Office de la Propriété Intellectuelle du Canada
Un organisme d'Industrie Canada

Canadian Intellectual Property Office
An agency of Industry Canada

CA 2631925 C 2013/03/19

(11)(21) 2 631 925

(12) BREVET CANADIEN
CANADIAN PATENT

(13) C

(22) Date de dépôt/Filing Date: 2008/05/21

(41) Mise à la disp. pub./Open to Public Insp.: 2008/11/22

(45) Date de délivrance/Issue Date: 2013/03/19

(30) Priorité/Priority: 2007/05/22 (US11/751,875)

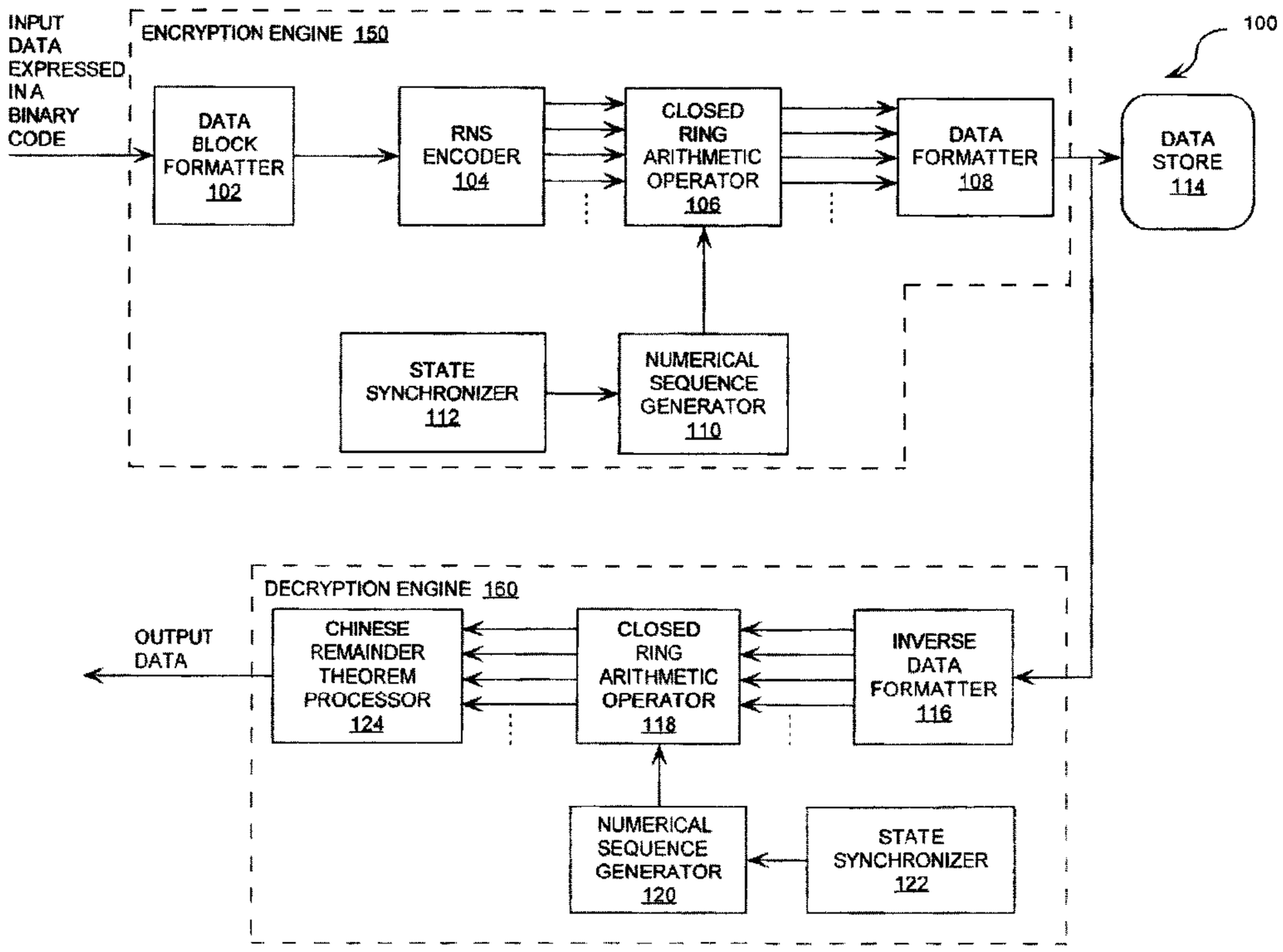
(51) Cl.Int./Int.Cl. *H04L 9/28* (2006.01),
H03M 13/00 (2006.01)

(72) Inventeurs/Inventors:
CHESTER, DAVID B., US;
MICHAELS, ALAN J., US

(73) Propriétaire/Owner:
HARRIS CORPORATION, US

(74) Agent: GOUDREAU GAGE DUBUC

(54) Titre : CHIFFREMENT PAR ERREURS INDUITES NON PONDEREES
(54) Title: ENCRYPTION VIA INDUCED UNWEIGHTED ERRORS



(57) Abrégé/Abstract:
A method for encrypting data is provided. The method includes formatting data represented in a weighted number system into data blocks. The method also includes converting the data blocks into a residue number system representation. The method further includes generating a first error generating sequence and inducing errors in the data blocks after converting the data blocks into a residue number system representation. It should be understood that the errors are induced in the data blocks by using the first error generating sequence. After inducing errors into the data blocks, the data of the data blocks is formatted into a form to be stored or transmitted. The method also includes generating a second error generating sequence synchronized with and identical to the first error generating sequence and correcting the errors in the data blocks using an operation which is an arithmetic inverse of a process used in inducing errors.

ABSTRACT

A method for encrypting data is provided. The method includes formatting data represented in a weighted number system into data blocks. The method also includes converting the data blocks into a residue number system representation. The method further includes generating a first error generating sequence and inducing errors in the data blocks after converting the data blocks into a residue number system representation. It should be understood that the errors are induced in the data blocks by using the first error generating sequence. After inducing errors into the data blocks, the data of the data blocks is formatted into a form to be stored or transmitted. The method also includes generating a second error generating sequence synchronized with and identical to the first error generating sequence and correcting the errors in the data blocks using an operation which is an arithmetic inverse of a process used in inducing errors.

ENCRYPTION VIA INDUCED UNWEIGHTED ERRORS

The invention concerns cryptography. More particularly, the invention concerns a cryptographic system that encodes data by combining an input signal in a residue number system representation with an error generating sequence.

There are many methods known in the art for encrypting data prior to the data's transmission over a communications network. One such method involves receiving an input signal at a cryptographic device. The input signal is expressed in a binary number system representation, i.e., characters of the English language are represented by a sequence of bits. Each bit of the sequence has a zero (0) or a one (1) value. The method also involves generating an encryption sequence. The encryption sequence can be a pseudo-random number sequence generated using a highly non-linear method. The method further involves inducing changes in the input signal using the encryption sequence. Such changes mask the input data and are introduced into the input signal by performing an arithmetic operation. These arithmetic operations are typically arithmetic operations in Galois fields. For example, a binary encryption sequence is added to or subtracted from the input signal bit by bit using Galois field GF[2] arithmetic. Alternatively, the binary encryption sequence is multiplied with the input signal using Galois field GF[2] arithmetic. In this regard, it should be appreciated that each of the above listed arithmetic operations results in modifying every bit of the input signal. In effect, the data contained in the input signal is encrypted.

As will be understood by a person skilled in the art, an advantage of such an encryption method is that it provides a high degree of security. Despite such an advantage, this method suffers from certain drawbacks. For example, algorithms typically employed for generating the encryption sequence are computationally intensive. As such, software programs implementing these algorithms have a slow data processing time.

Another such method involves receiving an input signal expressed in a binary number system representation at a cryptographic device. The method also

involves generating a cryptographic sequence. The method further involves arithmetically operating on an N bit block of the input signal using an N bit block of the encryption sequence in Galois field arithmetic. The encryption is induced in the input signal by performing an arithmetic operation, such as addition, subtraction, or multiplication. As a result of performing the arithmetic operation, the entire block is modified. For example, each character of the English language is represented by an eight (8) bit sequence. A block of eight (8) characters forms a sixty-four (64) bit block. Each bit of the sixty-four (64) bit block has a zero (0) or a one (1) value. As such, the sixty-four (64) bits of each input block sequence are modified by combining the input signal using Galois field arithmetic with a block of the encryption sequence. Consequently, the data contained in the input signal is encrypted.

As will be understood by a person skilled in the art, computational complexity can be reduced by reducing the complexity of the encryption sequence generation method. This can be done either by reducing the periodicity of the algorithm or by reducing the amount of non-linearity in the algorithm. As will be understood by a person skilled in the art, an advantage of this encryption method is that it is less computationally intensive than the previously described method. Despite such an advantage, this method suffers from certain drawbacks. For example, this method offers a lower degree of security.

In some encryption applications, a trade off between a degree of security and an encryption algorithm's computational intensity is not justified. Such applications include, but are not limited to, an encryption of medical records and an encryption of credit card records. As such, there is a need for an encryption method offering a moderate degree of security at an appreciably lower implementation complexity than that of a conventional high security encryption method.

A method is provided for encrypting data. The method is based on the difficulty in mapping error statistics between errors induced in residues and the corresponding errors resulting when residue number systems are mapped into a weighted number system. This difficult error mapping masks the characteristics of the method used to induce errors and thus encrypts the input data. The inverse

operation is performed to decrypt the encrypted information. The masking of the error inducing or encrypting sequence allows the use of a less complex and thus less computationally intensive sequence generation while yielding an acceptable level of security.

The method includes formatting data represented in a weighted number system into two or more data blocks of N bits in length. The method also includes converting the data in the data blocks into a residue number system representation. The method further includes generating a first error generating sequence, such as a simple linear sequence, a pseudo-random number sequence, a chaotic sequence, or a complex non-linear sequence. Thereafter, the method includes inducing errors in the data blocks by using the error generating sequence to arithmetically operate on one or more residues using Galois arithmetic. In this regard, it should be understood that the errors are induced in the data blocks after converting the data into the residue number system representation.

According to an aspect of the invention, the method includes introducing known errors in one or more residue values generated in the residue number system. The method also includes performing an arithmetic operation on the residue values based on the first error generating sequence. The method further includes transforming the data blocks to a weighted number system. In this regard, it should be understood that this transformation is performed after errors are induced in the data blocks.

According to another aspect of the invention, the method includes generating the first error generating sequence in a weighted number system or a residue number system. The method also includes inducing errors in only selected residues. It should be appreciated that the residues may be varied over time. The method further includes formatting the data into a weighted form to be stored or transmitted after inducing errors in the data blocks.

According to yet another aspect of the invention, the method includes receiving the data blocks after inducing errors in the data blocks. The method also includes generating a second error generating sequence synchronized with and

identical to the first error generating sequence. The method further includes correcting the errors in the data blocks using an operation which is an arithmetic inverse of a process used in inducing errors in the data blocks. The method includes converting the data blocks from the residue number system to the weighted number system after correcting the errors in the data blocks. The weighted number system is selected from a group consisting of a decimal, a binary, an octal and a hexadecimal number system.

Embodiments will be described with reference to the following drawing figures, in which like numerals represent like items throughout the figures, and in which:

FIG. 1 is a block diagram of a cryptographic system having an encryption engine and a decryption engine that is useful in understanding the invention.

FIG. 2 is a flow diagram of a method for encrypting and decrypting data that is useful for understanding the invention.

FIG. 3 is a graph showing an autocorrelation of an input sequence that is useful for understanding the invention.

FIG. 4 is a graph showing an autocorrelation of an encrypted sequence that is useful for understanding the invention.

Referring now to FIG. 1, there is provided a block diagram of a cryptographic system 100 that is useful in understanding the invention. The system 100 is comprised of an encryption engine 150 and a decryption engine 160. As shown in FIG. 1, the encryption engine 150 is comprised of a data block formatter 102, a residue number system (RNS) encoder 104, a closed-ring-arithmetic operator 106, and a data formatter 108. The encryption engine 150 is also comprised of a numerical sequence generator 110 and a state synchronizer 112.

The data block formatter 102 is comprised of hardware and software configured to receive an input signal including bits of information expressed in a weighted number system representation. The phrase "weighted number system" as used herein refers to a number system other than a residue number system. Such

number systems include, but are not limited to, a decimal number system, a binary number system, an octal number system, and a hexadecimal number system. The data block formatter 102 is also comprised of hardware and software configured to arrange the bits of information into a plurality of data blocks.

According to an aspect of the invention, each data block represents one or more characters of the English language. According to another aspect of the invention, the data blocks contain a number of bits defined in accordance with a particular system 100 application. In this regard, it should be appreciated that each data block may contain a fixed number of bits. Alternatively, each data block may contain a variable number of bits, ranging from a minimum size to a maximum size. In this regard, it should be understood that the number of bits contained in each data block may be varied in a deterministic manner that is known by the decryption engine 160 but has random or pseudo-random characteristics.

Referring again to FIG. 1, the data block formatter 102 is further comprised of hardware and software configured to communicate data blocks to the RNS encoder 104. The RNS encoder 104 is comprised of hardware and software configured to receive data blocks from the data block formatter 102. The RNS encoder 104 is also comprised of hardware and software configured to convert the data in the data blocks into a residue number system (RNS) representation via modulo operations. In this regard, it should be understood that the conversion of data into a RNS representation can be done on a character by character basis, i.e. in a data block by data block manner. It should also be understood that modulo operations can be performed in parallel to decrease a processing time of the RNS encoder 104. Modulo operations are well known to persons skilled in the art. Thus, such operations will not be described in great detail herein. However, it should be appreciated that any modulo operation known in the art can be used without limitation, provided that the residue number system employed has moduli m_0, $m_1 \ldots m_{N-1}$ which are selected as relatively prime numbers. The phrase "relatively prime numbers" as used herein refers to numbers having a greatest common divisor of one (1).

According to an embodiment of the invention, binary-to-residue conversion operations are computed. In such a scenario, it is preferable to implement moduli equal to or near a power of two (2). For example, if the RNS encoder 104 is required to employ a relatively simple binary to RNS conversion method, then the moduli used by the RNS encoder 104 can be advantageously selected as 2^N, 2^N-1, and 2^N+1. As should be understood, such moduli are preferable because they are guaranteed to be relatively prime and have the lowest possible implementation complexity. Still, the invention is not limited in this regard.

Examples of binary-to-residue circuits for moduli 2^N, 2^N-1, and 2^N+1 are disclosed in "Modulo-Free Architecture for Binary to Residue Transformation with respect to {2m-1, 2m, 2m+1} Moduli Set, 1994," written by F. Pourbigharaz, H. M. Yassine.

The following Example (1) is provided to assist a reader in understanding a computed modulo operation for converting binary data into a RNS representation. However, the scope of the invention is not limited in any way thereby.

EXAMPLE 1

A first data block is comprised of a binary bit sequence 00001001 representing a horizontal tab, the tenth character of the ASCII character set. As will be understood by a person skilled in the art, the binary bit sequence 00001001 has a decimal value of nine (9). As the Extended ASCII character set has 256 possible values with decimal values from zero (0) to two hundred fifty-five (255) and corresponding binary values from 00000000 to 11111111, the moduli set must supply a dynamic range $M = m_0, m_1, \ldots, m_{P-1}$. P is the number of moduli of at least two hundred fifty-five (255). Three (3) moduli m_0, m_1, m_2 are selected to have values of seven (7), eight (8), and nine (9). This gives a dynamic range of five hundred four (504). Accordingly, the RNS representation of the data in the first data block can be computed using Equation (1).

$$R = \{V_{bbs} \text{ modulo } m_0,\ V_{bbs} \text{ modulo } m_1,\ V_{bbs} \text{ modulo } m_2\} \quad (1)$$

where R is a residue value 3-tuple for the data in a data block. V_{bbs} is a value of a binary bit sequence representing a character of the English language. m_0, m_1, m_2 are the moduli. By substituting the above-listed values for V_{bbs} and m_0, m_1, m_2 into Equation (1), the residue values for the data in the first data block can be computed as follows: R = {9 modulo 7 = 2, 9 modulo 8 = 1, 9 modulo 9 = 0} having a binary bit sequence of {010 001 000}.

According to another embodiment of the invention, binary-to-residue conversion operations are advantageously implemented in look-up table operations. This embodiment is especially for block sizes of ten (10) bits or less due to the manageable look-up table address space. In such a scenario, the RNS encoder 104 has a relatively fast processing time as compared to that of a RNS encoder employing a computational binary to RNS conversion algorithm. Still, the invention is not limited in this regard. Any other binary-to-residue conversion technique known in the art can be used without limitation.

Referring again to FIG. 1, the RNS encoder 104 is also comprised of hardware and software configured to communicate residue values expressed in a binary number system representation to the closed-ring-arithmetic operator 106. The closed-ring-arithmetic operator 106 is comprised of hardware and software configured to receive residue values from the RNS encoder 104. As shown in FIG. 1, the closed-ring-arithmetic operator 106 is coupled to the numerical sequence generator 110. The numerical sequence generator 110 is comprised of hardware and software configured to generate an error generating sequence. The phrase "error generating sequence" as used herein refers to a sequence of serially linked numbers expressed in a binary number representation, i.e. a sequence of serially linked bit segments having n bits of a zero value (0) or a one (1) value. In this regard, it should be appreciated that the numerical sequence generator 110 is coupled to the state synchronizer 112. State synchronizers are well known to persons skilled in the art. Thus, the state synchronizer 112 will not be described in great detail herein. However, it should be appreciated that the state synchronizer 112 is comprised of hardware and software

configured to accept a "key." As should be appreciated, the key is provided for enabling the generation of the identical of error generating sequence at the receiver for decrypting the message. The "key" is also provided for enabling the induction of errors in selected residues.

Referring again to FIG. 1, the state synchronizer 112 is configured to communicate the "key" to the numerical sequence generator 110. Numerical sequence generators are well known to persons skilled in the art. Thus, the numerical sequence generator 110 will not be described in great detail herein. However, it should be appreciated that the numerical sequence generator 110 performs actions to generate an error generating sequence with an initial state defined in a received "key." Such error generating sequence types include, but are not limited to, a numerical sequence in a weighted number system representation or a numerical sequence in a residue number system representation. As will be understood by a person skilled in the art, the type of error generating sequence can be defined by a complexity and robustness of the system 100. In this regard, it should be appreciated that the error generating sequence can be selected as a simple pseudo random number sequence, a chaotic sequence, or a complex nonlinear sequence.

Referring again to FIG. 1, the numerical sequence generator 110 is also comprised of hardware and software configured to communicate an error generating sequence to the closed-ring-arithmetic operator 106. Upon receipt of one or more residue values from the RNS encoder 104 and an error generating sequence from the numerical sequence generator 110, the closed-ring-arithmetic operator 106 performs actions to induce errors in at least one of the residue values by using the error generating sequence. In this regard, it should be appreciated that the closed-ring-arithmetic operator 106 induces errors in residue values by performing an arithmetic operation, such as a Galois field addition, a Galois field subtraction, a Galois field multiplication, a Galois extension field multiplication, or a related operation on a subset of Galois sub-fields. For example, the closed-ring-arithmetic operator 106 performs actions to add in Galois field $GF[m_L]$ all or a portion of the error generating sequence to residue values thereby inducing errors in the residue values.

According to an aspect of the invention, each residue value is expressed in a binary number system representation. Specifically, each residue is a number expressed as a sequence of bits. Each bit of the sequence has a zero (0) value or a one (1) value. In this regard, it should be appreciated that the arithmetic operation performed by the closed-ring-arithmetic operator 106 modifies one or more bits of each residue value or selected residue values in Galois field GF[m_L]. It should also be appreciated that the selected residue values in which errors are induced may be varied over time.

After inducing errors in one or more residue values, the closed-ring-arithmetic operator 106 communicates non-error induced residue values (NEIRVs) and/or error induced residue values (EIRVs) to the data formatter 108. The phrase "non-error induced residue value" as used herein refers to a residue value absent of induced errors, i.e., none of the residue value bits were modified by an arithmetic operation performed by the closed-ring-arithmetic operator 106. In this regard, it should be appreciated that the arithmetic operation performed by the closed-ring-arithmetic operator 106 can modify one or more bits of selected residue values. Accordingly, the closed-ring-arithmetic operator 106 may communicate residue values (i.e., NEIRVs) other than the selected residue values having induced errors (i.e., EIRVs) to the data formatter 108. The phrase "error induced residue value" as used herein refers to a numerical value representing the sum of a residue value and all or a portion of an error generating sequence, the result of subtracting all or a portion of an error generating sequence from a residue value, or the product of a residue value and all or a portion of an error generating sequence, i.e, one or more bits of the residue value was modified by an arithmetic operation performed by the closed-ring-arithmetic operator 106.

The data formatter 108 is comprised of hardware and software configured to convert the NEIRVs and the EIRVs into a proper form for transmission to a data store 114 and/or the decryption engine 160. For example, the data formatter 108 can perform actions to interleave the NEIRVs and the EIRVs. Subsequent to the formatting step, the NEIRVs and the EIRVs are stored or transmitted to some remote

location. For example, the NEIRVs and the EIRVs can be transmitted to the data store 114 and/or the decryption engine 160. The data formatter 108 can also perform actions to convert the NEIRVs and the EIRVs to a weighted number system representation prior to formatting the same for transmission to the data store 114 and/or the decryption engine 160. In this regard, it should be appreciated that the data formatter 108 can employ a Chinese Remainder Theorem (CRT) algorithm, a modified CRT algorithm, a mixed-radix conversion (MRC) algorithm, or a modified MRC algorithm to convert the NEIRVs and the EIRVs into a weighted number system representation prior to formatting and transmission to the data store 114 and/or the decryption engine 160.

As shown in FIG. 1, the decryption engine 160 is comprised of an inverse data formatter 116, a closed-ring-arithmetic operator 118, and a Chinese Remainder Theorem (CRT) processor 124. The decryption engine 160 is also comprised of a numerical sequence generator 120 and a state synchronizer 122. The inverse data formatter 116 is comprised of hardware and software configured to convert a received signal including interleaved data (i.e., NEIRVs and EIRVs) into a form suitable for communication to the closed-ring-arithmetic operator 118. The inverse data formatter 116 can extract the data that has been interleaved and perform any other actions necessary to extract NEIRVs and EIRVs from a formatted signal communicated from the data formatter 108. If the NEIRVs and the EIRVs are converted to a weighted number system prior to transmission by the data formatter 108, then the inverse data formatter 116 can also perform actions to convert the same into an unweighted number system representation. The phrase "unweighted number system" as used herein refers to the residue number system.

Referring again to FIG. 1, the state synchronizer 122 performs actions to synchronize at least one state of the numerical sequence generator 120 with at least one state of the numerical sequence generator 110. In this regard, it should be appreciated that the state synchronizer 122 generates a "key" and communicates the same to the numerical sequence generator 120. The "key" is provided for enabling the generation of the same error generating sequence employed by the encryption

engine 150. Upon receipt of the "key," the numerical sequence generator 120 performs actions to generate the same error generating sequence generated by the numerical sequence generator 110 of the encryption engine 150. Thereafter, the numerical sequence generator 120 performs actions to communicate the error generating sequence to the closed-ring-arithmetic operator 118.

Upon receipt of NEIRVs and EIRVs from the inverse data formatter 116 and the error generating sequence from the numerical sequence generator 120, the closed-ring-arithmetic operator 118 performs actions to remove the error previously induced in the EIRVs by the closed-ring-arithmetic operator 106. In this regard, the closed-ring-arithmetic operator 118 inverts the arithmetic operations performed by the closed-ring-arithmetic operator 106 of the encryption engine 150. Once the errors are removed from the EIRVs, the error free residue values (including any NEIRVs) are communicated from the closed-ring-arithmetic operator 118 to the CRT processor 124. The CRT processor 124 performs arithmetic operations and/or look up table operations to obtain an output data from the error free residue values. In this regard, it should be appreciated that the CRT processor 124 is comprised of hardware and software configured to recover sequences of binary bits representing characters of the English language from the error free residue values by employing a Chinese remainder theorem (CRT) algorithm. CRT algorithms are well known to persons skilled in the art. Thus, CRT algorithms will not be described in great detail herein. However, it should be appreciated that any CRT algorithm can be used without limitation. It should also be appreciated that the output data is comprised of substantially the same sequence of binary bits contained in the input signal received at the encryption engine 150.

Referring now to FIG. 2, there is provided a flow diagram of a method 200 for encrypting and decrypting data that is useful for understanding the invention. As shown in FIG. 2, the method 200 begins at step 202 and continues with step 204. In step 204, an input data expressed in a weighted number system representation is received at an encryption engine 150. After step 204, step 206 is performed where the input data is processed to form two or more data blocks. In this regard, it should be

appreciated that the data blocks may contain a fixed number of bits or a variable number of bits ranging from a minimum size to a maximum size. Subsequently, the method 200 continues with step 208. In step 208, data in the data blocks is converted into a residue number system representation via modulo operations. Modulo operations are well known to persons skilled in the art. Thus, modulo operations will not be described in great detail herein. However, it should be appreciated that modulo operations are selected in accordance with a particular method 200 application.

Referring again to FIG. 2, the method 200 continues with step 210. In step 210, a "key" is generated for enabling a generation of a particular type of error generating sequence. In step 212, the "key" is communicated to a numerical sequence generator 110. Upon receipt of the "key," the numerical sequence generator 110 generates an error generating sequence of a type defined in and synchronized by the "key." Such types of error generating sequences include, but are not limited to, a numerical sequence expressed in weighted number system representation or a numerical sequence expressed in a residue number system representation. In step 216, a closed-ring-arithmetic operation is performed on the residue values generated in a residue number system using the error generating sequence. The closed-ring-arithmetic operation can be a Galois field addition, a Galois field subtraction, or a Galois field multiplication. For example, all or a portion of the error generating sequence is added to the residue values thereby inducing errors in the residue values. Thereafter, the error induced residue values (EIRVs) are processed to format the same for storage or transmission in step 218. Such formatting can include converting the EIRVs and NEIRVs to a weighted number system. Such formatting can also include interleaving the EIRVs and NEIRVs. Such formatting can further include any other formatting steps to prepare the same for storage or transmission. In step 220, the formatted encrypted data are communicated to a data store 114 and/or a decryption engine 160 with or without conversion to a weighted number system representation. Subsequently, the method 200 continues with step 222 of FIG. 2B.

In step 222, a decryption engine 160 receives the formatted encrypted data. Upon receipt of the formatted encrypted data, the decryption engine 160

performs processing on the same. This processing can involve deformatting the formatted encrypted data so that the EIRVs and NEIRVs are in a proper form for communication to a closed-ring-arithmetic operator 118. This step can involve de-interleaving the EIRVs and NEIRVs from an interleaved sequence. This step can also involve converting the EIRVs and NEIRVs to an unweighted number system. After step 224, step 226 is performed where at least one state of a numerical sequence generator 120 contained in the decryption engine 160 and at least one state of a numerical sequence generator 110 contained in an encryption engine 150 are synchronized. In step 228, an error generating sequence is generated by the numerical sequence generator 120. In this regard, it should be appreciated that the error generating sequence is the same as the error generating sequence generated by the encryption engine 150. It should also be appreciated that the error generating sequence is generated in a weighted number system or a residue number system. Subsequently, a closed-ring-operation is performed to remove the error induced in the EIRVs to obtain error free residue values. In step 232, Chinese remainder theorem (CRT) operations and/or CRT look-up table operations are performed to obtain output data from the error free residue values. In this regard, it should be appreciated that the output data is comprised of the same or substantially the same sequence of binary bits contained in the input data. After step 232, step 234 is performed where the method 200 ends.

The following Example (2) is provided in order to illustrate a principle of the present invention. MATLAB® software is used to illustrate the principle of the present invention. The scope of the invention, however, is not to be considered limited in any way thereby.

EXAMPLE 2

Briefly, an input data "the quick brown fox" is generated and converted into a residue number system representation using modulo operations. The residue number system employed is selected to have moduli three (3), five (5), and nineteen (19). Thereafter, a manipulation in residue space is performed by simply

flipping the least significant bit (LSB) in the mod three (3) moduli illustrating the principle that a very predictable manipulation of the same bit in residue space manifests itself as a statistically complex behavior in a weighted number representation of characters. The simple toggling of an LSB in a single residue thereby generates output characters Þ□ÔÈÂÞÑÎÞÅÎ×. Subsequently, the characters are converted back to a residue number system representation. A manipulation in residue space is performed by flipping an LSB in the mod three (3) moduli thereby generating an output data "the quick brown fox."

```
%
% Define primes
%
p(1) = 3;
p(2) = 5;
p(3) = 19;
%
% Calculate M
%
capm = prod(p);
%
% Define b's
%
b(1) = 2;
b(2) = 3;
b(3) = 14;
%
% Define M/p's
%
mop(1) = 5*19;
mop(2) = 3*19;
mop(3) = 3*5;
%
% Define input string
%
str ='The quick brown fox';
seq = uint8(str);
intct = length(seq);
%
%Begin conversion into residue space
%
resarr = zeros(3,intct);
```

```
for ind = 1:intct
   for ind1 = 1:3
      resarr(ind1,ind) = mod(seq(ind),p(ind1));
   end
end
%
% Insert encryption fcn here.
%
for ind = 1:intct
   bv = dec2bin(resarr(1,ind),2);
   if bv(1) == '0'
      bv(1) = '1';
   else
      bv(1) = '0';
   end
   resarr(1,ind) = bin2dec(bv);
end
%
% Convert back to weighted
%
outarr = zeros(1,intct);
for int = 1:intct
   outarr(int) = mod(resarr(1,int)*b(1)*mop(1) + ...
      resarr(2,int)*b(2)*mop(2) +resarr(3,int)*b(3)*mop(3),capm);
end
outarr = uint8(outarr);
eout = char(outarr);
eout
%
% Now decript
%
seq = uint8(eout);
intct = length(seq);
%
%Begin conversion into residue space
%
resarr = zeros(3,intct);
for ind = 1:intct
   for ind1 = 1:3
      resarr(ind1,ind) = mod(seq(ind),p(ind1));
   end
end
%
% Insert dencryption fcn here.
%
```

```
for ind = 1:intct
   bv = dec2bin(resarr(1,ind),2);
   if bv(1) == '0'
      bv(1) = '1';
   else
      bv(1) = '0';
   end
   resarr(1,ind) = bin2dec(bv);
end
%
% Convert back to weighted
%
outarr = zeros(1,intct);
for int = 1:intct
   outarr(int) = mod(resarr(1,int)*b(1)*mop(1) + ...
      resarr(2,int)*b(2)*mop(2) +resarr(3,int)*b(3)*mop(3),capm);
end
outarr = uint8(outarr);
eout = char(outarr);
eout
```

The following Example (3) is provided in order to further illustrate the mathematical aspects of the present invention. The scope of the invention, however, is not to be considered limited in any way thereby.

EXAMPLE 3

Briefly, an input data is generated and converted into a residue number system representation using modulo operations. The input data includes:

"In classical physics, due to interference, light is observed to take the stationary path between two points; but how does light know where it`s going? That is, if the start and end points are known, the path that will take the shortest time can be calculated. However, when light is first emitted, the end point is not known, so how is it that light always takes the quickest path? In some interpretations, it is suggested that according to QED light does not have to - it simply goes over every possible path, and the observer (at a particular location) simply detects the mathematical result of all wave functions added up (as a sum of all line integrals). For other interpretations, paths are

viewed as non physical, mathematical constructs that are equivalent to other, possibly infinite, sets of mathematical expansions. According to QED, light can go slower or faster than c, but will travel at speed c on average.

Physically, QED describes charged particles (and their antiparticles) interacting with each other by the exchange of photons. The magnitude of these interactions can be computed using perturbation theory; these rather complex formulas have a remarkable pictorial representation as Feynman diagrams. QED was the theory to which Feynman diagrams were first applied. These diagrams were invented on the basis of Lagrangian mechanics. Using a Feynman diagram, one decides every possible path between the start and end points. Each path is assigned a complex-valued probability amplitude, and the actual amplitude we observe is the sum of all amplitudes over all possible paths. Obviously, among all possible paths the ones with stationary phase contribute most (due to lack of destructive interference with some neighboring counter-phase paths) – this results in the stationary classical path between the two points.

QED doesn`t predict what will happen in an experiment, but it can predict the probability of what will happen in an experiment, which is how it is experimentally verified. Predictions of QED agree with experiments to an extremely high degree of accuracy: currently about 10^-12 (and limited by experimental errors); for details see precision tests of QED. This makes QED the most accurate physical theory constructed thus far.

Near the end of his life, Richard P. Feynman gave a series of lectures on QED intended for the lay public. These lectures were transcribed and published as Feynman (1985), QED: The strange theory of light and matter, a classic non-mathematical exposition of QED from the point of view articulated above."

The residue number system employed is selected to have moduli two (2), seven (7), and seventeen (17). Thereafter, a manipulation in residue space is performed by simply flipping the least significant bit (LSB) in the mod two (2)

moduli.

For simplicity of illustration, a simple normally distributed random variable is generated with the number of samples equal to the number of eight (8) bit blocks representing the input data. In other embodiments a more complex method of generating the error sequence is used. The first three (3) fractional bits of each random number is restricted to value from zero (0) to six (6) and are added in Galois field GF[7] to the mod seven (7) moduli. The next 5 (5) fractional bits from the random sample is limited to values from zero (0) to sixteen (16) and are added in Galois field GF[17] to the mod seventeen (17) moduli. The result is reformatted via the Chinese Remainder Theorem (CRT) to a weighted eight (8) bit character representation.

The simple operations in residue space generate output characters:

Ø□uÎ³à□4-□à½+Ui□À□□\Coo¸*éS&[P□□\}Ñ8□<×V¶'

×´□□□Y□□·,'*□Q□4¿□}□.□

hæÏáODS□çÄ□ÆV□¤#NsáÄ□□³cÞå□ L□y[□è□G(À#×Ü^□^□é□{ço□8u

N□È□

□àÙ}6□OºÚsÆÓØ-

~□¶)ãØÙ\□$□□□/□□□»[b")□B¿Ïc±□åzu□A□□Î×Ç□□uQç]ÀÌ¯Ý4□@Ï□□2+ä

Æ□□□□□¶¯9Ò□3M}Ëê)□~ÌKÛ/UÀ:J#51*CÌs¢åG^□Ñ#CntÈ?□□□□e.XÐN¿□T

Ý□u□□□ß□¾Äp□ÜQ□□}tM□A¢Ð□-/□□w□¶{ØE□D<êuS6#Ï/□□$!□

ãÝ¹&äË×<□äËÝÜ[□□W&.E□ßz²Û-

f□}-FgPç/+ãZÏí!Î(¡;□□=ÜpÄ4□å□Ü-□oÍÓ"·jn$`@

Rå83H□OÒ§□ç□□JÐ.ádçíp;□§Ü□hÁS¼µ]ÌGËDìcÎ¾¢J>□©Dmy7mÚÝß□□êXãË

□i□5éàfd¨QÕKRFç□/□□Ä0□76»·{

Ù^am□□

S□/í"p sw□□\8lhåÆË_+®#@□Z×□@p&P_u□T+Zq«□=¬□5§M52ÁÚ□□È'ëÒ□

□□¼Ñ×`=¹f«Ã²□\Ñ¼□□í$□ºí□è□zÙ□:□□%Æ□)Þ)¶ª=

"Aá□□p□{æÐÊ×6t¸§Û□ºTÉ□□d![¨1#»□@ÝL%³L□Î□Â□q`?f1□\□□J

□Íf<□ÖKlx□□!×"')ÉYN/¶oj□ÅÖ□ÁjK□□□¡WÀ3iÎ□{J□□~Ë□4□ÏÕtÉ□□ÚØCt

7Y#ÎÅZá

M2Ä½f°□ì¡□□ZÝ»

}YíM|□Þ®-□[]□v{©&6□□Ú~°□qÜÀ!Ö□y□Ì-Ú□Û%bEÇÐA

º\ÔrÔ9@Ñ"ÑÅ□i Í¤ß¥□MÙÑÓ6 Æ□oi□□□iÙ□U□!f□M2 ?Ð□-a□2ÎÁ'

>^3°²ÂÏÊØÅ□p□Þ□□#r□□P□4□YÍ@yàÙO□□°¬Ou□ÏÌFÉ□Ï□Ä¹G&z¯Þ-

□¢DkE¾ÝGçt¬£,{Á}1□µ,´u·□ÄÛß¾Ú×

kµ□~]|m□:□[®5NÈ□□

×à□Câ□□□□&□ÅÜ□M£X □B0gS0□□È□Oc-.¢^ÕjÏ6É□□□j□mÕVÛ=□□

ä2EÅh]□}ÅG@2³□□Ei4åZußH□×(~F)¾kå□%È|)

¿□·\□□□×ä$~□Í□E□¶□)©Ö□×Ó□□Ì¥

yHs□¦E

ÉxuÅ□□ØÇÎµÍ%½b;æèÊaÈ¯Ð□à□□¯CÍ□r□X+PÅ=pã|¤Ï|N□ìÍnä□ÙGÈ□¼V K

½<□*□□Jg¶q□□!Spo3Ì=Ú&ÕU{□|Ò=Ü

□□□<®¬3C"ac□ã7BfÏ□°□q1äëT□□°´□Ë

Ù □ÎßÁ¶□□#ä□;YL{i0°Ih½Æh□=□□íS¨ÂÀã©Ü□ÆJÃÝ®Æc[¸ÊádBË□×□□ÊÝ-

□□□7□*9zv&`a[Ã*?□□Ì□a²□-8/Öê □#ëÖ□-

□Ï□wÌ¾:ËTÜh¯²J1Ôß¿□EJ□□□×□×,L§□QéZ□¾

gÔ□àn□.\¿N]äc©Ö□×´yß□£Yo=

□□*$4Ö7@□©A*

u

it□~#ÊTÈDi□3ºhÄ5□□□êç´):1·ÌèkÔÒ%ái8□l□jxiái©À□ÁäÒls□¬?F

□ÍÊÝm~àÌ□c□Èx?°□Ï%ét□V²æG7□Ç°5□C~}m□Ò□□ß6ß□[=fH9¿□□□

Æ$ÌÁ

{□Ca¯ÈL=Ì□0x□Ï□□½Z□æª□□i□¿Ò5Vs□ÂYÃã□¼¿n□Ô§□□Ö©ÄTÁ

□&=□å□□©ªc/¡ÙË&cÄ&m□j)¯(ÔWçx□□±□Ì□

y,;□Î#□®9 Rw□Ùè¶fct×ÁÚi¡Ý66+w

í□YÞ□]|?º%§*1åÝ],À□B»K"□»âÍ=VayB9A¬f□□G□¯ß□6"O&ÁÈ□5MÓ8Þ□ncÉ

n□i*½s`q£ã>□□%.¾SÚ□3□í.CBBMè{□ÁmMÒ

4ãÖ°á}?ÖÅ□ÉJq.□□¿o¡ÍØ\C□@□'~" □&)□¿zR□□Ò&ÅÂÇÔv¶g□¾□

f□Ä□Ü.□`z□Ë©|

□&)+z2v□É$□~□d□Jè|□□□e.□+□□ÖíKÄ©Á<ÚqÈ□+¿4äÊm-¨□'gx¡□A¿Û1|#f=`

¯ÒN}

Ù□□Â7□□.°□Ê□□i^S□q□¨Ã2.2¡éu□X{É□□|AN□US<

à2\RSm□¸Ò¼+iq[á7fÛÚB□□x±.§□Â6□É#¥^7}|ç¡□!Ý□□sÀ/AZ□□Èèmm

¡·íC*1

n²ã¯ÓÛÆ□:ÊÛ»Eç*CÓË

´uP□¶=nY°·Y□ebk'èèÓÛ□□°□□\µßJ□□Ú£y□□g¬□6

æ[i-(z¹f□_i,□Û□ì8□ □+ÆR□ægÙ0C□>Ì-

¯lMaà□©JÃ¼åâÞ'S1_é□□"<a□PuL□ºçKdêMl

-□□É□rÝiß$q±□ÁVØ□S5HÑµ$³□7□□éq-□□ □□ÏWÞ´-

+g□□!Ö0)□Ñ¨n.B,□ªà`Y=,~

34□wsÔ□Pq6há¯□M□Ö□□/MHg¥Í□^Øì□ÀqCÉJ·Æ□×Le,ä6±!□q□tÁY□IÎRÐ¿

©□ã53□□*Ü+L□ê□L□}¿z&äz(x}MEfZWÒ□.O=Ú$A□/³□

¨E@8.□Þ;|□□äÕ□Ö□ØÈD)§x□j□«Hk^¡=KnÄÒB

[O□¶&3

□¶ÈÉ□□ËÆÛ□□¡-3LÖ[{j□!ÔÎ□b®kGçwzÓ~ÉYm□9ÓË□`□=DaTë·g□X'ÄÇ3c

#o!$¦□□V]=Ò□[L□*□¢@3~□□¢□²'4`©\e~Äw4Û□v□□?--Ã□çßlY□k□

¡□Å-□É□□g□Ö#'R¬□åi±²RÖ□=□P□□

□à6ÈÇ°Á□□¨kz2ÉÚÅ*□m¶íå·íjÛ-"`Ý□Å»¼□c□*êÔpEÑqÈª1nO-2,E□×{Ç□□

□□

□ÏÉÕ□

å

EVÑ'□ÌÐË'[J©

É□®

Gl{□wBÝÔ×ÒhäÔ-Ã□Úç□"□'¡,□ªg□²Þ¥~4×uV~□Á¶*□=□gçV

½□q7zæ%Çe¬□□*$Ù□□V×9i0âF□□¥y ääà=4ËJ□©"g0□rß.

In the decryption engine, the characters are converted back to a residue number system representation. A manipulation in residue space is performed by flipping an LSB in the mod two (2) moduli. The seed used to generate the random

number sequence in the encryption is used as the key and seed for the random number generator in the decryption engine. The first three (3) fractional bits of each random number is restricted to value from zero (0) to six (6) and are subtracted in Galois field GF[7] to the mod seven (7) moduli. The next five (5) fractional bits form the random sample is limited to values from zero (0) to sixteen (16) and are subtracted in Galois field GF[17] to the mod seventeen (17) moduli. The result is converted back to character format via the Chinese Remainder Theorem (CRT). Consequently, an output data is generated which corresponds to the original input data.

For purpose pf understanding the invention, the following example is provided of a MATLAB® software routine for residue encryption and decryption. Those skilled in the art will appreciate that the invention is not limited to the routine provided.

```
%
% MATLAB example routine for Residue encryption
%
% Written by: David B. Chester
% Harris Corporation
%
% 4/9/2007
%
% Define primes
%
p(1) = 2;
p(2) = 7;
p(3) = 17;
%
% Calculate M
%
capm = prod(p);
%
% Define b's
%
b(1) = 1;
b(2) = 6;
b(3) = 11;
%
% Define M/p's
%
mop(1) = 7*17;
```

```
mop(2) = 2*17;
mop(3) = 2*7;
%
% Define input string to be encrypted and then decrypted
%
str = ['In classical physics, due to interference, light is observed to'...
   ,char(13),...
'take the stationary path between two points; but how does light'...
,char(13),...
'know where it`s going? That is, if the start and end points are'...
,char(13),...
'known, the path that will take the shortest time can be'...
,char(13),...
'calculated. However, when light is first emitted, the end point'...
,char(13),...
'is not known, so how is it that light always takes the quickest'...
,char(13),...
'path? In some interpretations, it is suggested that according to'...
,char(13),...
'QED light does not have to - it simply goes over every possible'...
,char(13),...
'path, and the observer (at a particular location) simply detects'...
,char(13),...
'the mathematical result of all wave functions added up (as a sum'...
,char(13),...
'of all line integrals). For other interpretations, paths are'...
,char(13),...
'viewed as non physical, mathematical constructs that are'...
,char(13),...
'equivalent to other, possibly infinite, sets of mathematical'...
,char(13),...
'expansions. According to QED, light can go slower or faster than'...
,char(13),...
'c, but will travel at speed c on average.'...
,char(13),...
char(13),...
'Physically, QED describes charged particles (and their'...
,char(13),...
'antiparticles) interacting with each other by the exchange of'...
,char(13),...
'photons. The magnitude of these interactions can be computed'...
,char(13),...
'using perturbation theory; these rather complex formulas have'...
,char(13),...
'a remarkable pictorial representation as Feynman diagrams. QED'...
```

```
,char(13),...
'was the theory to which Feynman diagrams were first applied.'...
,char(13),...
'These diagrams were invented on the basis of Lagrangian'...
,char(13),...
'mechanics. Using a Feynman diagram, one decides every possible'...
,char(13),...
'path between the start and end points. Each path is assigned a'...
,char(13),...
'complex-valued probability amplitude, and the actual amplitude'...
,char(13),...
'we observe is the sum of all amplitudes over all possible' ...
,char(13),...
'paths. Obviously, among all possible paths the ones with'...
,char(13),...
'stationary phase contribute most (due to lack of destructive'...
,char(13),...
'interference with some neighboring counter-phase paths) - this'...
,char(13),...
'results in the stationary classical path between the two'...
,char(13),...
'points.'...
,char(13),...
char(13),...
'QED doesn`t predict what will happen in an experiment, but it'...
,char(13),...
'can predict the probability of what will happen in an experiment,'...
,char(13),...
'which is how it is experimentally verified. Predictions of QED'...
,char(13),...
'agree with experiments to an extremely high degree of'...
,char(13),...
'accuracy: currently about 10^-12 (and limited by experimental'...
,char(13),...
'errors); for details see precision tests of QED. This makes QED'...
,char(13),...
'the most accurate physical theory constructed thus far.'...
,char(13),...
char(13),...
'Near the end of his life, Richard P. Feynman gave a series of'...
,char(13),...
'lectures on QED intended for the lay public. These lectures'...
,char(13),...
'were transcribed and published as Feynman (1985), QED: The'...
,char(13),...
```

```
    'strange theory of light and matter, a classic non-mathematical'...
    ,char(13),...
    'exposition of QED from the point of view articulated above.'];
seq = uint8(str);
%
% Calculate the character length of the sequence
intct = length(seq);
%
% Read the current random number generator state and store it as the
% encryption key
key = randn('state');
%
% Generate a random sequence for encryption
% In this example, a simple normally distributed random number is used
normseq = randn(1,intct);
%
% Convert each floating point random number to a log2(p(2)) bit
% and a log2(p(3) bit representation
%
% Positives only
normseq = abs(normseq);
%
% Get the top p(2) fractional bits of the sequence for the p(2) field
% "error" generator.
% Limit the p(2) field values to 0 -> p(2) -1
frprt = normseq - floor(normseq);
errseq = round((p(2)-1)*frprt);
%
% Get the top p(2)+1 to p(2)+p(3) fractional bits of the sequence for
% the p(3) field "error" generator.
% Limit the p(3) field values to 0 -> p(3) -1
shf = 2^(ceil(log2(p(2))));
frprt = shf*normseq - floor(shf*normseq);
errseq1 = round((p(3)-1)*frprt);
%
%Begin conversion into residue space
%
% Initialize the residue values
resarr = zeros(3,intct);
% Initialize the output array.
outarr = zeros(1,intct);
%
% Calculate the residues
for ind = 1:intct
    for ind1 = 1:3
```

```
        resarr(ind1,ind) = mod(seq(ind),p(ind1));
      end
    end
    % Generate the errors for encryption
    %
    % In this implementation, the last bit (LSB) is inverted, the
    % next 3 LSBs are added to a random bit sequence in GF(p(2)),
    % and the 5 MSBs are added to another random bit sequence in GF(p(3)).
    for ind = 1:intct
      % Convert the p(1) RNS rep. to binary and invert it. This can only
      % be done because p(1) = 2 and is used to force a change although it is
      % not necessary.
      bv = dec2bin(resarr(1,ind),1);
      if bv(1) == '1'
        bv(1) = '0';
      else
        bv(1) = '1';
      end
      %
      % Convert the first residue back to binary.
      resarr(1,ind) = bin2dec(bv);
      %
      % Add the error to residue for p(2).
      resarr(2,ind) = mod(errseq(ind)+resarr(2,ind),p(2));
      %
      % Add the error for residue p(3).
      resarr(3,ind) = mod(errseq1(ind)+resarr(3,ind),p(3));
    %
    % Convert back to weighted using the CRT
    %
      outarr(ind) = mod(resarr(1,ind)*b(1)*mop(1) + ...
        resarr(2,ind)*b(2)*mop(2) + resarr(3,ind)*b(3)*mop(3),capm);
    end
    %
    % Convert the output to characters for output
    outarre = uint8(outarr);
    eout = char(outarre);
    eout
    %
    % Now decript
    %
    % Convert the encrypted input to a numerical array and calcuate its length.
    seq = uint8(eout);
    intct = length(seq);
    %
```

```
% Generate the same error sequence as generated for encryption
randn('state',key);
normseqd = randn(1,intct);
%
%Begin conversion into residue space
%
% Initialize the residue array.
resarr = zeros(3,intct);
% Initialize the output array.
outarr = zeros(1,intct);
%
% Convert to residues
for ind = 1:intct
   for ind1 = 1:3
      resarr(ind1,ind) = mod(seq(ind),p(ind1));
   end
end
%
% Convert each floating point random number to an 8 bit representation
%
%
% Convert each floating point random number to a log2(p(2)) bit
% and a log2(p(3) bit representation
%
% Positives only
normseq = abs(normseq);
%
% Get the top p(2) fractional bits of the sequence for the p(2) field
% "error" generator.
% Limit the p(2) field values to 0 -> p(2) -1
frprt = normseq - floor(normseq);
errseq = round((p(2)-1)*frprt);
%
% Get the top p(2)+1 to p(2)+p(3) fractional bits of the sequence for
% the p(3) field "error" generator.
% Limit the p(3) field values to 0 -> p(3) -1
shf = 2^(ceil(log2(p(2))));
frprt = shf*normseq - floor(shf*normseq);
errseq1 = round((p(3)-1)*frprt);
%
% Generate the errors for encryption
%
% In this implementation, the last bit (LSB) is inverted, the
% next 3 LSBs are added to a random bit sequence in GF(p(2)),
% and the 5 MSBs are added to another random bit sequence in GF(p(3)).
```

```
for ind = 1:intct
   % Convert the p(1) RNS rep. to binary and invert it. This can only
   % be done because p(1) = 2 and is used to force a change although it is
   % not necessary.
   bv = dec2bin(resarr(1,ind),1);
   if bv(1) == '1'
      bv(1) = '0';
   else
      bv(1) = '1';
   end
   %
   % Convert the first residue back to binary.
   resarr(1,ind) = bin2dec(bv);
   %
   % Subtract the error from residue for p(2) (inverse of the encryption
   % function used.
   resarr(2,ind) = mod(-errseq(ind)+resarr(2,ind),p(2));
   %
   % Subtract the error from residue for p(2) (inverse of the encryption
   % function used.
   resarr(3,ind) = mod(-errseq1(ind)+resarr(3,ind),p(3));
%
% Convert back to weighted using the CRT
%
   outarr(ind) = mod(resarr(1,ind)*b(1)*mop(1) + ...
      resarr(2,ind)*b(2)*mop(2) +resarr(3,ind)*b(3)*mop(3),capm);
end
%
% Convert the decrypted sequence to characters for output.
outarr = uint8(outarr);
eout = char(outarr);
% Display the decrypted sequence.
eout
```

FIG. 3 is a graph showing an autocorrelation of the input sequence.

FIG. 4 is a graph showing an autocorrelation of the encrypted sequence. The cross correlation resembles that of two (2) independent uniformly distributed random sequences, thus illustrating the independence of the two (2) statistics.

CLAIMS

1. Method for encrypting data, comprising:

formatting data represented in a weighted number system into a plurality of data blocks of N bits in length;

converting a decimal number defined by said data in each data block of said plurality of data blocks into a plurality of values, each representative of a residue of said decimal number modulo a given modulus among a plurality of moduli;

generating a first error generating sequence;

inducing errors exclusively in select values of said plurality of residue values by using said first error generating sequence to arithmetically operate on said select values using Galois arithmetic; and

converting said select values in which errors were induced and remaining values of said plurality of residue values in which errors were not induced to a weighted number system representation.

2. The method according to claim 1, wherein said step of inducing errors comprises introducing known errors in said select values generated in a residue number system.

3. The method according to claim 2, wherein said step of inducing errors further comprises performing an arithmetic operation on said plurality of residue values generated in a residue number system based on said first error generating sequence.

4. The method according to claim 1, further comprising generating said first error generating sequence in the weighted number system.

5. The method according to claim 1, further comprising generating said first error generating sequence in said residue number system.

6. The method according to claim 1, wherein said inducing step further comprises inducing errors in only selected residues.

7. The method according to claim 6, wherein said selected residues are varied over time.

8. The method according to claim 1, further comprising formatting said data to be stored or transmitted.

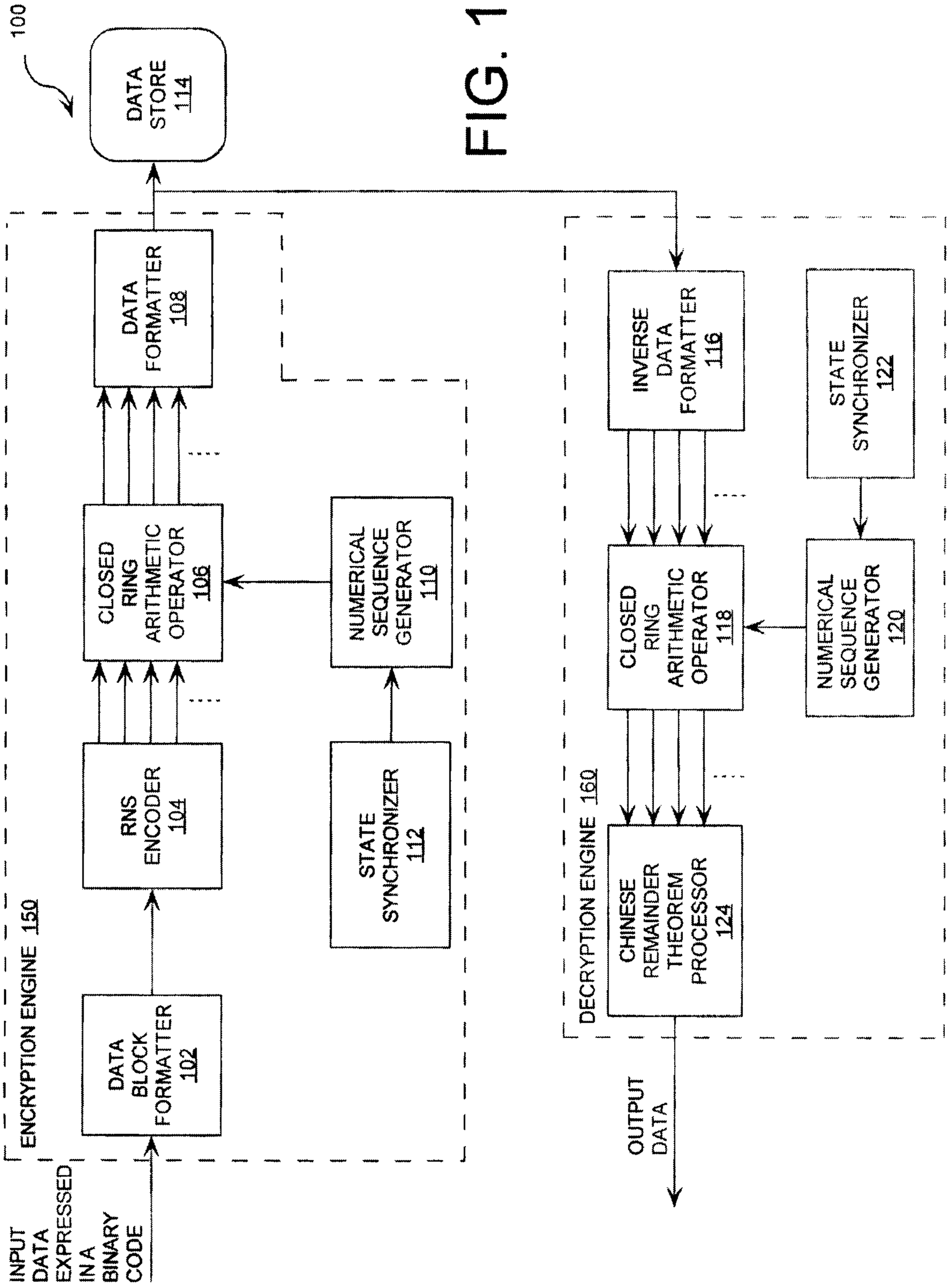
9. The method according to claim 1, further comprising:

receiving said plurality of data blocks after inducing said errors;

generating a second error generating sequence synchronized with and identical to said first error generating sequence; and

correcting said errors in said plurality of data blocks using an operation which is an arithmetic inverse of a process used in said inducing errors step.

FIG. 1


100
DATA STORE 114
ENCRYPTION ENGINE 150
DATA FORMATTER 108
CLOSED RING ARITHMETIC OPERATOR 106
NUMERICAL SEQUENCE GENERATOR 110
RNS ENCODER 104
STATE SYNCHRONIZER 112
DATA BLOCK FORMATTER 102
INPUT DATA EXPRESSED IN A BINARY CODE
DECRYPTION ENGINE 160
INVERSE DATA FORMATTER 116
STATE SYNCHRONIZER 122
CLOSED RING ARITHMETIC OPERATOR 118
NUMERICAL SEQUENCE GENERATOR 120
CHINESE REMAINDER THEOREM PROCESSOR 124
OUTPUT DATA

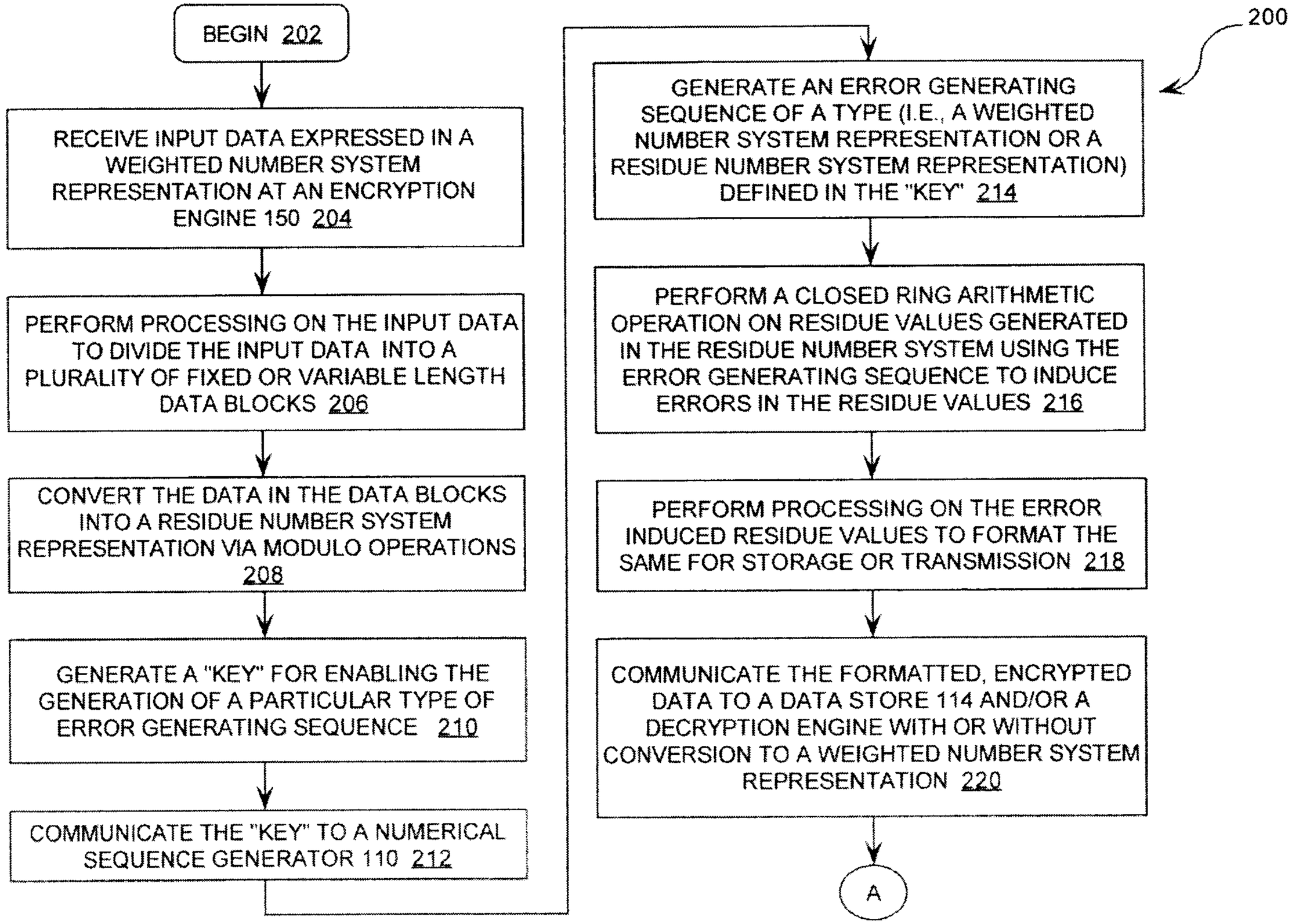


FIG. 2A

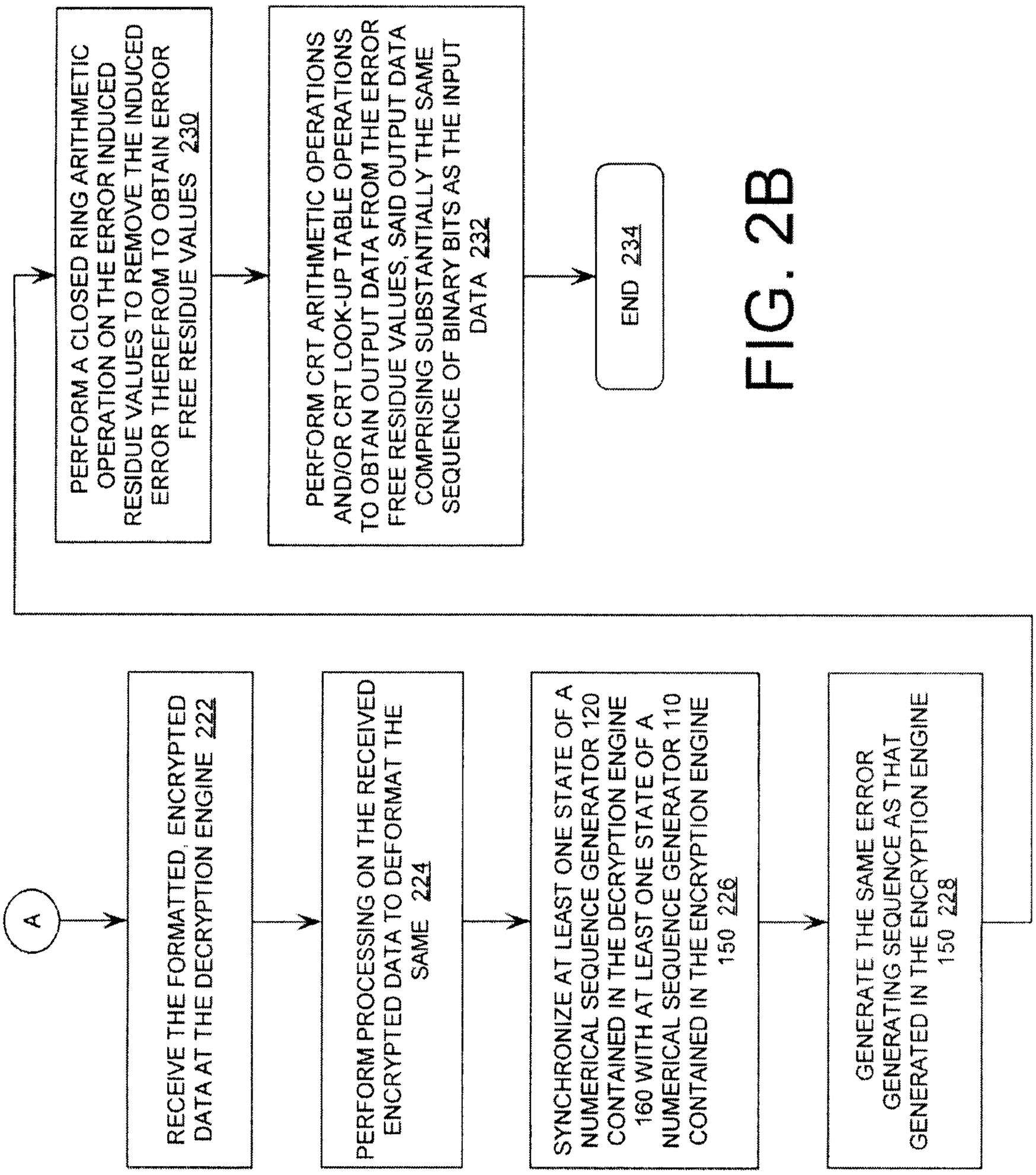

FROM FIG. 2A
A
RECEIVE THE FORMATTED, ENCRYPTED DATA AT THE DECRYPTION ENGINE 222
PERFORM PROCESSING ON THE RECEIVED ENCRYPTED DATA TO DEFORMAT THE SAME 224
SYNCHRONIZE AT LEAST ONE STATE OF A NUMERICAL SEQUENCE GENERATOR 120 CONTAINED IN THE DECRYPTION ENGINE 160 WITH AT LEAST ONE STATE OF A NUMERICAL SEQUENCE GENERATOR 110 CONTAINED IN THE ENCRYPTION ENGINE 150 226
GENERATE THE SAME ERROR GENERATING SEQUENCE AS THAT GENERATED IN THE ENCRYPTION ENGINE 150 228
PERFORM A CLOSED RING ARITHMETIC OPERATION ON THE ERROR INDUCED RESIDUE VALUES TO REMOVE THE INDUCED ERROR THEREFROM TO OBTAIN ERROR FREE RESIDUE VALUES 230
PERFORM CRT ARITHMETIC OPERATIONS AND/OR CRT LOOK-UP TABLE OPERATIONS TO OBTAIN OUTPUT DATA FROM THE ERROR FREE RESIDUE VALUES, SAID OUTPUT DATA COMPRISING SUBSTANTIALLY THE SAME SEQUENCE OF BINARY BITS AS THE INPUT DATA 232
END 234

FIG. 2B

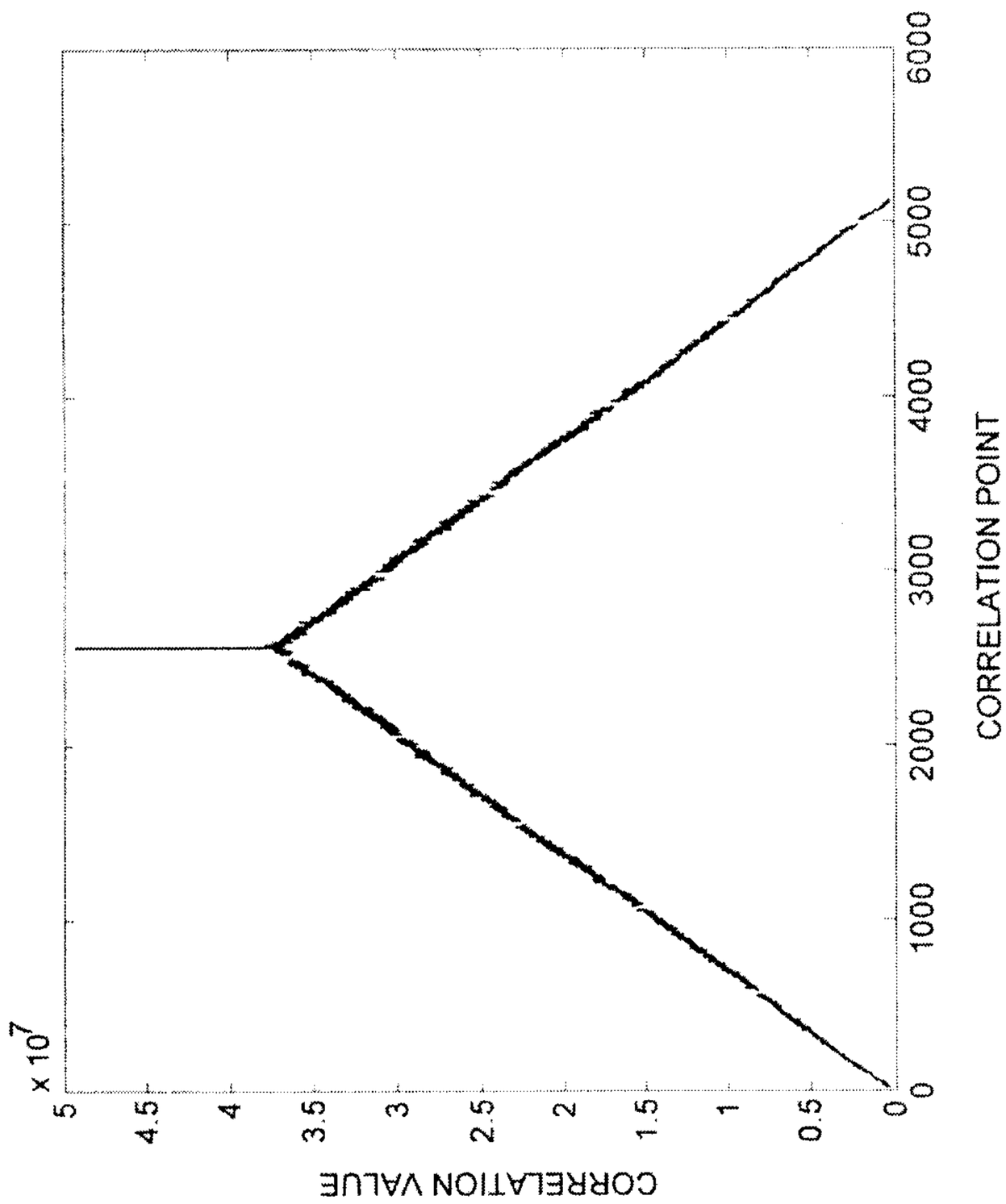

x 10^7
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
CORRELATION VALUE
0
1000
2000
3000
4000
5000
6000
CORRELATION POINT

FIG. 3

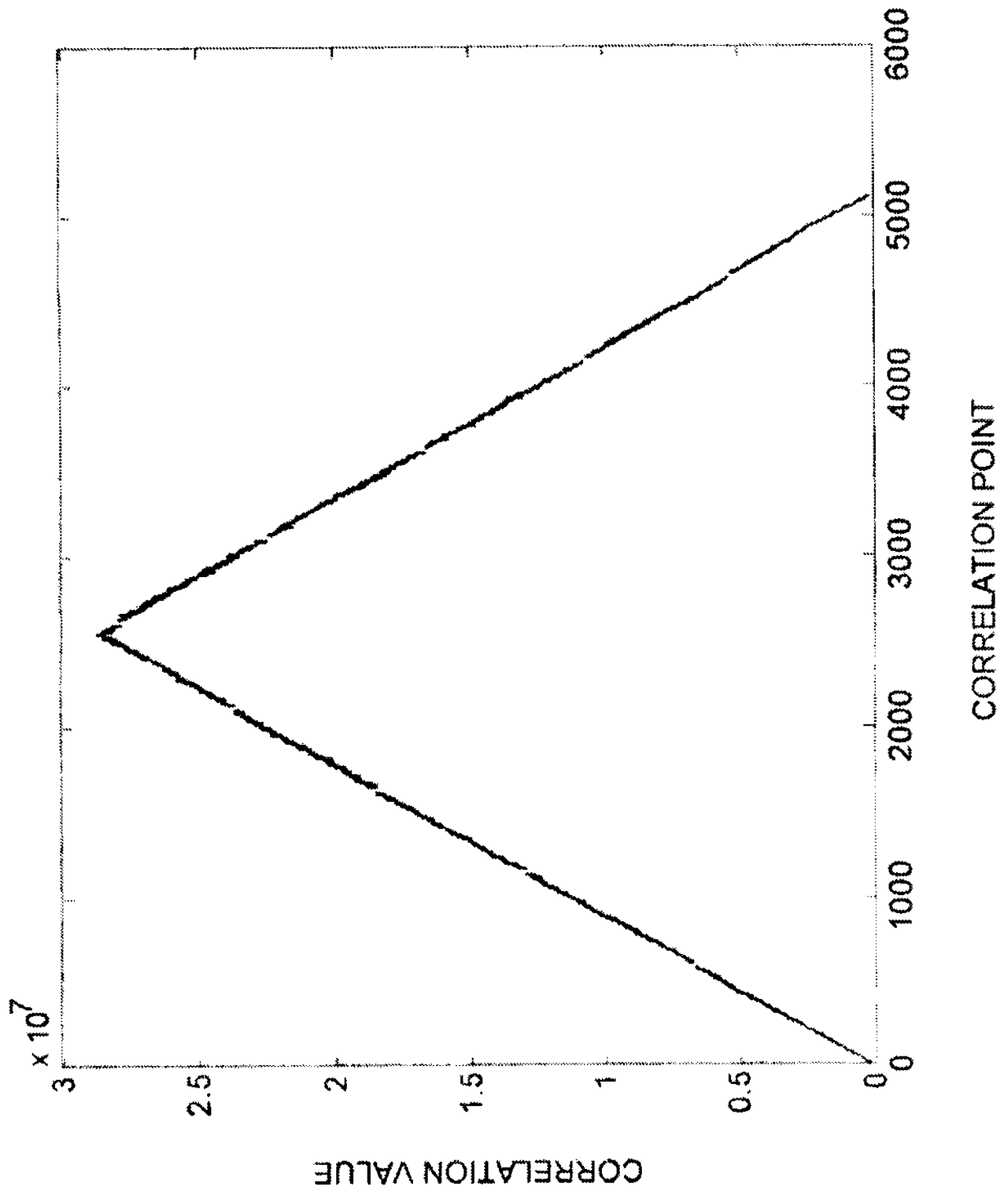
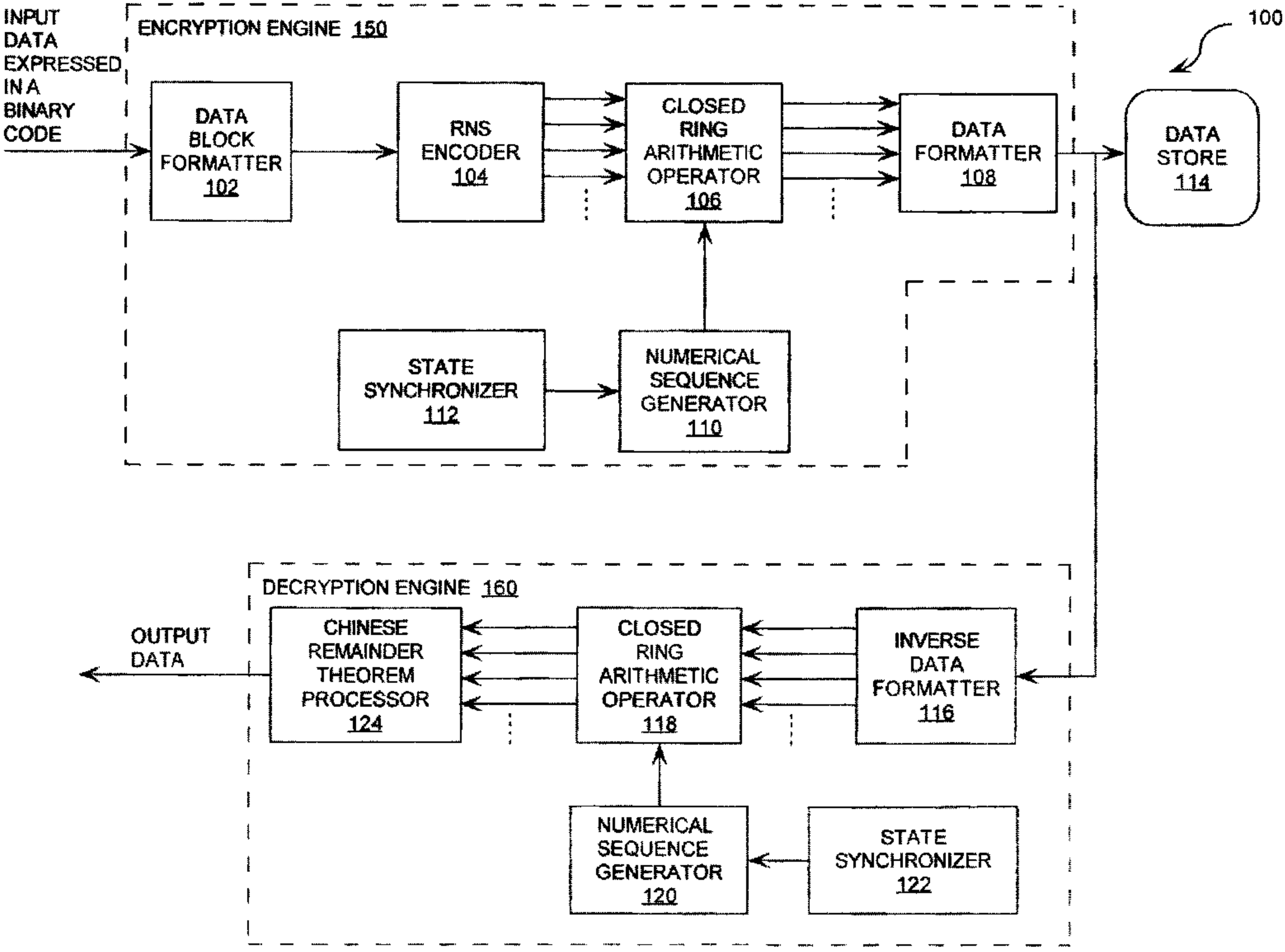

x 10^7
3
2.5
2
1.5
1
0.5
0
CORRELATION VALUE
0
1000
2000
3000
4000
5000
6000
CORRELATION POINT

FIG. 4


INPUT DATA EXPRESSED IN A BINARY CODE
ENCRYPTION ENGINE 150
100
DATA BLOCK FORMATTER 102
RNS ENCODER 104
CLOSED RING ARITHMETIC OPERATOR 106
DATA FORMATTER 108
DATA STORE 114
STATE SYNCHRONIZER 112
NUMERICAL SEQUENCE GENERATOR 110
DECRYPTION ENGINE 160
OUTPUT DATA
CHINESE REMAINDER THEOREM PROCESSOR 124
CLOSED RING ARITHMETIC OPERATOR 118
INVERSE DATA FORMATTER 116
NUMERICAL SEQUENCE GENERATOR 120
STATE SYNCHRONIZER 122