PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : H04L 9/26

A2

(11) International Publication Number: WO 99/16208

(43) International Publication Date: 1 April 1999 (01.04.99)

(21) International Application Number: PCT/US98/19804

(22) International Filing Date: 22 September 1998 (22.09.98)

(30) Priority Data:
08/934,582 22 September 1997 (22.09.97) US

(71) Applicant: QUALCOMM INCORPORATED [US/US]; 6455 Lusk Boulevard, San Diego, CA 92121 (US).

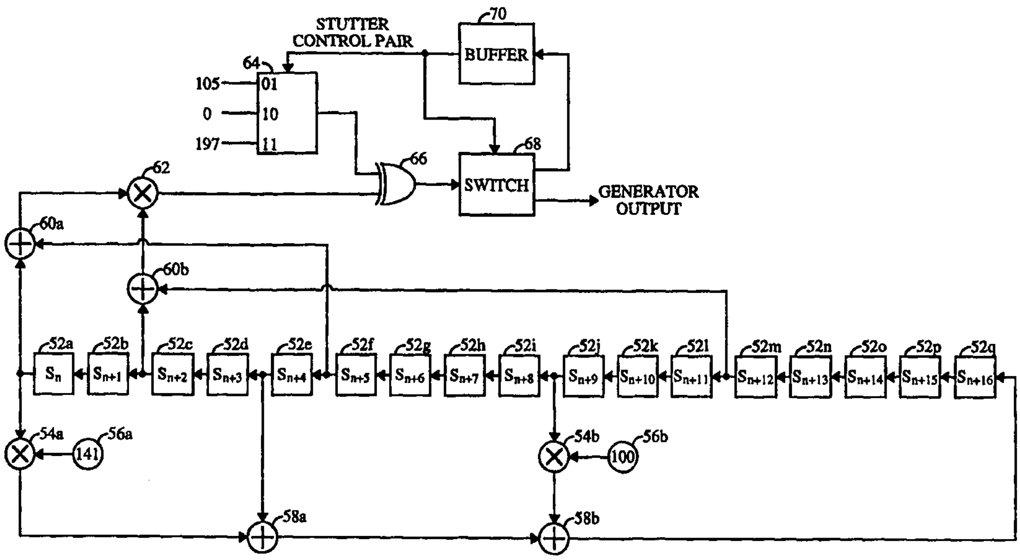
(72) Inventor: ROSE, Gregory, G.; 6 Kingston Avenue, Mortlake, NSW 2137 (AU).

(74) Agents: OGROD, Gregory, D. et al.; Qualcomm Incorporated, 6455 Lusk Boulevard, San Diego, CA 92121 (US).

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, HR, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published
Without international search report and to be republished upon receipt of that report.

(54) Title: METHOD AND APPARATUS FOR GENERATING ENCRYPTION STREAM CIPHERS



(57) Abstract

A method and apparatus for generating encryption stream ciphers. The recurrence relation is designed to operate over finite fields larger than GF(2). The linear feedback shift register (52) used to implement the recurrence relation can be implemented using a circular buffer (24a) or sliding a window (24b). Multiplications of the elements of the finite field are implemented using lookup tables. A non–linear output can be obtained by using one or a combination of non–linear processes. The stream ciphers can be designed to support multi–tier keying to suit the requirements of the applications for which the stream ciphers are used.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

METHOD AND APPARATUS FOR GENERATING ENCRYPTION STREAM CIPHERS

BACKGROUND OF THE INVENTION

I. Field of the Invention

The present invention relates to encryption. More particularly, the present invention relates to a method and apparatus for generating encryption stream ciphers.

II. Description of the Related Art

Encryption is a process whereby data is manipulated by a random process such that the data is made unintelligible by all but the targeted recipient. One method of encryption for digitized data is through the use of stream ciphers. Stream ciphers work by taking the data to be encrypted and a stream of pseudo-random bits (or encryption bit stream) generated by an encryption algorithm and combining them, usually with the exclusive-or (XOR) operation. Decryption is simply the process of generating the same encryption bit stream and removing the encryption bit stream with the corresponding operation from the encrypted data. If the XOR operation was performed at the encryption side, the same XOR operation is also performed at the decryption side. For a secured encryption, the encryption bit stream must be computationally difficult to predict.

Many of the techniques used for generating the stream of pseudo-random numbers are based on linear feedback shift register (LFSR) over the Galois finite field of order 2. This is a special case of the Galois Finite field of order 2^n where n is a positive integer. For $n = 1$, the elements of the Galois field comprise bit values zero and one. The register is updated by shifting the bits over by one bit position and calculating a new output bit. The new bit is shifted into the register. For a Fibonacci register, the output bit is a linear function of the bits in the register. For a Galois register, many bits are updated in accordance with the output bit just shifted out from the register. Mathematically, the Fibonacci and Galois register architectures are equivalent.

The operations involved in generating the stream of pseudo-random numbers, namely the shifting and bit extraction, are efficient in hardware but inefficient in software or other implementations employing a general purpose processor or microprocessor. The inefficiency increases as the length of the shift register exceeds the length of the registers in the processor used to generate the stream. In addition, for n = 0, only one output bit is generated for each set of operations which, again, results in a very inefficient use of the processor.

An exemplary application which utilizes stream ciphers is wireless telephony. An exemplary wireless telephony communication system is a code division multiple access (CDMA) system. The operation of CDMA system is disclosed in U.S. Patent No. 4,901,307, entitled "SPREAD SPECTRUM MULTIPLE ACCESS COMMUNICATION SYSTEM USING SATELLITE OR TERRESTRIAL REPEATERS," assigned to the assignee of the present invention, and incorporated by reference herein. The CDMA system is further disclosed in U.S. Patent No. 5,103,459, entitled SYSTEM AND METHOD FOR GENERATING SIGNAL WAVEFORMS IN A CDMA CELLULAR TELEPHONE SYSTEM, assigned to the assignee of the present invention, and incorporated by reference herein. Another CDMA system includes the GLOBALSTAR communication system for world wide communication utilizing low earth orbiting satellites. Other wireless telephony systems include time division multiple access (TDMA) systems and frequency division multiple access (FDMA) systems. The CDMA systems can be designed to conform to the "TIA/EIA/IS-95 Mobile Station-Base Station Compatibility Standard for Dual-Mode Wideband Spread Spectrum Cellular System", hereinafter referred to as the IS-95 standard. Similarly, the TDMA systems can be designed to conform to the TIA/EIA/IS-54 (TDMA) standard or to the European Global System for Mobile Communication (GSM) standard.

Encryption of digitized voice data in wireless telephony has been hampered by the lack of computational power in the remote station. This has led to weak encryption processes such as the Voice Privacy Mask used in the TDMA standard or to hardware generated stream ciphers such as the A5 cipher used in the GSM standard. The disadvantages of hardware based stream ciphers are the additional manufacturing cost of the hardware and the longer time and larger cost involved in the event the encryption process needs to be changed. Since many remote stations in wireless telephony systems and digital telephones comprise a microprocessor and memory, a stream cipher which is fast and uses little memory is well suited for these applications.

SUMMARY OF THE INVENTION

The present invention is a novel and improved method and apparatus for generating encryption stream ciphers. In accordance with the present invention, the recurrence relation is designed to operate over finite fields larger than GF(2). The linear feedback shift register used to implement the recurrence relation can be implemented using a circular buffer or sliding a window. In the exemplary embodiment, multiplications of the elements of the finite field are implemented using lookup tables. A non-linear output can be obtained by using one or a combination of non-linear processes. The stream ciphers can be designed to support multi-tier keying to suit the requirements of the applications for which the stream ciphers are used.

It is an object of the present invention to generate encryption stream ciphers using architectures which are simple to implement in a processor. In particular, more efficient implementations can be achieved by selecting a finite field which is more suited for the processor. The elements and coefficients of the recurrence relation can be selected to match the byte or word size of the processor. This allows for efficient manipulation of the elements by the processor. In the exemplary embodiment, the finite field selected is the Galois field with 256 elements ($GF(2^8)$). This results in elements and coefficients of the recurrence relation occupying one byte of memory which can be efficiently manipulated. In addition, the use of a larger finite field reduces the order of the recurrence relation. For a finite field $GF(2^n)$, the order k of the recurrence relation which encodes the same amount of states is reduced by a factor of n (or a factor of 8 for the exemplary $GF(2^8)$).

It is another object of the present invention to implement field multiplications using lookup tables. In the exemplary embodiment, a multiplication (of non-zero elements) in the field can be performed by taking the logarithm of each of the two operands, adding the logarithmic values, and exponentiating the combined logarithmic value. The logarithmic and exponential tables can be created using an irreducible polynomial. In the exemplary embodiment, the tables are pre-computed and stored in memory. Similarly, a field multiplication with a constant coefficient can be performed using a simple lookup table. Again, the table can be pre-computed using the irreducible polynomial and stored in memory.

It is yet another object of the present invention to remove linearity in the output of a linear feedback shift register by the use of one or a combination of the following processes: irregular stuttering (sometimes referred to as decimation), non-linear function, multiple shift registers and combining outputs from the registers, variable feedback polynomial on one register, and other non-linear processes. In the exemplary embodiment, the non-linear output can be use to randomly control the stuttering of the shift register. Additionally, a non-linear output can be derived by performing a non-linear operation on selected elements of the shift register. Furthermore, the output from the non-linear function can be XORed with a set of constants such that the non-linear output bits are unpredictably inverted.

It is yet another object of the present invention to implement the linear feedback shift register using a circular buffer or a sliding window. With the circular buffer or sliding window implementation, the elements are not shifted within the buffer. Instead, a pointer or index is used to indicate the location of the most recently computed element. The pointer is moved as new elements are computed and shifted into the circular buffer or sliding window. The pointer wraps around when it reaches an edge.

It is yet another object of the present invention to provide stream ciphers having multi-tier keying capability. In the exemplary embodiment, the state of the shift register is first initialized with a secret key. For some communication system wherein data are transmitted over frames, a stream cipher can be generated for each frame such that erased or out of sequence frames do not disrupt the operation of the encryption process. A second tier keying process can be initialized for each frame using a frame key initialization process.

BRIEF DESCRIPTION OF THE DRAWINGS

The features, objects, and advantages of the present invention will become more apparent from the detailed description set forth below when taken in conjunction with the drawings in which like reference characters identify correspondingly throughout and wherein:

FIG. 1 is a block diagram of an exemplary implementation of a recurrence relation;

FIG. 2 is a exemplary block diagram of an stream cipher generator utilizing a processor;

FIG. 3A and 3B are diagrams showing the contents of a circular buffer at time n and time n+1, respectively;

FIG. 3C is a diagram showing the content of a sliding window;

FIG. 4 is a block diagram of an exemplary stream cipher generator of the present invention;

FIG. 5 is a flow diagram of an exemplary secret key initialization process of the present invention; and

FIG. 6 is a flow diagram of an exemplary per frame initialization process of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Linear feedback shift register (LFSR) is based on a recurrence relation over the Galois field, where the output sequence is defined by the following recurrence relation :

$$S_{n+k} = C_{k-1}S_{n+k-1} + C_{k-2}S_{n+k-2} + \cdots + C_1S_{n+1} + C_0S_n \,, \qquad (1)$$

where S_{n+k} is the output element, C_j are constant coefficients, k is the order of the recurrence relation, and n is an index in time. The state variables S and coefficients C are elements of the underlying finite field. Equation (1) is sometimes expressed with a constant term which is ignored in this specification.

A block diagram of an exemplary implementation of the recurrence relation in equation (1) is illustrated in FIG. 1. For a recurrence relation of order k, register **12** comprises k elements S_n to S_{n+k-1}. The elements are provided to Galois field multipliers **14** which multiplies the elements with the constants C_j. The resultant products from multipliers **14** are provided Galois field adders **16** which sum the products to provide the output element.

For n = 1, the elements of GF(2) comprise a single bit (having a value of 0 or 1) and implementation of equation (1) requires many bit-wise operations. In this case, the implementation of the recurrence relation using a general purpose processor is inefficient because a processor which is designed to manipulate byte or word sized objects is utilized to perform many operations on single bits.

In the present invention, the linear feedback shift register is designed to operate over finite fields larger than GF(2). In particular, more efficient implementations can be achieved by selecting a finite field which is more suited for a processor. In the exemplary embodiment, the finite field selected is the Galois field with 256 elements (GF(2^8)) or other Galois fields with 2^n elements, where n is the word size of the processor.

In the preferred embodiment, a Galois field with 256 elements (GF(2^8)) is utilized. This results in each element and coefficient of the recurrence relation occupying one byte of memory. Byte manipulations can be performed efficiently by the processor. In addition, the order k of the recurrence relation which encodes the same amount of states is reduced by a factor of n, or 8 for GF(2^8).

In the present invention, a maximal length recurrence relation is utilized for optimal results. Maximal length refers to the length of the output sequence (or the number of states of the register) before repeating. For a recurrence relation of order k, the maximal length is $N^k - 1$, where N is the number of elements in the underlying finite field, and N = 256 in the preferred embodiment. The state of all zeros is not allowed.

An exemplary block diagram of a stream cipher generator utilizing a processor is shown in FIG. 2. Controller **20** connects to processor **22** and comprises the set of instructions which directs the operation of processor **22**. Thus, controller **20** can comprise a software program or a set of microcodes. Processor **22** is the hardware which performs the manipulation required by the generator. Processor **22** can be implemented as a microcontroller, a microprocessor, or a digital signal processor designed to performed the functions described herein. Memory element **24** connects to processor **22** and is used to implement the linear feedback shift register and to store pre-computed tables and instructions which are described below. Memory element **24** can be implemented with random-access-memory or other memory devices designed to perform the functions described herein. The instructions and tables can be stored in read-only memory, only the memory for the register itself needs to be modified during the execution of the algorithm.

I. Generating Non-Linear Output Stream

The use of linear feedback shift register for stream ciphers can be difficult to implement properly. This is because any linearity remaining in the output stream can be exploited to derive the state of the register at a point in time. The register can then be driven forward or backward as desired to recover the output stream. A number of techniques can be used to generate non-linear stream ciphers using linear feedback shift register. In the exemplary embodiment, these non-linear techniques comprise stuttering (or unpredictable decimation) of the register, the use of a non-linear function on the state of the register, the use of multiple registers and non-linear combination of the outputs of the registers, the use of variable feedback polynomials on one register, and other non-linear processes. These techniques are each described below. Some of the techniques are illustrated by the example below. Other techniques to generate non-linear stream ciphers can be utilized and are within the scope of the present invention.

Stuttering is the process whereby the register is clocked at a variable and unpredictable manner. Stuttering is simple to implement and provides good results. With stuttering, the output associated with some states of the register are not provided at the stream cipher, thus making is more difficult to reconstruct the state of the register from the stream cipher.

Using a non-linear function on the state of the shift register can also provide good results. For a recurrence relation, the output element is generated from a linear function of the state of the register and the coefficients, as defined by equation (1). To provide non-linearity, the output element can be generated from a non-linear function of the state of the register. In particular, non-linear functions which operate on byte or word sized data on general purpose processors can be utilized.

Using multiple shift registers and combining the outputs from the registers in a non-linear fashion can provide good results. Multiple shift registers can be easily implemented in hardware where additional cost is minimal and operating the shift registers in parallel to maintain the same operating speed is possible. For implementations on a general purpose processor, a single larger shift register which implements the multiple shift registers can be utilized since the larger shift register can be updated in a constant time (without reducing the overall speed).

Using a variable feedback polynomial which changes in an unpredictable manner on one register can also provide good results. Different polynomials can be interchanged in a random order or the polynomial can be altered in a random manner. The implementation of this technique is can be simple if properly designed.

II. Operations on Elements of Larger Order Finite Fields

The Galois field $GF(2^8)$ comprises 256 elements. The elements of Galois field $GF(2^8)$ can be represented in one of several different ways. A common and standard representation is to form the field from the coefficients modulo 2 of all polynomials with degree less than 8. That is, the element α of the field can be represented by a byte with bits $(a_7, a_6, ..., a_0)$ which represent the polynomial :

$$a_7x^7 + a_6x^6 + ... + a_1x + a_0 \ . \tag{2}$$

The bits are also referred to as the coefficients of the polynomial. The addition operation on two polynomials represented by equation (2) can be performed by addition modulo two for each of the corresponding coefficients $(a_7, a_6, ..., a_0)$. Stated differently, the addition operation on two bytes can be achieved by performing the exclusive-OR on the two bytes. The additive identity is the polynomial with all zero coefficients (0, 0, ..., 0).

Multiplication in the field can be performed by normal polynomial multiplication with modulo two coefficients. However, multiplication of two polynomials of order n produces a resultant polynomial of order (2n-1) which needs to be reduced to a polynomial of order n. In the exemplary embodiment, the reduction is achieved by dividing the resultant polynomial by an irreducible polynomial, discarding the quotient, and retaining the remainder as the reduced polynomial. The selection of the irreducible polynomial alters the mapping of the elements of the group into encoded bytes in memory, but does not otherwise affect the actual group operation. In the exemplary embodiment, the irreducible polynomial of degree 8 is selected to be :

$$x^8 + x^6 + x^3 + x^2 + 1 \ . \tag{3}$$

Other irreducible monic polynomial of degree 8 can also be used and are within the scope of the present invention. The multiplicative identity element is $(a_7, a_6, ..., a_0) = (0, 0, ..., 1)$.

Polynomial multiplication and the subsequent reduction are complicated operations on a general purpose processor. However, for Galois fields having a moderate number of elements, these operations can be performed by lookup tables and more simple operations. In the exemplary performed by lookup tables and more simple operations. In the exemplary embodiment, a multiplication (of non-zero elements) in the field can be performed by taking the logarithm of each of the two operands, adding the logarithmic values modulo 255, and exponentiating the combined logarithmic value. The reduction can be incorporated within the lookup tables.

The exponential and logarithm tables can be generated as follows. First, a generator g of the multiplicative subgroup GF(2^8) is determined. In this case, the byte value g=2 (representing the polynomial x) is a generator. The exponential table, shown in Table 1, is a 256-byte table of the values g^i, for i = 0, 1, ... 2^8-1. For g^i (considered as an integer) of less than 256, the value of the exponential is as expected as evidenced by the first eight entries in the first row of Table 1. Since g=2, each entry in the table is twice the value of the entry to the immediate left (taking into account the fact that Table 1 wraps to the next row). However, for each g^i greater than 255, the exponential is reduced by the irreducible polynomial shown in equation (3). For example, the exponential x^8 (first row, ninth column) is reduced by the irreducible polynomial $x^8+x^6+x^3+x^2+1$ to produce the remainder $-x^6-x^3-x^2-1$. This remainder is equivalent to $x^6+x^3+x^2+1$ for modulo two operations and is represented as 77 ($2^6+2^3+2^2+1$) in Table 1. The process is repeated until g^i for all index i = 0 to 255 are computed.

Having defined the exponential table, the logarithm table can be computed as the inverse of the exponential table. In Table 1, there is a unique one to one mapping of the exponential value g^i for each index i which results from using an irreducible polynomial. For Table 1, the mapping is i $\Leftrightarrow$ 2^i, or the value stored in the i-th location is 2^i. Taking $\log_2$ of both sides results in the following : $\log_2(i)$ $\Leftrightarrow$ i. These two mappings indicate that if the content of the i-th location in the exponential table is used as the index of the logarithm table, the log of this index is the index of the exponential table. For example, for i = 254, the exponential value $2^i = 2^{254} = 166$ as shown in the last row, fifth column in Table 1. Taking $\log_2$ of both sides yields $254 = \log_2(166)$. Thus, the entry for the index i = 166 in the logarithmic table is set to 254. The process is repeated until all entries in the logarithmic table have been mapped. The log of 0 is an undefined number. In the exemplary embodiment, a zero is used as a place holder.

Having defined the exponential and logarithmic tables, a multiplication (of non-zero elements) in the field can be performed by looking up the logarithmic of each of the two operands in the logarithmic table, adding the logarithmic values using modulo 255, and exponentiating the combined logarithmic value by looking up the exponential table. Thus, the multiplication operation in the field can be performed with three lookup operations and a truncated addition. In the exemplary Galois field $GF(2^8)$, each table is 255 bytes long and can be pre-computed and stored in memory. In the exemplary embodiment, the logarithm table has an unused entry in position 0 to avoid the need to subtract 1 from the indexes. Note that when either operand is a zero, the corresponding entry in the logarithmic table does not represent a real value. To provide the correct result, each operand needs to be tested to see if it is zero, in which case the result is 0, before performing the multiplication operation as described.

For the generation of the output element from a linear feedback shift register using a recurrence relation, the situation is simpler since the coefficients C_j are constant as shown in equation (1). For efficient implementation, these coefficients are selected to be 0 or 1 whenever possible. Where C_j have values other than 0 or 1, a table can be pre-computed for the multiplication $t_i = C_j \bullet i$, where $i = 0, 1, 2, \ldots, 2^8-1$. In this case, the multiplication operation can be performed with a single table lookup and no tests. Such a table is fixed and can be stored in read-only memory.

Table 1 - Exponential Table

i	xx0	xx1	xx2	xx3	xx4	xx5	xx6	xx7	xx8	xx9
00x	1	2	4	8	16	32	64	128	77	154
01x	121	242	169	31	62	124	248	189	55	110
02x	220	245	167	3	6	12	24	48	96	192
03x	205	215	227	139	91	182	33	66	132	69
04x	138	89	178	41	82	164	5	10	20	40
05x	80	160	13	26	52	104	208	237	151	99
06x	198	193	207	211	235	155	123	246	161	15
07x	30	60	120	240	173	23	46	92	184	61
08x	122	244	165	7	14	28	56	112	224	141
09x	87	174	17	34	68	136	93	186	57	114
10x	228	133	71	142	81	162	9	18	36	72
11x	144	109	218	249	191	51	102	204	213	231
12x	131	75	150	97	194	201	223	243	171	27
13x	54	108	216	253	183	35	70	140	85	170
14x	25	50	100	200	221	247	163	11	22	44
15x	88	176	45	90	180	37	74	148	101	202
16x	217	255	179	43	86	172	21	42	84	168
17x	29	58	116	232	157	119	238	145	111	222
18x	241	175	19	38	76	152	125	250	185	63
19x	126	252	181	39	78	156	117	234	153	127
20x	254	177	47	94	188	53	106	212	229	135
21x	67	134	65	130	73	146	105	210	233	159
22x	115	230	129	79	158	113	226	137	95	190
23x	49	98	196	197	199	195	203	219	251	187
24x	59	118	236	149	103	206	209	239	147	107
25x	214	225	143	83	166					

Table 2 - Logarithmic Table

i	xx0	xx1	xx2	xx3	xx4	xx5	xx6	xx7	xx8	xx9
00x	0	0	1	23	2	46	24	83	3	106
01x	47	147	25	52	84	69	4	92	107	182
02x	48	166	148	75	26	140	53	129	85	170
03x	70	13	5	36	93	135	108	155	183	193
04x	49	43	167	163	149	152	76	202	27	230
05x	141	115	54	205	130	18	86	98	171	240
06x	71	79	14	189	6	212	37	210	94	39
07x	136	102	109	214	156	121	184	8	194	223
08x	50	104	44	253	168	138	164	90	150	41
09x	153	34	77	96	203	228	28	123	231	59
10x	142	158	116	244	55	216	206	249	131	111
11x	19	178	87	225	99	220	172	196	241	175
12x	72	10	80	66	15	186	190	199	7	222
13x	213	120	38	101	211	209	95	227	40	33
14x	137	89	103	252	110	177	215	248	157	243
15x	122	58	185	198	9	65	195	174	224	219
16x	51	68	105	146	45	82	254	22	169	12
17x	139	128	165	74	91	181	151	201	42	162
18x	154	192	35	134	78	188	97	239	204	17
19x	229	114	29	61	124	235	232	233	60	234
20x	143	125	159	236	117	30	245	62	56	246
21x	217	63	207	118	250	31	132	160	112	237
22x	20	144	179	126	88	251	226	32	100	208
23x	221	119	173	218	197	64	242	57	176	247
24x	73	180	11	127	81	21	67	145	16	113
25x	187	238	191	133	200	161				

III. Memory Implementation

When implemented in hardware, shifting bits is a simple and efficient operation. Using a processor and for a shift register larger than the registers of the processor, shifting bits is an iterative procedure which is very inefficient. When the units to be shifted are bytes or words, shifting becomes simpler because there is no carry between bytes. However, the shifting process is still iterative and inefficient.

In the exemplary embodiment, the linear feedback shift register is implemented with circular buffer or sliding window. The diagrams showing the contents of circular buffer **24a** at time n at time n+1 are shown in FIGS. 3A and 3B, respectively. For circular buffer **24a**, each element of the shift register is stored in a corresponding location in memory. A single index, or pointer **30**, maintains the memory location of the most recent element stored in memory, which is S_{k-1} in FIG. 3A. At time n+1, the new element S_k is computed and stored over the oldest element S_0 in memory, as shown in FIG. 3B. Thus, instead of shifting all elements in memory, pointer **30** is moved to the memory location of the new element S_k. When pointer **30** reaches the end of circular buffer **24a**, it is reset to the beginning (as shown in FIGS. 3A and 3B). Thus, circular buffer **24a** acts as if it is a circle and not a straight line.

Circular buffer **24a** can be shifted from left-to-right, or right-to-left as shown in FIGS, 3A and 3B. Correspondingly, pointer **30** can move left-to-right, or right-to-left as shown in FIGS. 3A and 3B. The choice in the direction of the shift is a matter of implementation style and does not affect the output result.

To generate an output element in accordance with a recurrence relation, more than one elements are typically required from memory. The memory location associated with each required element can be indicated by a separate pointer which is updated when the register is shifted. Alternatively, the memory location associated with each required element can be computed from pointer **30** as necessary. Since there is a one-to-one mapping of each element to a memory location, a particular element can be obtained by determining the offset of that element from the newest element (in accordance with the recurrence relation), adding that offset to pointer **30**, and addressing the memory location indicated by the updated pointer. Because of the circular nature of the memory, the calculation of the updated pointer is determined by an addition modulo k of the offset to pointer **30**. Addition modulo k is simple when k is a power of two but is otherwise an inefficient operation on a processor.

In the preferred embodiment, the shift register is implemented with sliding window **24b** as shown in FIG. 3C. Sliding window **24b** is at least twice as long as circular buffer **24a** and comprises two circular buffers **32a** and **32b** arranged adjacent to each other. Each of circular buffers **32a** and **32b** behaves like circular **24a** described above. Circular buffer **32b** is an exact replica of circular buffer **32a**. Thus, each element of the shift register is stored in two corresponding locations in memory, one each for circular buffers **32a** and **32b**. Pointer **34** maintains the memory location of the most recent element stored in circular buffer **32a**, which is S_{k-1} in FIG. 3C. In the exemplary embodiment, pointer **34** starts at the middle of sliding window **24b**, moves right-to-left, and resets to the middle again when it reaches the end on the left side.

From FIG. 3C, it can be observed that no matter where in circular buffer **32a** pointer **34** appears, the previous k-1 elements can be addressed to the right of pointer **34**. Thus, to address an element in the shift register in accordance with the recurrence relation, an offset of k-1 or less is added to pointer **34**. Addition modulo k is not required since the updated pointer is always to the right of pointer **34** and computational efficiency is obtained. For this implementation, sliding window **24b** can be of any length at least twice as long as circular buffer **24a**, with any excess bytes being ignored. Furthermore, the update time is constant and short.

IV. Exemplary Stream Cipher Based on LFSR Over $GF(2^8)$

The present invention can be best illustrated by an exemplary generator for a stream cipher based on a linear feedback shift register over $GF(2^8)$. The stream cipher described below uses the byte operations described above over the Galois field of order 8 with the representation of $\oplus$ and $\otimes$ for operations of addition and multiplication, respectively, over the Galois field. In the exemplary embodiment, table lookup is utilized for the required multiplication with constants C_j. In the exemplary embodiment, a sliding window is used to allow fast updating of the shift register.

A block diagram of the exemplary generator is shown in FIG. 4. In the exemplary embodiment, linear feedback shift register **52** is 17 octets (or 136 bits) long which allows shift register **52** to be in $2^{136} - 1$ (or approximately 8.7×10^{40}) states. The state where the entire register is 0 is not a valid state and does not occur from any other state. The time to update register **52** with a particular number of non-zero elements in the recurrence relation is constant irrespective of the length of register **52**. Thus, additional length for register **52** (for higher order recurrence relation) can be implemented at a nominal cost of extra bytes in memory.

In the exemplary embodiment, linear feedback shift register **52** is updated in accordance with the following recurrence relation :

$$S_{n+17} = (100 \otimes S_{n+9}) \oplus S_{n+4} \oplus (141 \otimes S_n) \;, \tag{4}$$

where the operations are defined over GF(2^8), $\oplus$ is the exclusive-OR operation on two bytes represented by Galois adders **58**, and $\otimes$ is a polynomial modular multiplication represented by Galois multipliers **54** (see FIG. 4). In the exemplary embodiment, the modular multiplications on coefficients **56** are implemented using byte table lookups on pre-computed tables as described above. In the exemplary embodiment, the polynomial modular multiplication table is computed using the irreducible polynomial defined by equation (3). The recurrence relation in equation (4) was chosen to be maximal length, to have few non-zero coefficients, and so that the shift register elements used were distinct from ones used for the non-linear functions below.

In the exemplary embodiment, to disguise the linearity of shift register **52**, two of the techniques described above are used, namely stuttering and using a non-linear function. Additional non-linearity techniques are utilized and are described below.

In the exemplary embodiment, non-linearity is introduced by performing a non-linear operation on multiple elements of shift register **52**. In the exemplary embodiment, four of the elements of shift register **52** are combined using a function which is non-linear. An exemplary non-linear function is the following :

$$V_n = (S_n + S_{n+5}) \times (S_{n+2} + S_{n+12}) \;, \tag{5}$$

where V_n is the non-linear output (or the generator output), + is the addition truncated modulo 256 represented by arithmetic adders **60**, and $\times$ is the multiplication modulo 257 represented by modular multiplier **62** and described below. In the exemplary embodiment, the four bytes used are S_n, S_{n+2}, S_{n+5} and S_{n+12}, where S_n is the oldest calculated element in the sequence according to the recurrence relation in equation (4). These elements are selected such that, as the register shifts, no two elements are used in the computation of two of the generator outputs. The pairwise distances between these elements are distinct values. For example, S_{n+12} is not combined with S_{n+5}, S_{n+2}, nor S_n again as it is shifted through register **52**.

Simple byte addition, with the result truncated modulo 256, is made non-linear in GF(2^8) by the carry between bits. In the exemplary embodiment, two pairs of elements in the register {(S_n and S_{n+5}) and (S_{n+2} and S_{n+12})} are combined using addition modulo 256 to yield two intermediate results. However, addition modulo 256 is not ideal since the least significant bits have no carry input and are still combined linearly.

Another non-linear function which can be computed conveniently on a processor is multiplication. However, truncation of a normal multiplication into a single byte may not yield good result because multiplication modulo 256 does not form a group since the results are not well distributed within the field. A multiplicative group of the field of integers modulo the prime number 257 can be used. This group consists of integers in the range of 1 to 256 with the group operation being integer multiplication reduced modulo 257. Note that the value 0 does not appear in the group but the value 256 does. In the exemplary embodiment, the value of 256 can be represented by a byte value of 0.

Typically, processors can perform multiplication instructions efficiently but many have no capability to perform, nor to perform efficiently, divide or modulus instructions. Thus, the modulo reduction by 257 can represent a performance bottleneck. However, reduction modulo 257 can be computed using other computational modulo 2^n, which in the case of n=8 are efficient on common processors. It can be shown that for a value X in the range of 1 to 2^{16} - 1 (where X is the result of a multiplication of two 8th order operands), reduction modulo 257 can be computed as :

$$X_{257} = \left\{ X_{256} - \frac{X}{256} \right\}_{257}, \qquad (6)$$

where X_{257} is the reduction modulo 257 of X and X_{256} is the reduction modulo 256 of X. Equation (6) indicates that reduction modulo 257 of a 16-bit number can be obtained by subtracting the 8 most significant bits (X/256) from the 8 least significant bits (X_{256}). The result of the subtraction is in the range of -255 and 255 and may be negative. If the result is negative, it can be adjusted to the correct range by adding 257. In the alternative embodiment, reduction modulo 257 can be performed with a lookup table comprising 65,536 elements, each 8 bits wide.

Multiplication of the two intermediate results is one of many non-linear functions which can be utilized. Other non-linear functions, such as bent functions or permuting byte values before combining them, can also be implemented using lookup tables. The present invention is directed at the use of these various non-linear functions for producing non-linear output.

implemented using lookup tables. The present invention is directed at the use of these various non-linear functions for producing non-linear output.

In the exemplary embodiment, stuttering is also utilized to inject additional non-linearity. The non-linear output derived from the state of the linear feedback shift register as described above may be used to reconstruct the state of the shift register. This reconstruction can be made more difficult by not representing some of the states at the output of the generator, and choosing which in an unpredictable manner. In the exemplary embodiment, the non-linear output is used to determine what subsequent bytes of non-linear output appear in the output stream. When the generator is started, the first output byte is used as the stutter control byte. In the exemplary embodiment, each stutter control byte is divided into four pairs of bits, with the least significant pair being used first. When all four pairs have been used, the next non-linear output byte from the generator is used as the next stutter control byte, and so on.

Each pair of stutter control bits can take on one of four values. In the exemplary embodiment, the action performed for each pair value is tabulated in Table 3.

Table 3

Pair Value	Action of Generator
(0, 0)	Register is cycled but no output is produced
(0, 1)	Register is cycled and the non-linear output XOR with the constant $(0\,1\,1\,0\,1\,0\,0\,1)_2$ becomes the output of the generator. Register is cycled again.
(1, 0)	Register is cycled twice and the non-linear output becomes the output of the generator.
(1, 1)	Register is cycled and the non-linear output XOR with the constant $(1\,1\,0\,0\,0\,1\,0\,1)_2$ becomes the output of the generator.

As shown in Table 3, in the exemplary embodiment, when the pair value is (0, 0), the register is cycled once but no output is produced. Cycling of the register denotes the calculation of the next sequence output in accordance with equation (4) and the shifting this new element into the register. The next stutter control pair is then used to determine the action to be taken next.

In the exemplary embodiment, when the pair value is (0, 1) the register is cycled and the non-linear output generated in accordance with equation (5). The non-linear output is XORed with the constant $(0\,1\,1\,0\,1\,0\,0\,1)_2$ and the result is provided as the generator output. The register is then cycled again. In FIG. 4, the XORed function is performed by XOR gate **66** and the constant is selected by multiplexer (MUX) **64** using the stutter control pair from buffer **70**. The output from XOR gate **66** is provided to switch **68** which provides the generator output and the output byte for stutter control in accordance with the value of the stutter control pair. The output byte for stutter control is provided to buffer **70**.

In the exemplary embodiment, when the pair value is (1, 0) the register is cycled twice and the non-linear output generated in accordance with equation (5) is provided as the generator output.

In the exemplary embodiment, when the pair value is (1, 1) the register is cycled and the non-linear output generated in accordance with equation (5). The non-linear output is then XORed with the constant $(1\,1\,0\,0\,0\,1\,0\,1)_2$ and the result is provided as the generator output.

In the exemplary embodiment, the constants which are used in the above steps are selected such that when a generator output is produced, half of the bits in the output are inverted with respect to the outputs produced by the other stutter control pairs. For stutter control pair (1, 0), the non-linear output can be viewed as being XORed with the constant $(0\,0\,0\,0\,0\,0\,0\,0)_2$. Thus, the Hamming distance between any of the three constants is four. The bit inversion further masks the linearity of the generator and frustrates any attempt to reconstruct the state based on the generator output.

The present invention supports a multi-tier keying structure. A stream cipher which supports multi-tier keying structure is especially useful for wireless communication system wherein data are transmitted in frames which may be received in error or out-of-sequence. An exemplary two-tier keying structure is described below.

In the exemplary embodiment, one secret key is used to initialized the generator. The secret key is used to cause the generator to take an unpredictable leap in the sequence. In the exemplary embodiment, the secret key has a length of four to k-1 bytes (or 32 to 128 bits for the exemplary recurrence relation of order 17). Secret keys of less than 4 bytes are not preferred because the initial randomization may not be adequate. Secret keys of greater than k-1 bytes can also be utilized but are redundant, and care should be taken so that a value for the key does not cause the register state to be set to all 0, a state which cannot happen with the current limitation.

embodiment, at block **112**, the state of the shift register is first initialized with the Fibonacci numbers modulo 256. Thus, elements S_0, S_1, S_2, S_3, S_4, S_5, and so on, are initialized with 1, 1, 2, 3, 5, 8, and so on, respectively. Although Fibonacci numbers are used, any set of non-zero numbers which are not linearly related in the Galois field can be used to initialized the register. These number should not have exploitable linear relationship which can be used to reconstruct the state of the register.

Next, the loop index n is set to zero, at block **114**. The secret key initialization process then enters a loop. In the first step within the loop, at block **116**, the first unused byte of the key material is added to S_n. Addition of the key material causes the generator to take an unpredictable leap in the sequence. The key is then shifted by one byte, at block **118**, such that byte used in block **116** is deleted. The register is then cycled, at block **120**. The combination of blocks **116** and **120** effectively performs the following calculation :

$$S_{n+17} = \left(100 \otimes S_{n+9}\right) \oplus S_{n+4} \oplus \left(141 \otimes \left(S_n \oplus K\right)\right) , \qquad (7)$$

where K is the first unused byte of the key material. The loop index n is incremented, at block **122**. A determination is then made whether all key material have been used, at block **124**. If the answer is no, the process returns to block **116**. Otherwise, the process continues to block **126**.

In the exemplary embodiment, the length of the key is added to S_n, at block **126**. Addition of the length of the key causes the generator to take an additional leap in the sequence. The process then enters a second loop. In the first step within the second loop, at block **128**, the register is cycled The loop index n is incremented, at block **130**, and compared against the order k of the generator, at block **132**. If n is not equal to k, the process returns to block **128**. Otherwise, if n is equal to k, the process continues to block **134** where the state of the generator are saved. The process then terminates at block **136**.

In addition to the secret key, a secondary key can also be used in the present invention. The secondary key is not considered secret but is used in an exemplary wireless telephony system to generate a unique cipher for each frame of data. This ensures that erased or out-of-sequence frames do not disrupt the flow of information. In the exemplary embodiment, the stream cipher accepts a per-frame key, called a frame key, in the form of a 4-octet unsigned integer. The per-frame initialization is similar to the secret key cipher accepts a per-frame key, called a frame key, in the form of a 4-octet unsigned integer. The per-frame initialization is similar to the secret key initialization above but is performed for each frame of data. If the use of the stream cipher is such that it is unnecessary to utilize per-frame key information, for example for file transfer over a reliable link, the per-frame initialization process can be omitted.

A flow diagram of an exemplary per-frame initialization process with the frame key is shown in FIG. 6. The process starts at block **210**. In the exemplary embodiment, at block **212**, the state of the generator is initialized with the state saved from the secret key initialization process as described above. Next, the loop index n is set to zero, at block **214**. The per-frame initialization process then enters a loop. In the first step within the loop, at block **216**, the least significant byte of the frame key is added modulo 256 to S_n. The frame key is then shifted by three bits, at block **218**, such that the three least significant bits used in block **216** are deleted. The register is then cycled, at block **220**. In the exemplary embodiment, the loop index n is incremented at block **222** and compared against 11 at block **224**. The value of 11, as used in block **224**, corresponds to the 32 bits used as the frame key and the fact that the frame key is shifted three bits at a time. Different selections of the frame key and different numbers of bits shifted at a time can result in different comparison values used in block **224**. If n is not equal to 11, the process returns to block **216**. Otherwise, if n is equal to 11, the process continues to block **226** and the register is cycled again. The loop index n is incremented, at block **228**, and compared against 2k, at block **230**. If n is not equal to 2k, the process returns to block **226**. Otherwise, if n is equal to 2k, the process terminates at block **232**.

The present invention has been described for the exemplary Galois finite field having 256 elements. Different finite fields can also be utilized such that the size of the elements matches the byte or word size of the processor used to manipulate the elements and/or the memory used to implement the shift register, or having other advantages. Thus, various finite fields having more than two elements can by utilized and are within the scope of the present invention.

The example shown above utilizes a variety of non-linear processes to mask the linearity of the recurrence relation. Other generators can be design utilizing different non-linear processes, or different combinations of the above described non-linear processes and other non-linear processes. Thus, the use of various non-linear processes to generate non-linear outputs can be contemplated and is within the scope of the present invention.

The example shown above utilizes a recurrence relation having an order of 17 and defined by equation (4). Recurrence relation having other orders can also be generated and are within the scope of the present invention. Furthermore, for a given order, various recurrence relations can be generated and are within the scope of the present invention. In the present invention, a maximal length recurrence relation is preferred for optimal results.

The previous description of the preferred embodiments is provided to enable any person skilled in the art to make or use the present invention. The various modifications to these embodiments will be readily apparent to those skilled in the art, and the generic principles defined herein may be applied to other embodiments without the use of the inventive faculty. Thus, the present invention is not intended to be limited to the embodiments shown herein but is to be accorded the widest scope consistent with the principles and novel features disclosed herein.

I CLAIM:

CLAIMS

1. A method for generating a stream cipher comprising :
selecting a finite field having an order greater than one;
selecting a recurrence relation over said finite field;
selecting at least one non-linear process; and
computing said stream cipher in accordance with said recurrence relation and said non-linear process.

2. The method of claim 1 wherein said finite field is selected based on a word size of a processor used to compute said stream cipher.

3. The method of claim 1 wherein said finite field is a Galois field comprising 256 elements.

4. The method of claim 1 wherein said recurrence relation is maximal length.

5. The method of claim 1 wherein said recurrence relation has an order of 17.

6. The method of claim 1 wherein said recurrence relation is implemented with a linear feedback shift register.

7. The method of claim 6 wherein said linear feedback shift register is implemented with a circular buffer.

8. The method of claim 6 wherein said linear feedback shift register is implemented with a sliding window.

9. The method of claim 1 wherein said at least one non-linear process comprises stuttering a shift register used to implement said recurrence relation.

10. The method of claim 9 wherein said stuttering is performed in accordance with said stream cipher.

11. The method of claim 1 wherein said at least one non-linear process comprises computing a non-linear function of a state of said generator.

12. The method of claim 1 wherein said at least one non-linear process comprises bit inversion of selected bits of said stream cipher.

13. The method of claim 12 wherein said bit inversion is performed in accordance with a set of constants having a Hamming distance chosen to invert half of the bits in said stream cipher.

14. The method of claim 1 wherein said computing step comprises field multiplication and modulo addition.

15. The method of claim 14 wherein a result of said field multiplication is reduced by a modulus of a prime number.

16. The method of claim 15 wherein said prime number is 257.

17. The method of claim 14 wherein said field multiplication is performed with lookup tables.

18. The method of claim 17 wherein said lookup tables are pre-computed and stored in a memory element.

19. The method of claim 14 wherein said field multiplication is performed by :

looking up a table of the logarithmic value of each of two operands;

modulo adding logarithmic values of said two operands to obtain a combined logarithmic value; and

looking up a table of exponential value of said combined logarithmic value.

20. The method of claim 1 further comprising the step of :

initializing said generator with a secret key.

21. The method of claim 20 wherein said initializing step comprises the steps of :

adding a least significant byte of said secret key to said recurrence relation;

shifting said secret key by one byte; and

repeating said adding step and said shifting steps until all bytes in said secret key are added to said recurrence relation.

22. The method of claim 20 wherein a length of said secret key is less than an order of said recurrence relation.

23. The method of claim 20 further comprising the step of :

initializing said generator with a per frame key.

24. The method of claim 23 wherein said initializing said generator with a per frame key step comprises the steps of :

adding a least significant byte of said per frame key to said recurrence relation;

shifting said per frame key by three bits;

repeating said adding step and said shifting steps until all bytes in said per frame key are added to said recurrence relation.

25. The method of claim 23 wherein a length of said per frame key is four octets long.

26. The method of claim 23 wherein said initializing said generator with a per frame key step is performed for each data frame.

27. An apparatus for generating a stream cipher comprising :

a processor for receiving instructions for performing a recurrence relation and at least one non-linear process, said processor performing manipulations on elements in accordance with said instructions.

28. The apparatus of claim 27 wherein said recurrence relation is defined over a finite field having an order of greater than one.

29. The apparatus of claim 28 wherein said finite field is selected based on a word size of said processor.

30. The apparatus of claim 28 wherein said finite field is a Galois field comprising 256 elements.

31. The apparatus of claim 27 wherein said recurrence relation is maximal length.

32. The apparatus of claim 27 wherein said recurrence relation has an order of 17.

33. The apparatus of claim 27 wherein said recurrence relation is implemented with a linear feedback shift register.

34. The apparatus of claim 33 wherein said linear feedback shift register is implemented with a circular buffer.

35. The apparatus of claim 33 wherein said linear feedback shift register is implemented with a sliding window.

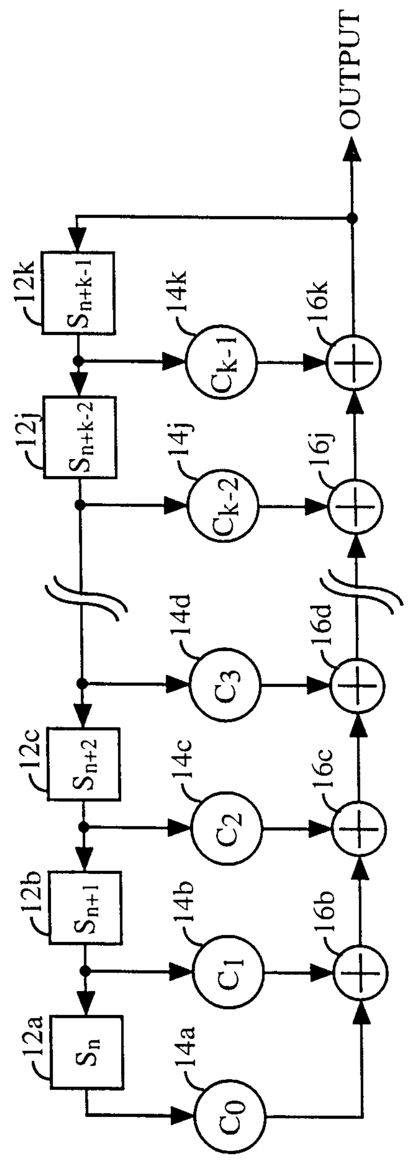

OUTPUT
12a
12b
12c
12j
12k
S_n
S_{n+1}
S_{n+2}
S_{n+k-2}
S_{n+k-1}
14a
14b
14c
14d
14j
14k
C_0
C_1
C_2
C_3
C_{k-2}
C_{k-1}
16b
16c
16d
16j
16k

FIG. 1

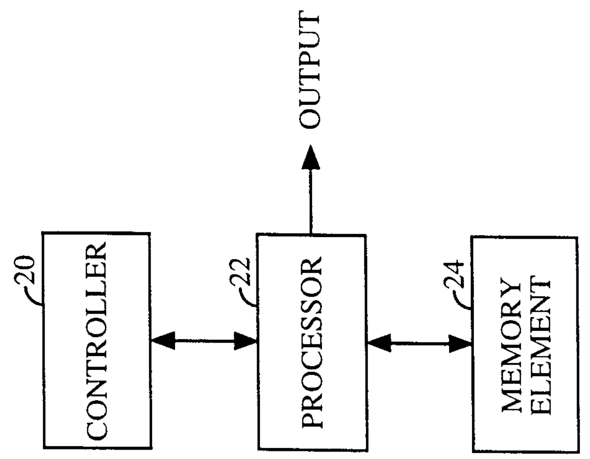


FIG. 2

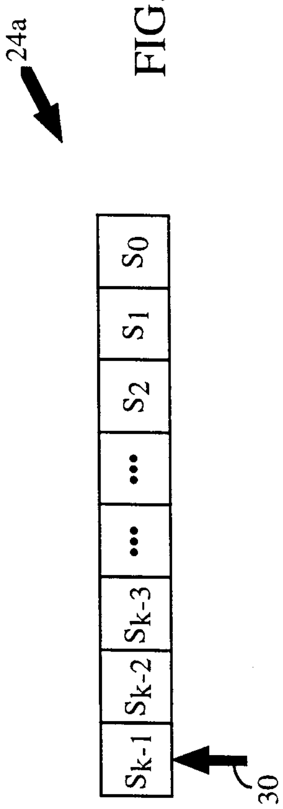


FIG. 3A

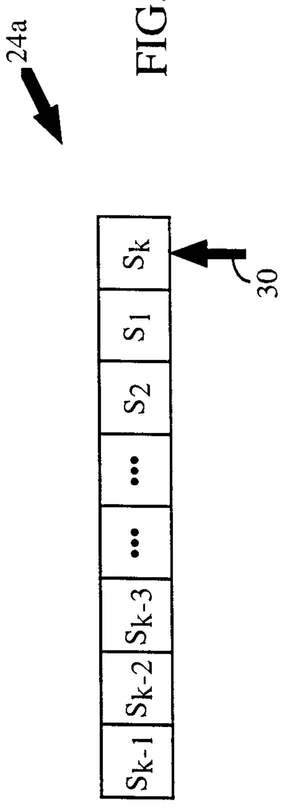


FIG. 3B

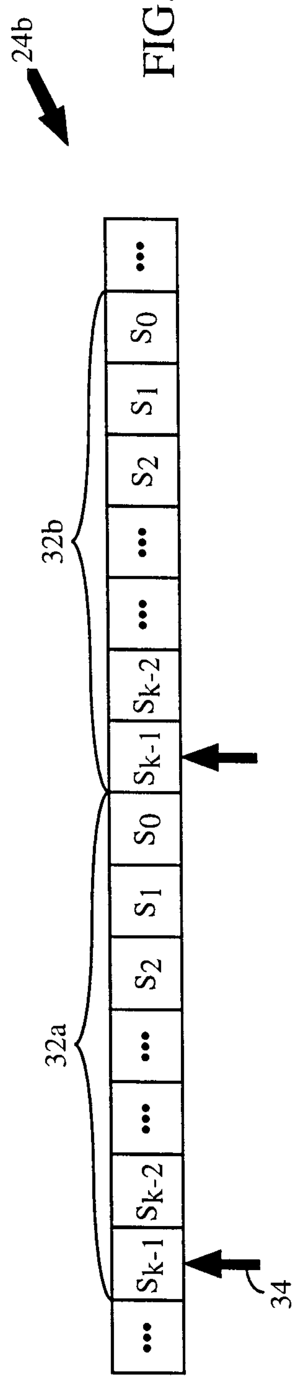


FIG. 3C

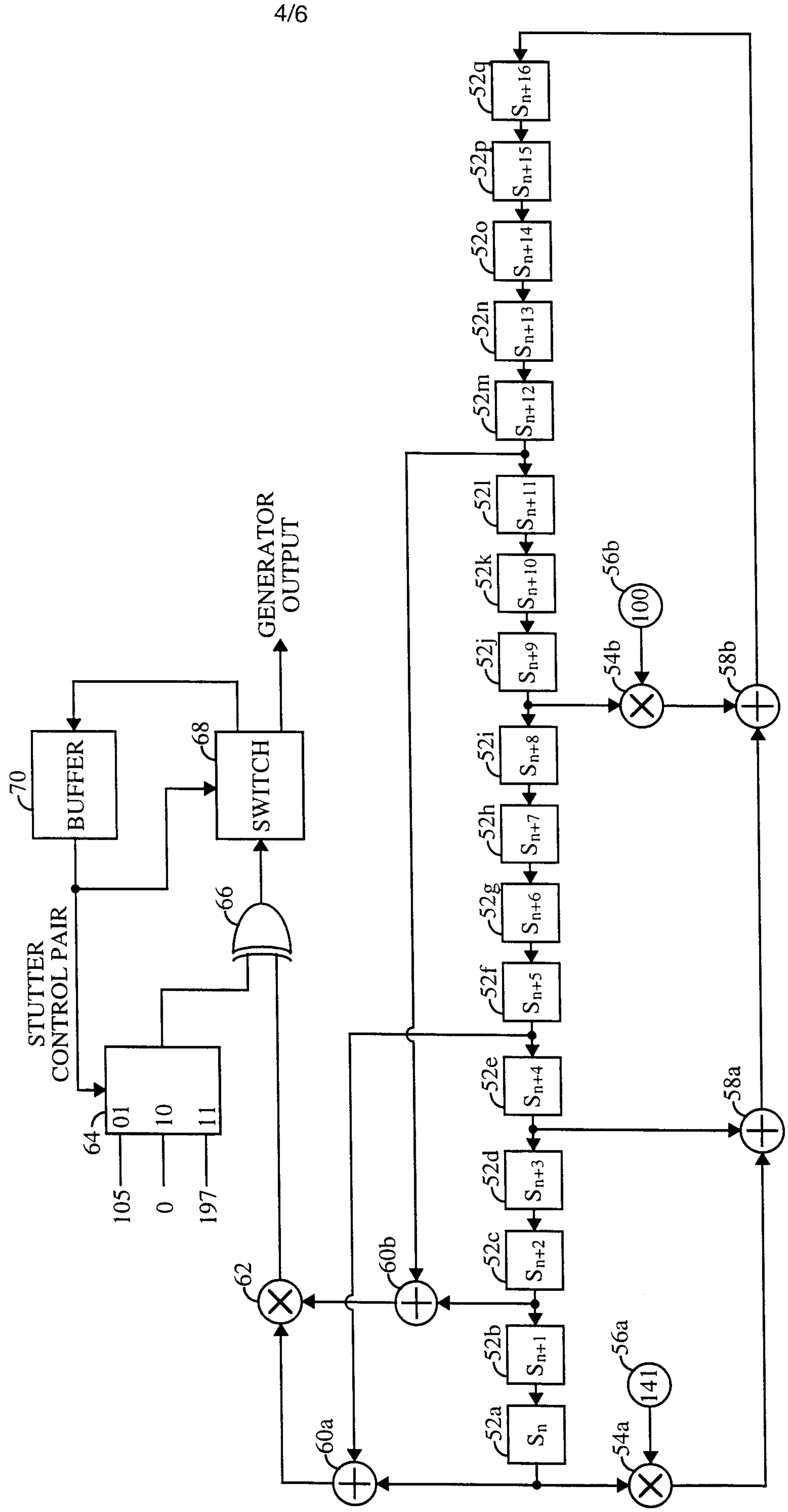

GENERATOR OUTPUT
BUFFER
70
SWITCH
68
66
STUTTER CONTROL PAIR
64
01
10
11
105
0
197
62
60a
60b
52a
S_n
52b
S_{n+1}
52c
S_{n+2}
52d
S_{n+3}
52e
S_{n+4}
52f
S_{n+5}
52g
S_{n+6}
52h
S_{n+7}
52i
S_{n+8}
52j
S_{n+9}
52k
S_{n+10}
52l
S_{n+11}
52m
S_{n+12}
52n
S_{n+13}
52o
S_{n+14}
52p
S_{n+15}
52q
S_{n+16}
54a
56a
141
54b
56b
100
58a
58b

FIG. 4

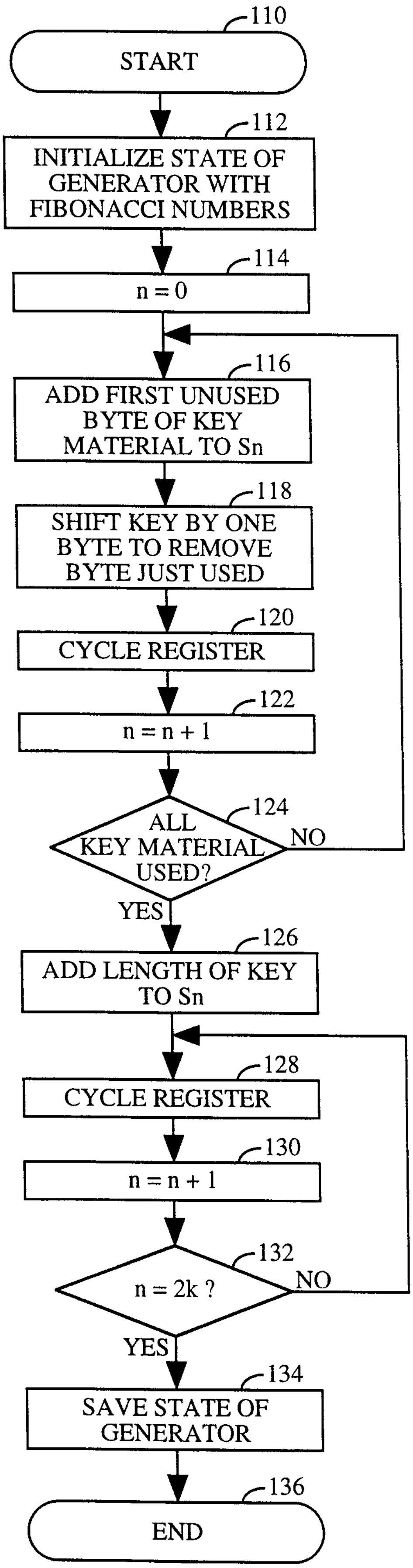

110
START
112
INITIALIZE STATE OF GENERATOR WITH FIBONACCI NUMBERS
114
n = 0
116
ADD FIRST UNUSED BYTE OF KEY MATERIAL TO Sn
118
SHIFT KEY BY ONE BYTE TO REMOVE BYTE JUST USED
120
CYCLE REGISTER
122
n = n + 1
124
ALL KEY MATERIAL USED?
NO
YES
126
ADD LENGTH OF KEY TO Sn
128
CYCLE REGISTER
130
n = n + 1
132
n = 2k ?
NO
YES
134
SAVE STATE OF GENERATOR
136
END

FIG. 5

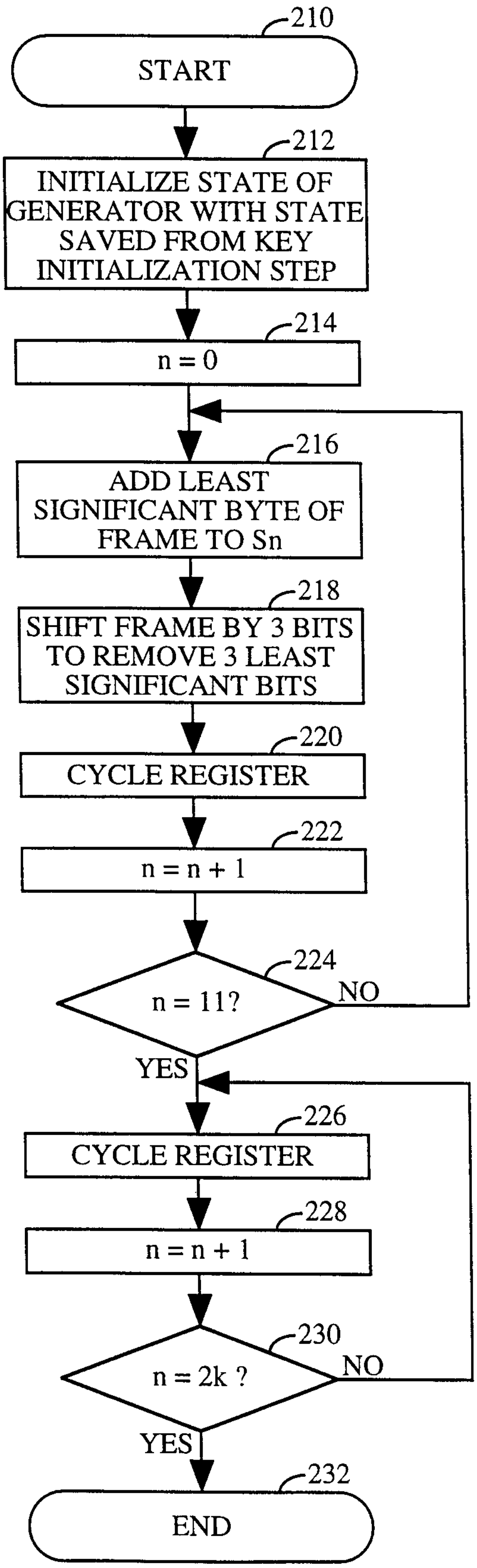

210
START
212
INITIALIZE STATE OF GENERATOR WITH STATE SAVED FROM KEY INITIALIZATION STEP
214
n = 0
216
ADD LEAST SIGNIFICANT BYTE OF FRAME TO Sn
218
SHIFT FRAME BY 3 BITS TO REMOVE 3 LEAST SIGNIFICANT BITS
220
CYCLE REGISTER
222
n = n + 1
224
n = 11?
NO
YES
226
CYCLE REGISTER
228
n = n + 1
230
n = 2k ?
NO
YES
232
END

FIG. 6