TR45.3

Appendix A to IS-54

Rev. B

Dual-Mode Cellular System

Authentication,
Message Encryption,
Voice Privacy Mask Generation,
Shared Secret Data Generation,
A-Key Verification, and Test Data

February 1992

Table of Contents

1. Introduction

This document describes several cryptographic functions for use in new cellular systems that are under development. The first function, called "CAVE", performs an authentication algorithm by combining a random challenge (RAND) from the cellular switch with information from the subscriber's equipment. If the result that is calculated by the subscriber matches the result produced by the switch, then the subscriber will be considered to be authentic.

CAVE is also used to generate a set of cryptovariables for the Cellular Message Encryption Algorithm (CMEA) message encryption process. See §2.4.1.

A further application of CAVE is the generation of 520 bits for the duplex voice privacy masks. This is described in §2.4.2.

The generation of a subscriber's "shared secret data" from his unique A-key is described in §2.2.

A procedure to verify the manual entry of the A-key is discussed in §2.1.

Example test data are presented in §4.

Manufacturers are cautioned that no mechanisms should be provided for the display at the mobile station (or any other equipment that may be interfaced with it) of A-key, SSD_A, SSD_B, or other cryptovariables associated with the cryptographic functions described in this document. The invocation of test mode in the mobile station must not alter the operational values of A-key, SSD_A, SSD_B or other cryptovariables.

Note: The notation 54§ is used to indicate the referenced section of IS-54. The notation 0x indicates a hexadecimal (base 16) number.

1.1 Definitions

AAV — Authentication Algorithm Version, an 8-bit constant equal to hexadecimal 0xC7, used in the algorithm. Use of different values for this constant in some future version would allow other "versions" or "flavors" of the basic CAVE algorithm.

A-key — A 64-bit cryptographic key variable stored in the semi-permanent memory of the mobile station and also known to the home location register (HLR) of the mobile switch. It is entered once from the keypad of the mobile station when the mobile station is first put into service with a particular subscriber, and usually will remain unchanged unless the operator determines that its value has been compromised. It is used in the SSD Update transaction.

AND — Bitwise logical AND function.

CAVE — Cellular Authentication and Voice Encryption algorithm.

CaveTable — A 256-bit Table of 8-bit quantities, divides into table0 and table1.

CMEA	Cellular Message Encryption Algorithm.
CMEA-key	Eight different 8-bit registers identified separately as k0, k1, ... k7 or cmeakey[0 through 7]. The data in these registers results from the action of the CAVE algorithm and is used to encrypt certain messages.
iteration	Multi-Round execution of the CAVE algorithm. All applications of CAVE throughout this appendix use either 4 or 8 rounds per iteration.
k0,k1...k7	Eight 8-bit registers whose contents constitute the CMEA-key.
LFSR	Linear Feedback Shift Register.
LFSR_A	The A register, a synonym for bits 31-24 of the LFSR.
LFSR_B	The B register, a synonym for bits 23-16 of the LFSR.
LFSR_C	The C register, a synonym for bits 15-8 of the LFSR.
LFSR_D	The D register, a synonym for bits 7-0 of the LFSR.
LFSR-Cycle	An LFSR-cycle consists of the following steps: 1. Compute the value of bit A7 using the formula A7 = B6 XOR D2 XOR D1 XOR D0. Save this value temporarily without changing the prior value of the A7 bit in the A register. 2. Perform a linked 1-bit right shift on the four registers A,B,C,D, and discard the D0 bit which has been shifted out. 3. Use the previously computed and stored value of bit A7 from the first of these three statements.
LSB	Least Significant Bit.
MSB	Most Significant Bit.
OR	Bitwise logical inclusive OR function.
Offset1	An 8-bit quantity that points to one of the 256 4-bit values in table0. Arithmetic operations on Offset1 are performed modulo 256. Also called offset_1.
Offset2	An 8-bit quantity that points to one of the 256 4-bit values in table1. Arithmetic operations on Offset2 are performed modulo 256. Also called offset_2.
Round	A round is one individual execution of the CAVE algorithm.
R00-R15	Sixteen separate 8-bit mixing registers used in the CAVE algorithm. Also called register[0 through 15].
SSD	SSD is an abbreviation for Shared Secret Data. It consists of two quantities, SSD_A and SSD_B. A new value of the SSD quantities may be generated in the mobile station when desired by the operator via the SSD Update message transaction.
SSD_A	A 64-bit binary quantity in the semi-permanent memory of the mobile station and also known to the serving MSC. It is used in the computation of the authentication response.
SSD_A_NEW	The revised 64-bit quantity held separately from SSD_A, generated as a result of the SSD generation process.
SSD_B	A 64-bit binary quantity in the semi-permanent memory of the mobile station and also known to the serving MSC. It is used in the computation of the CMEA and VPM.
SSD_B_NEW	The revised 64-bit quantity held separately from SSD_B, generated as a result of the SSD generation process.

table0 — The low-order 4 bits of the 256-byte lookup table used in the CAVE algorithm. Computed as `CaveTable[] AND 0x0F`.

table1 — The high-order 4 bits of the 256-byte lookup table used in the CAVE algorithm. Computed as `CaveTable[] AND 0xF0`.

VPM — Voice Privacy Mask. This name describes two different 260-bit binary data values. One is XORed with the bits in the Forward Digital Traffic Channel and the other is XORed with the bits in the Reverse Digital Traffic Channel to provide so-called "voice privacy." The VPM used for BS to MS transmission is the Forward VPM; that for MS to BS transmission is the Reverse VPM.

XOR — Bitwise exclusive OR, also known as ring sum or binary bit sum/difference without carry/borrow.

2. CAVE Process

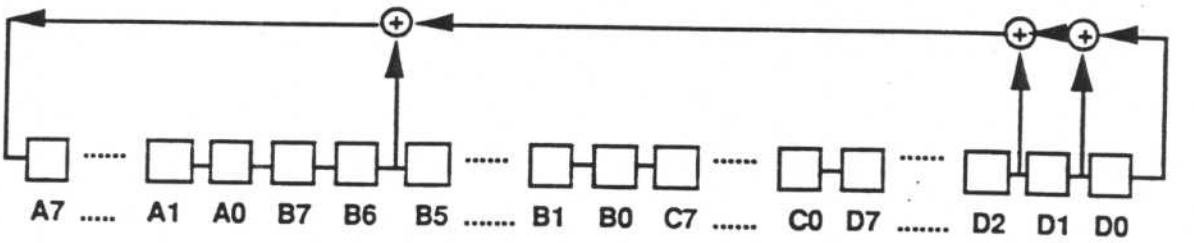
CAVE is a software-compatible non-linear mixing function, shown in Exhibit 2-1. Its primary components are a 32-bit linear-feedback shift register (LFSR), sixteen 8-bit mixing registers, and a 256-entry lookup table. The table is organized as two (256 x 4 bit) tables. The 256 byte table is listed in Exhibit 2-3. The low order 4 bits of the entries comprise *table0* and the high order 4 bits of the entries comprise *table1*.

The pictorial arrangement of Exhibit 2-1 shows that the linear-feedback shift register (LFSR) consists of the 8-bit register stages A, B, C, and D. The CAVE process repeatedly uses the LFSR and table to randomize the contents of the 8-bit mixing register stages R00, R01, R02, R03, R04, R05, R06, R07, R08, R09, R10, R11, R12, R13, R14, and R15. Two lookup table pointer offsets further randomize table access. Finally, eight 16-entry permutation recipes are embedded in the lookup tables to "shuffle" registers R00 through R15 after each computational "round" through the algorithm.

The algorithm operation consists of three steps: an initial loading, a repeated randomization consisting of 4 or 8 "rounds", and processing of the output. Initial loading consists of filling the LFSR, register stages R00 through R15, and the pointer offsets with information that is specific to the application. The randomization process is common to all cases that will be described in the later sections. Randomization is a detailed operation; it is described below by means of Exhibits 2-1, 2-2, and 2-3. The output processing utilizes the final (randomized) contents of R00 through R15 in a simple function whose result is sent to a subsequent task.

The CAVE Algorithm may be applied in an number of different cases. In each, there are different initialization requirements, and different output processing. All cases described in IS-54-B §2.3.12.1.4 through §2.3.12.1.8 are detailed in §2.1 through §2.4 of this appendix.

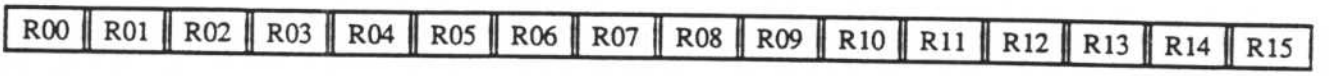
Exhibit 2-1 CAVE Elements



CAVE Linear Feedback Shift Register Block Diagram

A7(input) = B6 XOR D2 XOR D1 XOR D0

Mixing Registers:



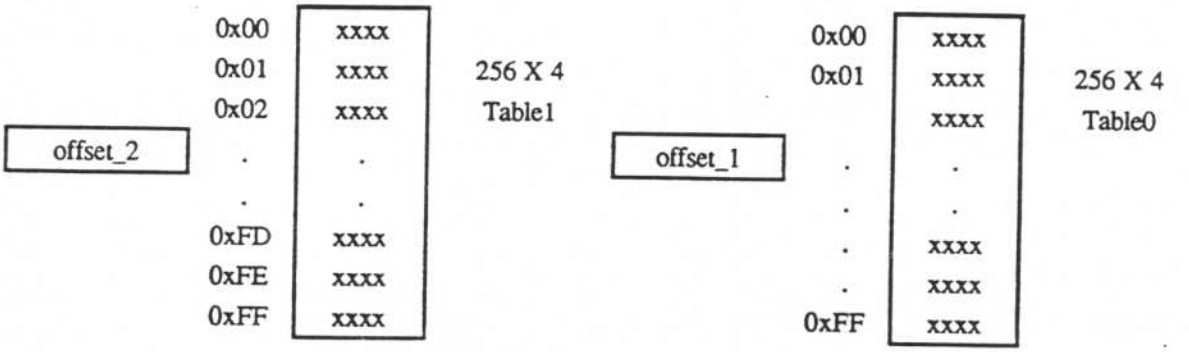


Exhibit 2-2 CAVE algorithm

```
LOMASK = 0x0F;
HIMASK = 0xF0;
Inputs:
   number_of_rounds: integer;     /* will be either 8 or 4 */
   LFSR: 32 bit integer;
   /* LFSR_A: 8 MSBs of LFSR;
   *  LFSR_D: 8 LSBs of LFSR; */
   offset_1, offset_2: unsigned 8 bit integer;
   register[0 through 15]: 8 bit integer;  /* R00 to R15 */
   T[0 through 15] : 8 bit integer;        /* temporary registers */
   CaveTable[0 through 255]: 8 bit integer;
Variables:
   temp_reg0: 8 bit integer;
   lowNibble: 8 bit integer;
   hiNibble:  8 bit integer;
   temp:    8 bit integer;
   round_index:     integer;
   R_index:    integer;
   fail_count: integer;
Functions:
```

```
1                  LFSR_cycle();                /* perform an LFSR cycle */
2                  Rotate_right_registers();    /* rotate the mixing registers */
3
4               For round_index = number_of_rounds-1 to 0
5                  {
6                  /* Save R0 for re-use later    */
7                  temp_reg0 = register[0];
8                  For each register from R_index = 0 to 15
9                     {
10                    fail_count = 0;
11                    while (TRUE)
12                       {
13                       offset_1 = offset_1+(LFSR_A XOR register[R_index]);
14                       lowNibble = CaveTable[offset_1] AND LOMASK;
15                       if (lowNibble is equal to register[R_index] AND LOMASK)
16                          {
17                          do LFSR_cycle;
18                          fail_count = fail_count + 1;
19                          if (fail_count is equal to 32)
20                             {
21                             LFSR_D = LFSR_D + 1;  /* no carry to LFSR_C */
22                             break while;          /* with no carry to LFSR_C */
23                             }
24                          }
25                       else
26                          break while;
27                       }
28
29                    fail_count = 0;
30                    while (TRUE)
31                       {
32                       offset_2 = offset_2 + (LFSR_B XOR register[R_index]);
33                       hiNibble = CaveTable[offset_2] AND HIMASK;
34                       if (hiNibble is equal to register[R_index] AND HIMASK)
35                          {
36                          do LFSR_cycle;
37                          fail_count = fail_count + 1;
38                          if (fail_count is equal to 32)
39                             {
40                             LFSR_D = LFSR_D + 1;  /* no carry to LFSR_C */
41                             break while;          /* with no carry to LFSR_C */
42                             }
43                          }
44                       else
45                          break while;
46                       }
47
48                    temp = (lowNibble OR hiNibble);
49                    if (R_index is equal to 15)
50                       register[R_index]=temp_reg0 XOR temp;
51                    else
52                       register[R_index] = register[R_index+1] XOR temp;
53                    do LFSR cycle;
54                    }
55                 do Rotate_right_registers;
56
57
58                 /* Shuffle the mixing registers */
```

```
    For each register from R_index = 0 to 15
       {
       temp = CaveTable[16*round_index + R_index] AND LOMASK;
       T[temp] = register[R_index];
       }
    For each register from R_index = 0 to 15
       register[R_index] = T[R_index];
    }

Function:
LFSR_cycle();
   {
   Variable:
      temp: 1 bit binary;
   temp = LFSR_B[6] XOR LFSR_D[2] XOR LFSR_D[1] XOR LFSR_D[0];
   Shift right LFSR    /* Discard LFSR_D[0] bit */
   LFSR_A[7] = temp;
   }


Function:         /* 128 bit cyclic shift right on R00 to R15 */
Rotate_right_registers();
   {
   Variable:
      temp_reg: 8 bit binary;
   temp_reg = register [15];
   For each register from R_index = 15 to 1
      {
      Shift register[R_index] one place right; /* set MSB = 0 */
      If (register[R_index-1] AND 0x1)
         register[R_index] = register[R_index] OR 0x80;
      }
   Shift register[0] one place right;  /* set MSB = 0  */
   if (temp_reg AND 0x1)
      register[0] = register[0] OR 0x80;
   }
```

Exhibit 2-3 CAVE Table

table0 is comprised by the 4 LSBs of the array
table1 is comprised by the 4 MSBs of the array

hi/lo	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	D9	23	5F	E6	CA	68	97	B0	7B	F2	0C	34	11	A5	8D	4E
1	0A	46	77	8D	10	9F	5E	62	F1	34	EC	A5	C9	B3	D8	2B
2	59	47	E3	D2	FF	AE	64	CA	15	8B	7D	38	21	BC	96	00
3	49	56	23	15	97	E4	CB	6F	F2	70	3C	88	BA	D1	0D	AE
4	E2	38	BA	44	9F	83	5D	1C	DE	AB	C7	65	F1	76	09	20
5	86	BD	0A	F1	3C	A7	29	93	CB	45	5F	E8	10	74	62	DE
6	B8	77	80	D1	12	26	AC	6D	E9	CF	F3	54	3A	0B	95	4E
7	B1	30	A4	96	F8	57	49	8E	05	1F	62	7C	C3	2B	DA	ED
8	BB	86	0D	7A	97	13	6C	4E	51	30	E5	F2	2F	D8	C4	A9
9	91	76	F0	17	43	38	29	84	A2	DB	EF	65	5E	CA	0D	BC
A	E7	FA	D8	81	6F	00	14	42	25	7C	5D	C9	9E	B6	33	AB
B	5A	6F	9B	D9	FE	71	44	C5	37	A2	88	2D	00	B6	13	EC
C	4E	96	A8	5A	B5	D7	C3	8D	3F	F2	EC	04	60	71	1B	29
D	04	79	E3	C7	1B	66	81	4A	25	9D	DC	5F	3E	B0	F8	A2
E	91	34	F6	5C	67	89	73	05	22	AA	CB	EE	BF	18	D0	4D
F	F5	36	AE	01	2F	94	C3	49	8B	BD	58	12	E0	77	6C	DA

2.1 A-Key Verification

The default value of the A-key when the mobile station is shipped from the factory will be all binary zeros. The value of the A-key is specified by the operator and is to be communicated to the subscriber according to the methods specified by each operator. A multiple NAM mobile station will require multiple A-keys, as well as multiple sets of the corresponding cryptovariables per A-key.

When the A-key digits are entered from the keypad, the number of digits entered is to be at least 6, and may be any number of digits up to and including 26. The entry procedure specified by the manufacturer of the mobile station shall unambiguously indicate the end of the sequence of entered digits. In a case where the number of digits entered at the keypad is less than 26, the leading most significant digits will be set equal to zero, in order to produce a 26 digit quantity called the "entry value".

The verification procedure checks the accuracy of the 26 decimal digit entry value. The first 20 digits are converted into a 64-bit representation to serve as an input to CAVE, along with the mobile station's ESN. CAVE is then run in the same manner as the authentication mode, and its 18-bit response compared to the binary equivalent of the last six entered digits. A match will cause the 64-bit pattern to become written to the subscriber's semi-permanent memory as the A-key. Furthermore, the SSD_A and the SSD_B

will be set to zero. In the case of a mismatch, an erroneous result is displayed to the subscriber by appropriate indications specified by the manufacturer of the mobile station, and no internal data is updated.

The first decimal digit of the "entry value" is considered to be the most significant of the 20 decimal digits, followed in succession by the other nineteen. The twenty-first digit is the most significant of the check digits, followed in succession by the remaining five. A decimal to binary conversion process converts both digit sequences into their equivalent mod-2 representation. For example, the 26 digits

1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0, 1 3 1 1 3 6

has a hexadecimal equivalent of

A B 5 4 A 9 8 C E B 1 F 0 A D 2, 2 0 0 4 0.

CAVE will be initialized and run as follows. First, the 32 most significant bits of the 64 bit entered number will be loaded into the LFSR. If this 32 bit pattern fills the LFSR with all zeros, then the LFSR will be loaded with the ESN. Then the entire 64 bit entered number will then be put into R00 through R07. The least significant 24 bits will be repeated into R09, R10, and R11. Authentication Algorithm Version (hexadecimal C7) will occupy R08, and ESN will be loaded into R12 through R15. CAVE will then be performed for eight rounds, as described in §2. AUTHR is obtained from the final value of CAVE registers R00, R01, R02, R13, R14, and R15. The two most significant bits of AUTHR are equal to the two least significant bits of R00 Xor R13. The next eight bits of AUTHR are equal to R01 Xor R14. Finally, the least significant bits of AUTHR are equal to R02 Xor R15.

The 18 bit answer will be compared to the binary equivalent of the last six entered digits. A match will enable semi-permanent storage. A mismatch can initiate corrective action.

Exhibit 2.1-1 Pseudo-code for A-key verification

```
In case of A-key verification
   {
   if (32 MSBs of A-key are not equal to 0)
      LFSR = 32 MSBs of A-key;
   else
      LFSR = ESN;
   register[0 through 7] = A-key;
   register[8] = AAV;
   register[9 through 11] = 24 LSBs of A-key;
   register[12 through 15] = ESN;
   number_of_rounds = 8;
   offset_1 = offset_2 = 128;
   do CAVE;
   Answer[2] = (register[0] XOR register[13]) AND 0x3;
   Answer[1] = register[1] XOR register[14];
   Answer[0] = register[2] XOR register[15];
   if (Answer is equal to A-key check_sum)
      A-key is verified;
   else
      A-key is not verified;
   }
```

2.2 SSD Generation and Updating

Refer to 54§2.3.12.1.8, "Updating the Shared Secret Data (SSD)". This section adds the details to enable implementation in subscriber and base equipment. Exhibit 2.2-1 shows the process graphically. Exhibit 2.2-2 indicates the operations in pseudo-code.

The input variables for this procedure are: RANDSSD (56 bits), Authentication Algorithm Version (8 bits), ESN (32 bits), and A-key (64 bits). CAVE will be initialized as follows. First, the LFSR will be loaded with the 32 least significant bits of RANDSSD XOR'd with the 32 most significant bits of A-key XOR'd with the 32 least significant bits of A-key. If the resulting bit pattern fills the LFSR with all zeroes, then the LFSR will be loaded with the 32 least significant bits of RANDSSD to prevent a trivial null result.

Registers R00 through R07 will be initialized with A-key, R08 will be the 8-bit Authentication Algorithm Version (11000111). R09, R10, and R11 will be the most significant bits of RANDSSD, and the ESN will be loaded into R12 through R15. Offset1 and Offset2 will initially be set to 128.

CAVE will be run for 8 rounds as previously described in §2. When this is complete, registers R00 through R07 will become SSD_A_NEW and Registers R08 through R15 will become SSD_B_NEW.

Exhibit 2.2-1 Generation of SSD_A and SSD_B

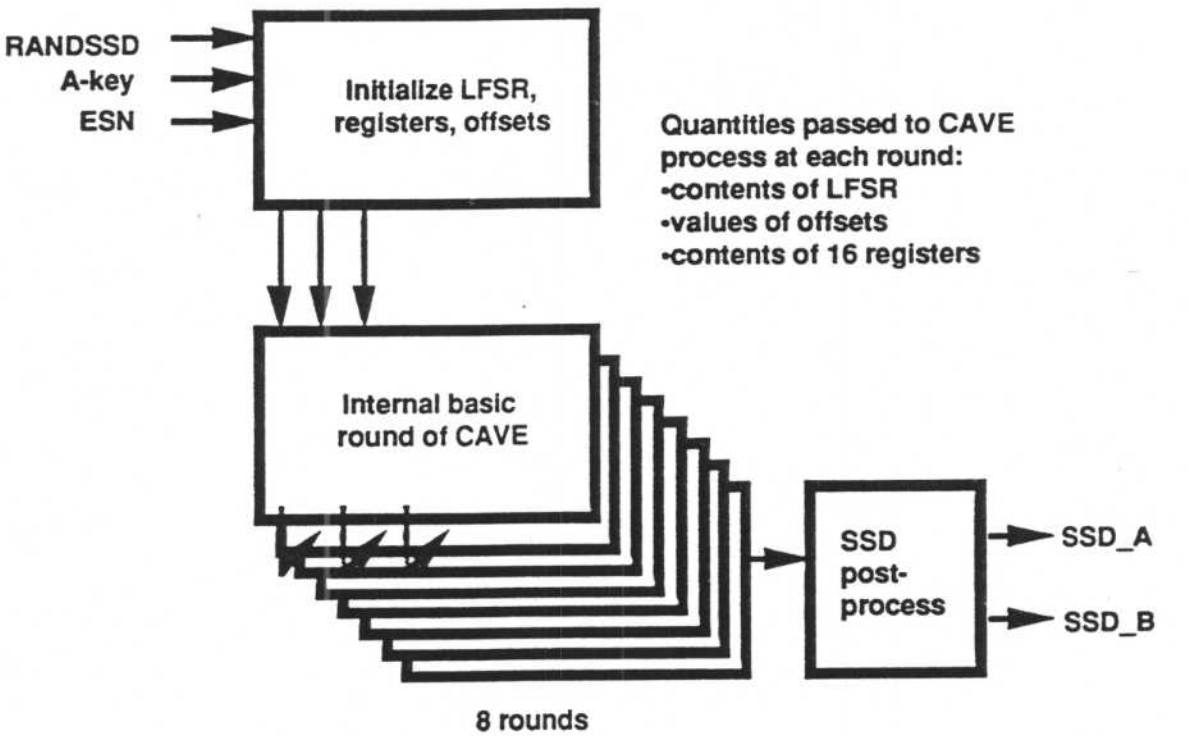


Exhibit 2.2-2 Pseudo-code for SSD Update

```
In case of SSD Update
   {
   number_of_rounds = 8;
   LFSR = 32 LSBs of RANDSSD;
   LFSR = LFSR XOR 32 MSBs of A-key XOR 32 LSBs of A-key;
   if (LFSR is equal to 0)
      LFSR = 32 LSBs of RANDSSD;
   register[0 through 7] = A-key;
   register[8] = AAV;
   register[9 through 11] = 24 MSBs of RANDSSD;
   register[12 through 15] = ESN;
   offset_1 = offset_2 = 128;
   do CAVE;
   SSD_A_NEW = register[0 through 7];
   SSD_B_NEW = register[8 through 15];
   }
```

2.3 Calculation of AUTHR and AUTHU

During any of the authentication procedures, as in 54§2.3.12.1.4, 2.3.12.1.5, 2.3.12.1.6, or 2.3.12.1.7, the secret data consists of SSD_A. R12 through R15 are initialized with the ESN. CAVE is run for eight rounds. The 18-bit result AUTHR is sent in word C where the MSB of AUTHR is next to the RANDC field and the LSB of AUTHR is next to the Parity field. The complete initial loading for AUTHR calculations is shown in tabular form as Exhibit 2.3-1. Exhibit 2.3-2 shows the process in graphical form, while pseudo-code for the process is given in Exhibit 2.3-3. A more detailed explanation of case-dependent initial loading procedures follows. AUTHU is the response in the case of a unique challenge. Outside of the difference in variable initialization indicated in Exhibit 2.3-1, the treatment of AUTHU is identical to that of AUTHR, and either variable will be referred to as AUTHR/AUTHU in the following.

The loading of R09, R10, and R11 is case-dependent. For MS registrations, MS terminations, and the unique challenge-response cases, these three registers are initially loaded with MIN1. For MS originations, a subset of the dialed digits is loaded into R09 through R11. IS-54 (Figure 2.3.12.1.6-1) states that the last 6 dialed digits transmitted by the mobile station are to be used. Each dialed digit is represented by a non-zero 4-bit code value. MIN1 is used to initially fill R09, R10, R11 and then the last dialed digits entered by the subscriber are used to replace all or part of this initial value. If a full 6 digits are dialed, the first digit of the 6 that was dialed is used as the most significant 4 bits of R09. The second digit is the least significant 4 bits of R09. The third digit is the most significant bit of R10. The fourth digit is the least significant 4 bits of R10. The fifth digit is the most significant 4 bits of R11. The sixth digit is the least significant 4 bits of R11. If less than 6 digits are dialed, then the least significant 4 bits of R11 are the last dialed digit, the second-last dialed digit becomes the most significant 4 bits of R11, and so on up to the first of the dialed digits.

The initial loading of the LFSR is also case-dependent. For MS registrations, MS terminations, and MS originations, the LFSR will be initially loaded with the 32-bit RAND. For the unique challenge-response case, the 24 most significant bits of the LFSR will be loaded with the special 24-bit RANDU, and the 8 least significant bits of the LFSR will be

loaded with the 8 least significant bits of MIN2. For all cases, the initial LFSR load will then be XOR'd with the 32 most significant bits of SSD_A XOR'd with the 32 least significant bits of SSD_A, then reloaded into the LFSR. If the resulting bit pattern fills the LFSR with all zeroes, then the LFSR will be restored to its initial load prior to the SSD_A XOR operation to prevent a trivial null result.

For all cases, the 18-bit authentication result AUTHR/AUTHU is obtained from the final value of CAVE registers R00, R01, R02, R13, R14, and R15. The two most significant bits of AUTHR/AUTHUare equal to the two least significant bits of R00 XOR R13. The next eight bits of AUTHR/AUTHUare equal to R01 XOR R14. Finally, the least significant bits of AUTHR/AUTHUare equal to R02 XOR R15.

Exhibit 2.3-1 CAVE Initial Loading for AUTHR/AUTHU Calculations

CAVE Item	CASE: MS Registration	Unique Challenge-Response	MS Origination	MS Termination
LFSR	RAND	RANDU (24 bits) MIN2 (8 bits)	RAND	RAND
Reg [0-7]	SSD_A	SSD_A	SSD_A	SSD_A
Reg [8]	AAV	AAV	AAV	AAV
Reg [9-11]	MIN1	MIN1	Subset of Dialed Digits	MIN1
Reg [12-15]	ESN	ESN	ESN	ESN

Exhibit 2.3-2 Calculation of AUTHR and AUTHU

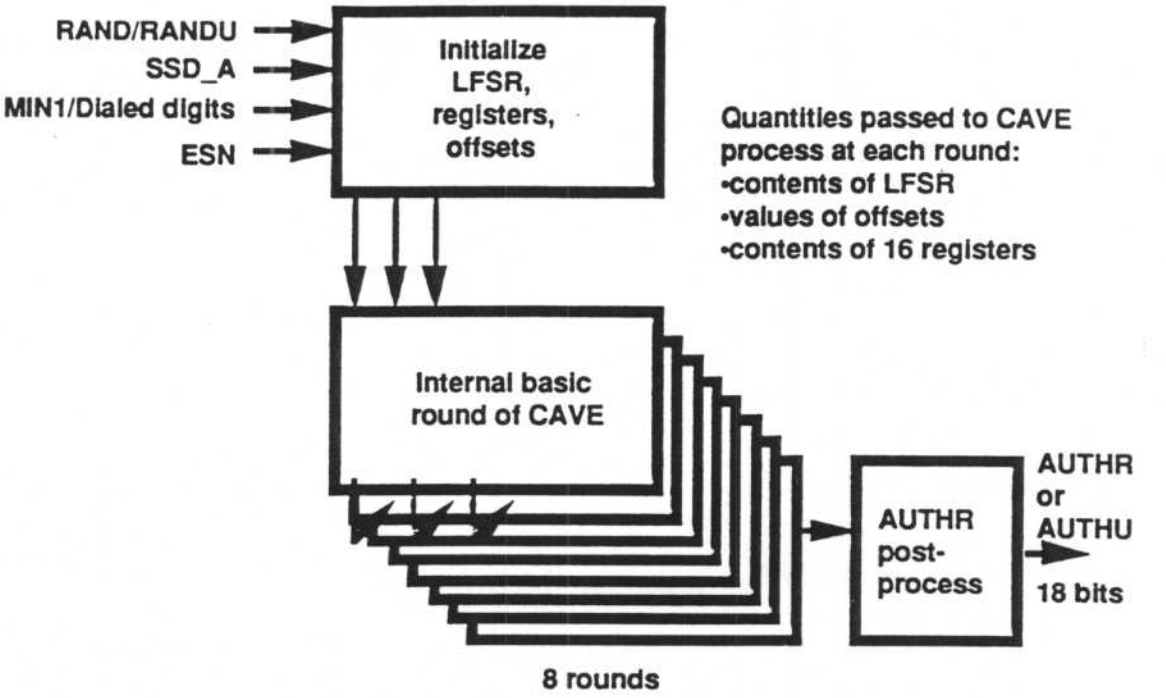


Exhibit 2.3-3 Pseudo-code for calculation of AUTHR

```
Variables:
   AUTHR: 18 bit binary value;    /* same for AUTHU */
   AUTHR[2]: 2 MSBs of AUTHR;
   AUTHR[1]: next 8 MSBs of AUTHR;
   AUTHR[0]: 8 LSBs of AUTHR;

In case of AUTHR/AUTHU calculation
   {
   number_of_rounds = 8;
   if (case is MS reg. or MS origination or MS termination)
      LFSR = RAND XOR 32 MSBs of SSD_A XOR 32 LSBs of SSD_A;
      if (LFSR is equal to 0)
         LFSR = RAND;
   if (case is unique challenge-response)
      LFSR = (RANDU<<8 OR 8 LSBs of MIN2)
               XOR 32 MSBs of SSD_A XOR 32 LSBs of SSD_A;
/*  RANDU<<8 indicates left shift 8 places  */
      if (LFSR is equal to 0)
         LFSR = (RANDU<<8 OR 8 LSBs of MIN2);
   register[0 through 7] = SSD_A;
   register[8] = AAV;
   register[9 through 11] = MIN1;
   If (case is MS origination)
      replace a nibble of register[9 through 11] with each of the
      last 1 to 6 dialed digits such that the low nibble of R11
      contains the last dialed digit;
   register[12 through 15] = ESN;
   offset_1 = offset_2 = 128;
   do CAVE;
   AUTHR[2] = (register[0] XOR register[13]) AND 0x03;
```

```
AUTHR[1] = register[1] XOR register[14];
AUTHR[0] = register[2] XOR register[15];  /* same for AUTHU */
}
```

2.4 CMEA key and VPM Generation

The process for generation of CMEA key and voice privacy mask (VPM) will generally be most efficient when concatenated together as described in the following. The post-authentication cryptovariables to be used are those from the last global challenge, and not those from any unique challenge. See Exhibits 2.3-2 and 2.3-3 for graphical detail of the generation process.

If the AUTH bit in the overhead message train is zero, the mobile station shall ignore the VPM and MEM bits is the Initial Traffic Channel Designation Message, Initial Voice Channel Designation Message, and the Status Message. The mobile station shall not apply the Message Encryption Mask or Voice Privacy Mask.

2.4.1 CMEA key Generation

Refer to Exhibit 2.3-2 or 2.3-3, "CMEA Key Generation and Voice Privacy Mask Generation." Eight bytes of CMEA session key are derived by running CAVE through an 8-round iteration and then two 4-round iterations following an authentication. This is shown in the upper portion of Exhibits 2.3-2 and 2.3-3. The post-authentication initialization and output processing requirements are as follows:

- First, the LFSR will be re-initialized to the exclusive-or sum of its post-authentication contents and both halves of SSD_B. If the resulting bit pattern fills the LFSR with all zeroes, then the LFSR will be loaded with RAND.
- Second, registers R00 through R07 will be initialized with SSD_B instead of SSD_A.
- Third, Registers R09, R10, and R11 should be loaded as the the AUTHR description of §2.3.
- Fourth, Registers R12 through R15 should be loaded with ESN.
- Fifth, the offset table pointers will begin this process at their final authentication value, rather than being reset to a predetermined state.
- Sixth, the LFSR is loaded before the second and third post-authentication iterations with a "roll-over RAND" comprised of the contents of R00, R01, R14, and R15. If the resulting bit pattern fills the LFSR with all zeroes, then the LFSR will be loaded with RAND.

The CMEA bytes drawn from iterations two and three are labelled:

- k0 = register[4] XOR register[8]; (iteration 2)
- k1 = register[5] XOR register[9]; (iteration 2)
- k2 = register[6] XOR register[10]; (iteration 2)
- k3 = register[7] XOR register[11]; (iteration 2)

- k4 = register[4] XOR register[8]; (iteration 3)
- k5 = register[5] XOR register[9]; (iteration 3)
- k6 = register[6] XOR register[10]; (iteration 3)
- k7 = register[7] XOR register[11]; (iteration 3)

2.4.2 Voice Privacy Mask Generation

VPM generation is a continuation of the CMEA key generation and should be performed at the same time under the same conditions as the CMEA key. CAVE is run for eleven iterations beyond those that produced the CMEA bytes. Each iteration consists of four rounds. The CAVE registers R00 through R15 are not reset between iterations, but the LFSR is reloaded between iterations with the "rollover RAND" as described in §2.4.1. The mask bit assignments are as follows; MS transmit/BS receive are the first 260 bits generated by this process starting at iteration 4 and concluding during iteration 9. MS receive/BS transmit are the remaining bits generated during iteration 9 and ending with the final bit of iteration 14. For each case, the most significant bit will be the highest order bit of the first XOR sum, followed in order by the remaining bits of that sum. The next highest bit will be the high bit of the second sum, etc. The least significant bit of MS transmit/BS receive case will be bit 4 of (R04 XOR R10) during iteration 9. The following bit, bit 3 of that word, will be the MSB of the mask for the MS receive/BS transmit case.

The coded Class1 bits cc0[i] and cc1[i] (see 54§2.1.3.3.3.4) represent information bits 0 thru 177 in the following order:

cc0[0]; cc1[0]; cc0[1]; cc1[1]; ...; cc0[88]; cc1[88].

The Class 2 bits CL2[i] (see IS-54 Table 2.1.3.3.4-1) represent Information bits 178 thru 259 in the following order:

CL2[0]; CL2[1]; ...; CL2[81].

The MS transmitter will XOR information bit 0 of its frame with the most significant bit (bit 7) of (R02 XOR R08) from iteration 4. Bit 1 will be XOR'd with bit 6, etc. until information bit 259 is XOR'd with bit 4 of (R04 XOR R10) from iteration 9. The BS receiver will perform the same operation on the recovered bit stream to decrypt the traffic.

In a similar fashion, the BS transmitter will XOR information bit 0 of its frame with bit 3 of iteration 9 (R04 XOR R10), bit 1 with bit 2 of iteration 9, (R04 XOR R10), etc., until information bit 259 is XOR'd with bit 0 of (R06 XOR R12) from iteration 14. The MS receiver will perform the same operation on its recovered bit stream to decrypt the traffic.

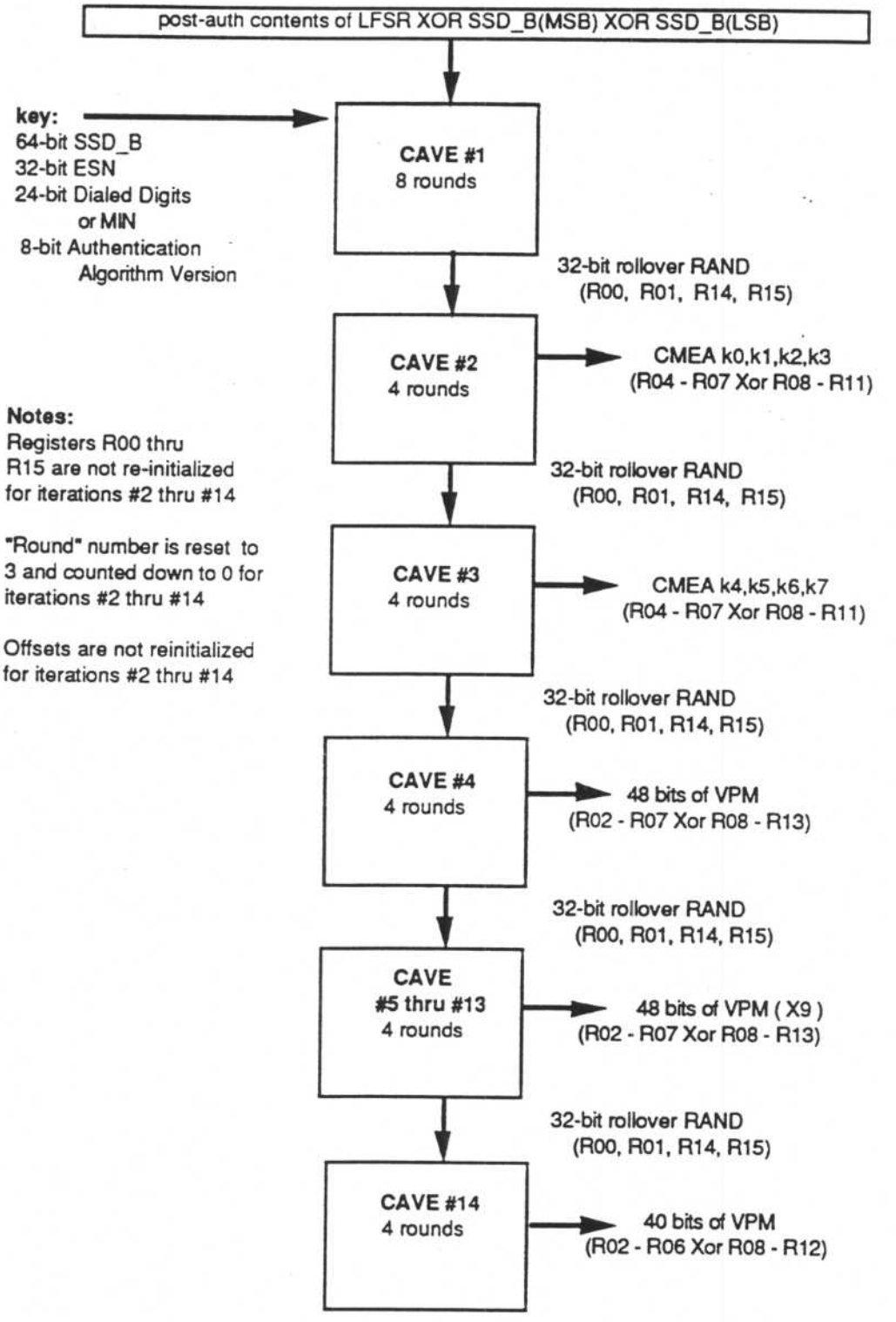
The VPM is not to be changed during a call. If VPM is not available at the time of an initial traffic channel designation upon entering the Conversation task, (typically due to calculation delay in its generation) then a VPM of all zeros is to be used until the operational VPM has been completely generated.

Exhibit 2.4-1 Pseudo-code for CMEA key and VPM generation

```
Outputs:
   cmeakey[0 through 7]: 8 bit binary key values; /* k0 - k7 */
   VPM[0 through 64]: 8 bit binary values (520 bits total)
   /* first 260 bits of VPM are Reverse Mask
    * second 260 bits generated are Forward Mask */

In case CMEA key & VPM generation
   {
   /* Iteration 1: first pass through CAVE */
   number_of_rounds = 8;
   LFSR = post_auth_LFSR;
   LFSR = LFSR XOR 32 MSBs of SSD_B XOR 32 LSBs of SSD_B;
   if (LFSR is equal to 0)
      LFSR = RAND;
   register[0 through 7] = SSD_B;
   register[8] = AAV;
   register[9 through 11] = MIN1;
   if (case is MS origination)
      replace a nibble of register[9 through 11] with each of the
      last 1 to 6 dialed digits such that the low nibble of R11
      contains the last dialed digit;
   register[12 through 15] = ESN;
   offset_1 = post_auth_offset_1; /* restore offsets to their */
   offset_2 = post_auth_offset_2; /* post authentication values */
   do CAVE;

   /* Iteration 2: generation of first CMEA key parameters */
   number_of_rounds = 4;
   do roll_LFSR;
   do CAVE;
   cmeakey[0] = register[4] XOR register[8];
   cmeakey[1] = register[5] XOR register[9];
   cmeakey[2] = register[6] XOR register[10];
   cmeakey[3] = register[7] XOR register[11];

   /* Iteration 3: generation of next CMEA key parameters */
   number_of_rounds = 4;
   do roll_LFSR;
   do CAVE;
   cmeakey[4] = register[4] XOR register[8];
   cmeakey[5] = register[5] XOR register[9];
   cmeakey[6] = register[6] XOR register[10];
   cmeakey[7] = register[7] XOR register[11];

   /* Iteration 4 - 13: generation of VPM */
   vpm_ptr = 0;
   For 10 repetitions
      {
      do roll_LFSR;
      number_of_rounds = 4;
      do CAVE;
      For r_ptr = 0 to 5
         {
         VPM[vpm_ptr] = register[r_ptr+2] XOR register[r_ptr+8];
         vpm_ptr = vpm_ptr + 1;
         }
```

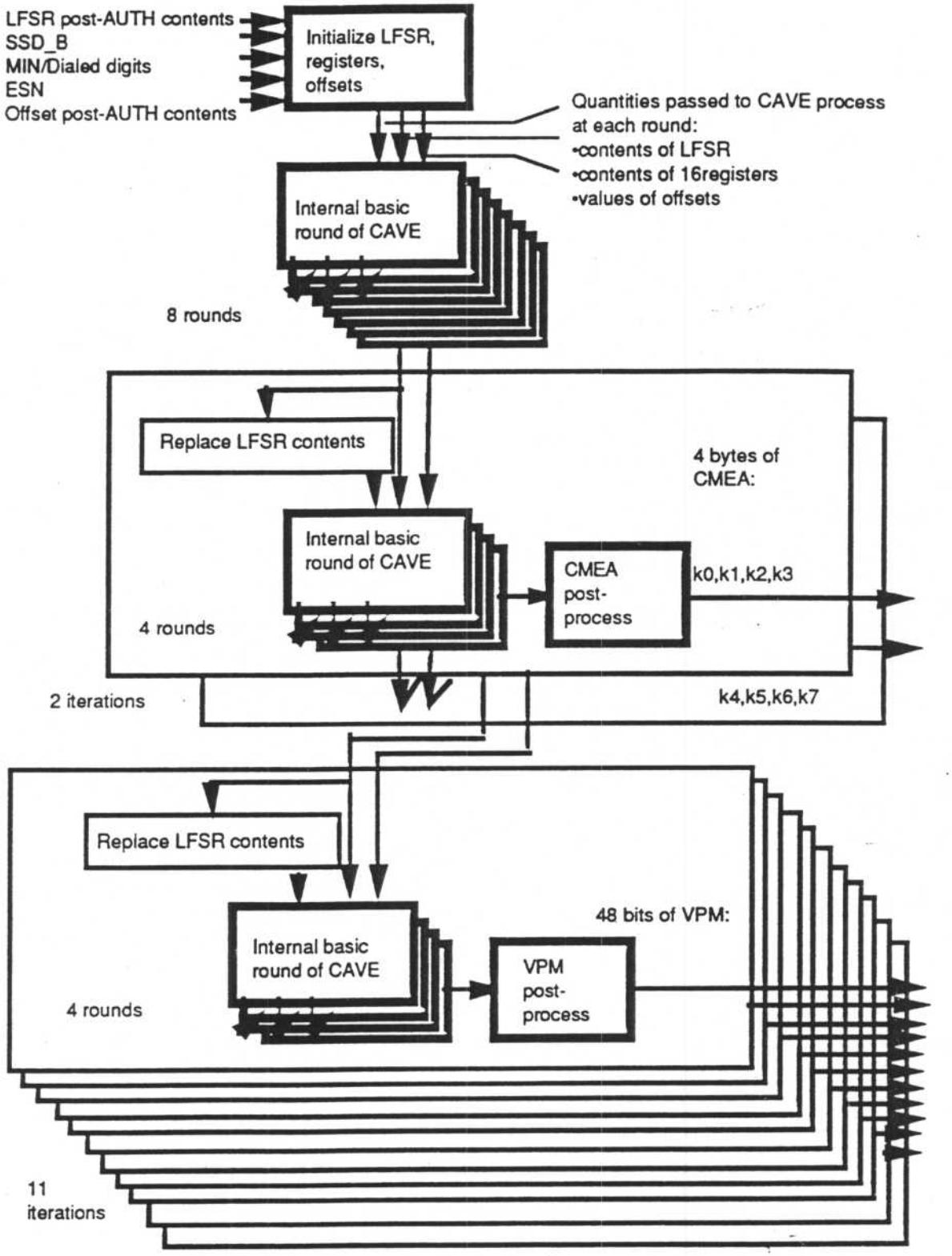
```
1                    }
2
3                 /* Iteration 14: generation of last VPM bits */
4                 do roll_LFSR;
5                 number_of_rounds = 4;
6                 do CAVE;
7                 For r_ptr = 0 to 4
8                       {
9                       VPM[vpm_ptr] = register[r_ptr+2] XOR register[r_ptr+8];
10                      vpm_ptr = vpm_ptr + 1;
11                      }
12                }
13
14
15             Function:
16             roll_LFSR;
17                {
18                LFSR_A = register[0];
19                LFSR_B = register[1];
20                LFSR_C = register[14];
21                LFSR_D = register[15];
22                if (LFSR is equal to 0)
23                LFSR = RAND;
24                }
25
```

Exhibit 2.4-2 Generation of CMEA key and VPM



1

Exhibit 2.4-3 Detailed Generation of CMEA key and VPM



2

3. CMEA Message Encryption Process

This process uses the CMEA eight-byte session key to produce enciphered messages via a unique CMEA algorithm. The process of CMEA key generation is described in §2.4. A description of the messages that are enciphered is included in this section. Note that the CMEA is generated after every origination or page response (mobile termination).

3.1 CMEA Algorithm

This algorithm encrypts and decrypts messages that are of length 8*n bits, where n is the number of message bytes. The message is first stored in an n-byte buffer called `msg_buf[]`, such that each byte is assigned to one "`msg_buf[]`" value. `msg_buf[]` will be encrypted by means of three operations before it is ready for transmission. Decryption will be performed in the same manner as encryption.

The function tbox( ) is frequently used. This is defined as:

tbox(z) = C(((C(((C(((C((zXOR k0)+k1)+z)XOR k2)+k3)+z)XOR k4)+k5)+z)XOR k6)+k7)+z

where "+" denotes modulo 256 addition,

"XOR " is the XOR function,

"z" is the function argument,

k0,...,k7 is defined above,

and C( ) is the outcome of a CAVE 8-bit table look-up. (Exhibit 2-3)

Exhibit 3.1-1 shows pseudo-code for an algorithmic procedure for tbox(). Note that all additions performed on the variables "temp" and "z" are modulo 256.

Exhibit 3.1-1 Pseudo-code for tbox

```
Input:
   z: unsigned integer range 0 to 255;
Variables:
   temp: unsigned 8 bit integer;
   k_index: integer;

tbox(z);
   {
   k_index = 0;
   temp = z;
   For 4 repetitions
      {
      temp = temp XOR cmeakey[k_index];
      temp = temp + cmeakey[k_index + 1];   /*mod 256 addition*/
      temp = z + CaveTable[temp];           /*mod 256 addition*/
      k_index = k_index + 2;
      }
   return(temp);
   }
```

The CMEA algorithm is the message encryption process used for both the encryption and decryption of a message. Each message to which the CMEA algorithm is applied must be a multiple of 8 bits in length. The CMEA algorithm may be divided into three distinct manipulations. See Exhibit 3.1-2.

Exhibit 3.1-2 Pseudo-code CMEA Algorithm

```
Inputs:
   byte_count: integer;  /* number of bytes in the message */
   msg_buf[0 to byte_count-1]: octet by octet representation of
                                          message;

Variables:
   msg_index: integer;
   k: unsigned integer range 0 to 255;
   z: unsigned integer range 0 to 255;

/* First Manipulation: */
z = 0;
For msg_index = 0 to byte_count-1
   {
   k = tbox(z XOR msg_index);
   msg_buf[msg_index] = msg_buf[msg_index]+k;  /*mod 256 addition*/
   z = z + msg_buf[msg_index];                 /*mod 256 addition*/
   }

/* Second Manipulation: */
half = byte_count/2;
For msg_index = 0 to half;
   {
   msg_buf[msg_index] = msg_buf[msg_index] XOR
                        (msg_buf[byte_count-1-msg_index] OR 0x1);
   }

/* Third Manipulation: */
z = 0;
For msg_index = 0 to byte_count-1
   {
   k = tbox(z XOR msg_index);
   z = z + msg_buf[msg_index];
   msg_buf[msg_index] = msg_buf[msg_index] - k;
   }                              /* this subtraction is mod256  */
                                  /* without borrow   */
```

3.2 Description of Messages

The following is a description of the eight messages that are enciphered. For each message, the enciphered fields are designated. The messages are grouped by channel designation.

3.2.1 Forward Voice Channel

3.2.1.1 Alert With Info (See §3.7.2.1. of IS-54)

The Alert with INFO message is encrypted. Word 1 of the Mobile Station Control Message contains the order and order qualifier fields that identify this message as **ALERT WITH INFO.** No field in Word 1 is encrypted. No field in Word 2 - First Alert With Info Word is encrypted.

The subsequent words contain a character representation. Each character transmitted is represented in IA5 form in a field of 8 bits. Each word contains up to three characters. The 24 bits that comprise the three characters in each FVC word are treated by CMEA as a single message. No additional fields are encrypted.

3.2.1.2 Flash With Info (See §3.7.2.1. of IS-54)

The Flash with INFO message is encrypted. Word 1 of the Mobile Station Control Message contains the order and order qualifier field that identify this message as **FLASH WITH INFO.** No field in Word 1 is encrypted. No field in Word 2 - Flash With Info Word is encrypted.

The subsequent words contain a character representation. Each character transmitted is represented in IA5 form in a field of 8 bits. Each word contains up to three characters. The 24 bits that comprise the three characters in each FVC word are treated by CMEA as a single message. No additional fields are encrypted.

3.2.2 Reverse Voice Channel

3.2.2.1 Called Address Message (See §4.1.2.2. of IS-54)

The 32 bits in Word D - First Word of the **Called Address Message** which comprise digits 1-8 are encrypted. These 32 bits are treated by CMEA as a new single message. No additional fields in Word D are encrypted.

The 32 bits in each Word E, F, and G of the **Called Address Message** which comprise further dialed digits are encrypted. These 32 bits are treated by CMEA as a new single message. No additional fields in these words are encrypted.

3.2.3 Forward Digital Traffic Channel

3.2.3.1 Alert With Info (See §3.7.3.1.3.2.1. of IS-54)

The FACCH message contains up to n characters each represented as 8 bits in the IA5 format. These are enciphered prior to convolutional coding. The CRC is computed on the resultant 48 bits. For the first slot of a multi-slot message (Continuation Flag = 0) all fields except the Message Type (40 bits total) are encrypted by CMEA. For the subsequent slots of a multi-slot

message (Continuation Flag =1) all fields are encrypted (total of 48 bits) by CMEA.

3.2.3.2 Flash With Info (See §3.7.3.1.3.2.14. of IS-54)

The FACCH message contains up to n characters each represented as 8 bits in the IA5 format. These are enciphered prior to convolutional coding. The CRC is computed on the resultant 48 bits. For the first slot of a multi-slot message (Continuation Flag = 0) all fields except the Message Type (40 bits total) are encrypted by CMEA. For the subsequent slots of a multi-slot message (Continuation Flag =1) all fields are encrypted (total of 48 bits) by CMEA.

3.2.4 Reverse Digital Traffic Channel

3.2.4.1 Flash With Info (See §2.7.3.1.3.2.8. of IS-54)

The FACCH message contains up to 63 characters each represented as 8 bits in the IA5 format. These are enciphered prior to convolutional coding. The CRC is computed on the resultant 48 bits. For the first slot of a multi-slot message (Continuation Flag = 0) all fields except the Message Type (40 bits total) are encrypted by CMEA. For the subsequent slots of a multi-slot message (Continuation Flag =1) all fields are encrypted (total of 48 bits) by CMEA.

3.2.4.2 Send Burst DTMF (See §2.7.3.1.3.2.9. of IS-54)

The FACCH message contains up to 64 digits each represented as 4 bits. These are enciphered prior to convolutional coding. The CRC is computed on the resultant 48 bits. For the first slot of a multi-slot message (Continuation Flag = 0) all fields except the Message Type (40 bits total) are encrypted by CMEA. For the subsequent slots of a multi-slot message (Continuation Flag =1) all fields are encrypted (total of 48 bits) by CMEA.

3.2.4.3 Send Continuous DTMF (See §2.7.3.1.3.2.10. of IS-54)

The FACCH message contains 1 digit represented as 4 bits. The message is enciphered prior to convolutional coding. The CRC is computed on the resultant 48 bits. All fields except the Message Type (40 bits total) are encrypted by CMEA.

4. Test Vectors

These two test cases utilize the following fixed input data (expressed in hexadecimal form):

RANDSSD	=	4D	18EE	AA05	895C
Authentication Algorithm Version	=				C7
MIN1	=			79	2971
MIN2	=				28D
ESN	=			D75A	96EC
msg_buf[0]	=	B6, 2D, A2, 44, FE, 9B			
...					
msg_buf[5]					

The following A-key and check digits should be entered in decimal form:

14 1421 3562 3730 9504 8808 6500

Conversion of the A-key, check digit entry into hex form will produce:

A-key, check bits = C442 F56B E9E1 7158, 1 51E4

The above entry, when combined with RANDSSD, will generate:

SSD_A = CC38 1294 9F4D CD0D

SSD_B = 3105 0234 580E 63B4

4.1 Vector 1 (MS Termination)

If RAND = 34A2 B05F:

AUTHR = 3 66F6

CMEA key k0,...k7 = A0 7B 1C D1 02 75 69 14

CMEA output = E5 6B 5F 01 65 C6

VPM = 18 93 94 82 4A 1A 2F 99 A5 39 F9 5B 4D 22 D5 7C

EE 32 AC 21 6B 26 0D 36 A7 C9 63 88 57 8C B9 57

E2 D6 CA 1D 77 B6 1F D5 C7 1A 73 A4 17 B2 12 1E

95 34 70 E3 9B CA 3F D0 50 BE 4F D6 47 80 CC B8

DF

4.2 Vector 2 (MS Termination)

If RAND = 5375 DF99:

AUTHR = 0 255A

CMEA key k0,...k7 = F0 06 A8 5A 05 CD B3 2A

CMEA output = 2B AD 16 A9 8F 32

VPM = 20 38 01 6B 89 3C F8 A0 28 48 98 75 AB 18 65 5A

49 6E 0B BB D2 CB A8 28 46 E6 D5 B4 12 B3 8C 9E

76 6C 9E D4 98 C8 A1 4A D2 DC 94 B0 F6 D4 3E E0

D1 6C 7E 9E AC 6B CA 43 02 C9 23 63 6F 61 68 E8

8F