of NTTs may vary depending on the remaining sprouts (the number of bits).

[0291] As an example, if all the sprouts remain (if the number of bits is 62 bits), g.sub.1*g.sub.2 may be processed using comNTT, and g.sub.3 may be processed using the embedded NTT to thus acquire G.sub.3. In this case, the electronic apparatus **400** may use two NTT processors.

[0292] As an example, if no g.sub.1 remains (i.e., the number of bits is 31 bits), g.sub.2*G.sub.3 may be processed using comNTT. In this case, the electronic apparatus **400** may use one NTT processor.

[0293] As an example, if no g.sub.2 remains (if the number of bits is 46 bits), g.sub.1 may be processed using the NTT, and g.sub.3 may be processed using the embedded NTT to thus acquire G.sub.3. In this case, the electronic apparatus **400** may use two NTT processors.

[0294] As an example, if no g.sub.3 remains (i.e., the number of bits is 47 bits), g.sub.1*g.sub.2 may be processed using comNTT. In this case, the electronic apparatus **400** may use one NTT processor.

[0295] As an example, if g.sub.3 is partially missing (i.e., the number of bits is 38 bits), g.sub.1*GG.sub.3 (the remaining g.sub.3~2.sup.7.fwdarw.GG.sub.3~2.sup.31) may be processed using comNTT. In this case, the electronic apparatus **400** may use one NTT processor.

[0296] The following description describes the optimization in the case of the key switching. The key switching is a process of switching the encrypted data to a new key, and includes several optimization methods to increase the efficiency. The present disclosure describes these optimization methods as including the following three methods (Method **1**, Method **2**, and Method **3**). [0297] 1) Method **1**: If ctxt and key are always in the same format (format a)

[0298] This method may be used as the optimization method if both a ciphertext ctxt and the key are always in the same format (format a). Here, the optimization may be performed using modulus multiplier increase (ModRaise) and modulus reduction (ModRed) or the inverse rescaling (Inv-Rescale). [0299] 2) Method **2**: If various optimizations are applied to ctxt, and the key is always in the format a.

[0300] This method may be used if various optimizations are applied to the ciphertext ctxt and the key is always in the format a. In this case, ctxt needs to be converted to the format a, and then be proceeded as in Method **1**. This conversion may require at most one additional NTT computation. However, the remaining process may be integrated into modulus upgrade (ModUp) to thus improve the efficiency. [0301] 3) Method **3**: If various optimizations are applied to ctxt, and the key is received in the format a and converted into various formats.

[0302] This method may be used if various optimizations are applied to the ciphertext, and the key is received in the format a and then converted into various formats. This method proceeds like Method **1**, and unable to be applied to a format c′.

[0303] The description describes an operation of the NTT processor based on the sprout in a 32-bit processor. The goal is to use the NTT as much as possible if the NTT is faster than Toom-Cook. However, the NTT or Discrete Fourier Transform (DFT) is unable to be used for the part g.sub.3. Here, Toom-Cook or text-book multiplication method may be used for g.sub.3. The sprout strategy presented in Grafting is a method for utilizing the NTT more efficiently by using 16 bits+2.sup.15 (g.sub.2 & g.sub.3) configuration. However, the NTT or the DFT is unable to be used for g.sub.3, thus requiring an alternative method for this part.

[0304] According to an embodiment of the present disclosure, a method using double sprouts may be provided. Here, a sprout **1** sp**1** may be 16 bits+2.sup.15 (g.sub.2 & g.sub.3). A sprout **2** sp**2** may be 24 bits+2.sup.7. The modulus chain may be (24+7).fwdarw. . . . .fwdarw.24.fwdarw.16+7.fwdarw. . . . .fwdarw.16.fwdarw.15.fwdarw. . . . .fwdarw.8.fwdarw.7.fwdarw. . . . .fwdarw.1. The modulus chain may be a method in which the sprout passes through various sizes, such as 24 bits, 16 bits and the like, and gradually decreasing to a smaller size at each step. The key switching may be applied as follows: sp**2**.fwdarw.sp**1**, Toom-Cook.fwdarw.sp**2** (based on M**1**). That is, the key switching process may be a method of switching the key from sp**2** to sp**1** and then applying the Toom-Cook method to switch the key back to sp**2**.

[0305] The present disclosure describes the key switching as including the following four methods (Method **1**, Method **2**, Method **3**, and Method **4**).

[0306] 1) Method **1**: A method of using two keys based on the sprout, where the Toom-Cook method may be applied if ModUp is to be performed on sp**1**. [0307] 2) Method **2**: A method of receiving a key for sp**1** and generating a key for sp**2** using ModSwitch, where the key for sp**1** is received and the key for sp**2** is generated in advance using ModSwitch, and the process may then proceed in the same way as Method **1**. [0308] 3) Method **3**: A method of using a single key (for sp**2**) and switching ctxt from sp**1** to sp**2** before ModUp, where the ciphertext is converted from format sp**1** to format sp**2** by using the single key for sp**2** before ModUp. [0309] 4) Method **4**: A method of using a single key (24 bits+16 bits+15 bits) and requiring a large P, where the processing is carried out using this single key (24 bits+16 bits+15 bits).

[0310] Meanwhile, the above-described embodiment describes the 64-bit or 32-bit processor, which is only an example, and may also be applied to processors of other bit sizes.

[0311] Meanwhile, according to an embodiment of the present disclosure, the scaling factor Δ may be dynamically changed if the operation on the homomorphic ciphertext is performed using Grafting. That is, the scaling factor Δ may be adjusted even while the operation is in progress. By utilizing this configuration, the modulus consumption may be reduced. That is, if the same scaling factor is used for all sub-circuits as in the prior art, the scaling factor Δ may remain the same, even though the operation depth and precision of each sub-circuit may differ. In this case, the modulus consumption may be large. However, as in the present disclosure, the scaling factor Δ may be adjusted for each sub-circuit to optimize the modulus consumption. That is, in the sub-procedure that does not require the high precision during the operation, the modulus consumption may be reduced by using the scaling factor Δ smaller than usual.

[0312] In particular, if the homomorphic operation is performed on the polynomial having a high degree such as the 31st, 63rd, or 127th degree polynomial, the coefficients of the high-degree terms are typically much smaller than those of the low-degree terms, thereby reducing the modulus consumption by using a smaller scaling factor Δ. For example, the 0th to 31st terms of the 63rd-degree polynomial may require four multiplicative depths, while the 32nd to 63rd terms may require five multiplicative depths. Here, the 0th to 31st terms may use a large scaling factor Δ.sub.large for the four depths, while the 32nd to 63rd terms may use a smaller scaling factor Δ.sub.small for the five depths, thereby achieving the modulus consumption corresponding a depth of max(Δ.sub.large×4, Δ.sub.small×5).

[0313] This configuration may reduce an additional operation cost.

[0314] As one or more embodiments, EvalMod is one of operations used in the homomorphic encryption (FHE), and generally indicates a process of evaluating the mod operation. EvalMod has multi-level multiplication depths, and it is thus important to improve the operation efficiency. In

particular, the precision related to Sine/Cosine Approximation may be adjusted using the EvalMod operation. Therefore, according to an embodiment of the present disclosure, Grafting may be utilized to adjust the operation precision, and the high-order coefficients may be optimized, thereby optimizing the modulus consumption. For example, if EvalMod is performed on the ciphertext having the maximum degree of 63, the EvalMod (FVa) method according to the present disclosure may save approximately 50 bits of modulus consumption compared to the existing method. As a result, the operation cost may be reduced by up to 17%.

[0315] That is, according to an embodiment of the present disclosure, the modulus consumption may be reduced.

[0316] Meanwhile, the method according to the various embodiments of the present disclosure may be provided as included in a computer program product. The computer program product may be traded as a commodity between a seller and a purchaser. The computer program product may be distributed in the form of a storage medium (for example, a compact disc read only memory (CD-ROM)) that may be read by a machine, or may be distributed online (for example, downloaded or uploaded) through an application store (for example, PlayStore™) or directly between two user devices (for example, smartphones). In case of the online distribution, at least a portion of the computer program product (e.g., downloadable application) may be at least temporarily stored on a machine-readable storage medium such as the memory of a server of a manufacturer, a server of an application store or a relay server, or be temporarily generated.

[0317] The various embodiments of the present disclosure may be implemented in software including an instruction stored on the machine-readable storage medium (e.g., computer-readable storage medium). A machine may be a device that invokes the stored instruction from a storage medium, may be operated based on the invoked instruction, and may include the electronic apparatus according to the disclosed embodiments.

[0318] Meanwhile, the machine-readable storage medium may be provided in the form of a non-transitory storage medium. Here, the "non-transitory storage medium" is a tangible device and may only indicate that this storage medium does not include a signal (e.g., electromagnetic wave), and this term does not distinguish a case where data is stored semi-permanently on the storage medium and a case where data is temporarily stored on the storage medium from each other. For example, the "non-transitory storage medium" may include a buffer in which data is temporarily stored.

[0319] If the instruction is executed by the processor, the processor may directly perform a function corresponding to the instruction or other components may perform the function corresponding to the instruction under the control of the processor. The instruction may include a code generated or executed by a compiler or an interpreter.

[0320] Although the embodiments are shown and described in the present disclosure as above, the present disclosure is not limited to the above-described specific embodiments, and may be variously modified by those skilled in the art to which the present disclosure pertains without departing from the gist of the present disclosure as claimed in the accompanying claims. These modifications should also be understood to fall within the scope and spirit of the present disclosure.

Claims

1. An electronic apparatus comprising: a memory for storing at least one instruction; and a processor, wherein the processor is configured to acquire a homomorphic ciphertext by using a Residue Number System (RNS) modulus including a plurality of moduli each having a size corresponding to a machine word size, and perform an operation on the homomorphic ciphertext by using rational rescaling.

2. The apparatus as claimed in claim 1, wherein the plurality of moduli include sprout moduli composed of a product of primes each having a size less than or equal to the machine word size.

3. The apparatus as claimed in claim 2, wherein the processor is configured to generate an intermediate modulus by resurrecting the sprout modulus, and acquire an output modulus by rescaling the intermediate modulus, the intermediate modulus being a multiple of the output modulus.

4. The apparatus as claimed in claim 2, wherein the processor is configured to perform a key switching operation on the homomorphic ciphertext by using the sprout modulus.

5. The apparatus as claimed in claim 2, wherein in case that the sprout modulus is expressed as a product of g_1, which is a 31-bit Number Theoretic Transform (NTT) prime, g_2, which is a 16-bit NTT prime or in the form of $2^{16}+1$, and g_3, which is a value in the form of 2, the processor is configured to process a product of g_1 and g_2 by using composite number-based NTT (Composite NTT) and process g_3 by using embedded NTT to thus integrate their results into G_3 if the machine word size is greater than 38 bits and less than or equal to 64 bits, and process g_3 by using the embedded NTT to thus acquire G_3 if the machine word size is less than or equal to 38 bits.

6. The apparatus as claimed in claim 5, wherein two NTT processors are used if the machine word size is greater than 38 bits and less than or equal to 64 bits, and one NTT processor is used if the machine word size is less than or equal to 38 bits.

7. The apparatus as claimed in claim 1, wherein the processor is configured to generate an intermediate modulus by upscaling the RNS modulus, perform a key switching operation on the intermediate modulus, and perform rescaling on the intermediate modulus on which the key switching operation is performed.

8. The apparatus as claimed in claim 1, wherein the processor is configured to adjust the plurality of homomorphic ciphertexts having different moduli and scaling factors by using the rational rescaling.

9. A control method of an electronic apparatus, the method comprising: acquiring a homomorphic ciphertext by using a Residue Number System (RNS) modulus including a plurality of moduli each having a size corresponding to a machine word size included in the electronic apparatus; and performing an operation on the homomorphic ciphertext by using rational rescaling.

10. The method as claimed in claim 9, wherein the plurality of moduli include sprout moduli composed of a product of primes each having a size less than or equal to the machine word size.

11. The method as claimed in claim 10, wherein the performing includes generating an intermediate modulus by resurrecting the sprout modulus, and acquiring an output modulus by rescaling the intermediate modulus, the intermediate modulus being a multiple of the output modulus.

12. The method as claimed in claim 10, wherein in the performing, a key switching operation is performed on the homomorphic ciphertext by using the sprout modulus.

13. The method as claimed in claim 10, wherein in case that the sprout modulus is expressed as a product of g_1, which is a 31-bit Number Theoretic Transform (NTT) prime, g_2, which is a 16-bit NTT prime or in the form of $2^{16}+1$, and g_3, which is a value in the form of 2, in the performing, a product of g_1 and g_2 is processed by using composite number-based NTT (Composite NTT) and g_3 is processed by using embedded NTT to thus integrate their results into G_3 if the machine word size is greater than 38 bits and less than or equal to 64 bits, and g_3 is processed by using the embedded NTT to thus acquire G_3 if the machine word size is less than or equal to 38 bits.

14. The method as claimed in claim 13, wherein two NTT processors are used by the electronic apparatus if the machine word size is greater than 38 bits and less than or equal to 64 bits, and one NTT processor is used by the electronic apparatus if the machine word size is less than or equal to 38 bits.

15. The method as claimed in claim 9, wherein the performing includes generating an intermediate modulus by upscaling the RNS modulus, performing a key switching operation on the intermediate modulus, and performing rescaling on the intermediate modulus on which the key switching operation is performed.