Copyright © 1981 American Telephone and Telegraph Company
THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 60, No. 7, September 1981
Printed in U.S.A.

Digital Signal Processor:

Private Communications

By C. A. McGONEGAL, D. A. BERKLEY, and N. S. JAYANT

(Manuscript received August 2, 1980)

Where normal safeguards for message privacy are not adequate, some form of encryption is required. Voice messages, encoded using an adaptive differential pulse-code-modulation encoder such as that described in a companion paper, may be encrypted for privacy (protection against casual eavesdropping) through similar digital signal processor programs with little additional computation. Two methods of implementation are described: The use of U-permutations for temporal scrambling of the transmitted bit stream and the use of bit-masking by stored random numbers. The relative merits of each system are discussed, illustrating both the flexibility and limitations of the digital signal processor for such applications.

I. INTRODUCTION

Situations occur in everyday telephone communication systems where the normal safeguards for message privacy may not be adequate. This could happen, for example, in a radiotelephone system where message contents could be easily intercepted by unauthorized listeners.

In this paper, we discuss two simple methods for ensuring short-term privacy for such telephone systems. These methods are based on the Adaptive Differential Pulse Code Modulation (ADPCM) codec described in a companion paper.[1] Both methods modify the ADPCM transmitted code word in such a way as to randomize the bit pattern. Decoding the resulting randomized code words requires advance knowledge of the randomization key.

The techniques discussed here are non-time-varying and have limited numbers of encryption keys. Thus, the message is only protected from casual listeners. Listeners who possess the necessary equipment can determine the required decoding key. However, the system is designed so that decoding cannot be accomplished in the duration of an average conversation.

Both these methods have been implemented using the Bell Laboratories Digital Signal Processing integrated circuit (DSP)[2] with only a slight increase in processing load relative to that required by non-encrypted ADPCM encoding and decoding. The resulting system should be able to support two or three simultaneous coders or decoders per integrated circuit.

Issues of key distribution and cryptanalysis are outside the scope of this paper.[3] Our purpose, rather, is to demonstrate what can be realized in terms of adapting an existing ADPCM DSP program for a potential privacy application. With ROM and RAM capabilities greater than what are available in the present DSP, levels of privacy can be straightforwardly enhanced—for example, by layering several permutation and masking operations, as in the Digital Encryption Standard.[3]

In the following section, we discuss privacy algorithms. Then, we consider the implementation of each using the DSP. Finally, we discuss the relative merits of each system. This discussion illustrates both the flexibility and limitations of the DSP for such an application.

II. PRIVACY CODING FOR ADPCM TRANSMISSION

There are three major requirements for a simple digital privacy system:

(*i*) It must be possible to generate a "large" number of encryption keys (bit rearrangement or masking patterns) automatically and easily.

(*ii*) The encrypted speech must be unintelligible if decoded by other than the proper key.

(*iii*) The system must be sufficiently tractable so as to be implemented without significant incremental cost.

The two encryption methods we will consider are as follows:

(*i*) U-permutations[4] where the bits in a given block of ADPCM code words are permuted from their normal order, producing a temporal scrambling of the bit stream.

(*ii*) Addition of stored pseudo-random numbers to the ADPCM code words to form randomly "masked" encrypted code words.

Other methods, such as use of linear congruential random number generation[5,6] to form masks, are possible, but methods (*i*) and (*ii*) above are both practical and illustrate the principles of encryption for privacy.

2.1 U-permutations

The class of uniform (or U-permutations on N bits is defined by[4]

$$s = k_1 r (\text{mod } N);\ r,s = 1, 2, \cdots, N, \tag{1}$$

where r is the initial bit position and s is the scrambled bit position in the block of N bits. The encryption key is k_1 and must be prime to N. Unscrambling of the N-bit block is accomplished by another U-permutation

$$r = k_2 s (\text{mod } N), \tag{2}$$

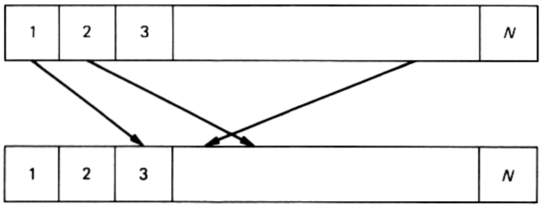
with $k_1 k_2(\text{mod } N) = 1$. Figure 1a illustrates a permutation of bits within a block of N bits, while Fig. 1b shows an example of uniform permutation for $N = 16$, $k_1 = 3$ and $k_2 = 11$.

It has been found that to satisfy the requirement for unintelligibility requires at least $N = 16$ for 24-kb/s ADPCM (8-kHz sampling, 3 bits per sample).[4] We have implemented an $N = 16$ system using 32-kb/s ADPCM (8-kHz sampling and 4 bits/sample to provide telephone quality speech). The scrambled speech is of very low intelligibility with casual listening. However, individual words from a limited vocabulary, such as spoken numbers, may be distinguishable, especially with experienced listening. An implementation with higher N faces some difficulties because of address space limitations. With the current DSP version $N > 32$ is impossible, as will be discussed later.

The number of keys in U-permutation is given by $N \cdot G(N)$, where $G(N)$ is the number of numbers that are prime to N.[4] There are 112 keys available for $N = 16$. For $N = 32$ this increases to 480 keys. The adequacy of a given number of keys depends on the application.

2.2 Random number masking

The random mask method we have considered is basically very simple. In essence, a different random number is added to each ADPCM code word before transmission and that same number is subtracted by



(a)

r	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
s	3	6	9	12	15	2	5	8	11	14	1	4	7	10	13	16

$N = 16 \quad k_1 = 3 \quad k_2 = 11$

(b)

Fig. 1—Example of (a) general bit permutation in a N-bit block and (b) uniform permutation with $N = 16$, $k_1 = 3$, and $k_2 = 11$.

the receiver. A finite table of random numbers is used and synchronization is required between transmitter and receiver.

The encryption key for any given transmission is the starting point in the random number table relative to the block synchronization. Additional keys may be produced by generating the random number mask using multiple table pointers and adding together the random numbers to form the code word mask. Using two pointers, the masking of ADPCM code word C with an L-number random table (4 bits per entry so $L = N/4$) can be written as

$$E = C + R(I + O_1) + R(I + O_2), \tag{3}$$

where $R(I)$ denotes the I-th word of the random table, E is the encrypted code word, $O_{1,2}$ are the table offsets relative to the beginning of the table and $I = 0$ at beginning of a transmission block (synchronization time). Decryption is accomplished by subtracting the same set of random numbers from E. The key words are $O_{1,2}$, leading to a maximum of L^2 possible keys of which $L(L - 1)/2$ are unique. Even for $L = 16$ the encrypted speech is essentially unintelligible and for larger L, the presence of speech is very hard to detect. (The output sounds like continuous white noise at all times.) The table size is limited by DSP ROM size of 1024 words, but $L = 512$ words is certainly practical. Two table pointers then give about 130,000 keys.

III. IMPLEMENTATION USING THE DSP

The basis for both DSP privacy implementations is the ADPCM codec discussed in the companion paper.[1] In both encryption systems the coder or decoder is recast slightly in "subroutine" form which allows more convenient handling of the block synchronization structure. Also, to avoid the problem of two's-complement sign extension, the ADPCM code word is converted to unsigned form; that is, the 4-bit code word, represented as -7 to 8 in the original coder, is offset by 7 and coded as 0 to 15.

3.1 U-permutations

The U-permutation for $N = 16$ is implemented by splitting blocks of four code words (4 bits each) into blocks of 16 one-bit words, rearranging the one-bit words according to the proper permutation and reassembling the permuted block of four words for transmission. One block of four words is being permuted, while a second block is being sent allowing a very simple program organization.

The required modulo N arithmetic is accomplished without any computation being required by overflowing the address register being used as the storage pointer. Thus, for $N = 16$, the disassembled one-bit words are stored at every fourth RAM location. The proper permutation increment is stored in the upper 8 bits of the increment register and the address sequence generated by overflow.

Assuming that the RYA register is pointed to the unsigned form of the ADPCM code word, the following DSP instructions disassemble the code word and store the resulting one-bit words in RAM. The code word is disassembled by loading the bit to be saved in bit 14 of the P register and zeroing all other P register bits. The C register is set to 2^{14}. (Notation for the DSP assembly language is given in a companion paper.[2])

```
                        i = 030;                             "permutation in-
                                                               crement"
                        rd = 0;                              "RAM storage ad-
                                                               dress"
                        a = p      p = 2048*ryz;             "get bit 1 of code
                                                               word"
                        a = p      p = 1*c;                  "set up to zero
                                                               other bits"
                        a = p&a    p = 4096*ryz;             "zero other bits;
                                                               get bit 2"
                w = a   a = p      p = 1*c;                  "set up to zero
                                                               other bits"
rdi = w                 a = p&a    p = 8192*ryz;             "save bit 1;
                                                               get bit 3"
                w = a   a = p      p = 1*c;                  "set up to zero
                                                               other bits"
rdi = w                 a = p&a    p = 16384*ryz;            "save bit 2;
                                                               get bit 4"
                w = a   a = p      p = 1*c;                  "set up to zero
                                                               other bits"
rdi = w                 a = p&a;                             "save bit 3; zero
                                                               other bits"
                w = a;
rdi = w;                                                     "save bit 4"
```

The single bits are reassembled into a code word by shifting and adding the bits using the following instructions:

```
k = 010;                                    "permutation increment"
ry = 160;;;                                 "RAM storage address"
a = p                 p = 1.*ryk;           "get bit 4"
a = p                 p = 1.*ryk;           "get bit 3"
a = p + 2*a           p = 1.*ryk;           "shift and add; get bit 2"
a = p + 2*a           p = 1.*ryk;           "shift and add; get bit 1"
a = p + 2*a;                                "code word reassembled"
```

In this scheme, the RAM values are refreshed at a 500-μs period (for an 8-kHz sampling rate) which is the maximum specified refresh time for the dynamic RAM.[2] To extend the method to $N = 32$, it is necessary to use spare program cycles (of which a sufficient number appears to be available) to supplement the "natural" RAM refresh cycles.

Permutation blocks larger than $N = 32$ bits are not possible using this approach since $2N$ words of RAM are required. Thus, $N = 64$ would fill the 128-word RAM on the DSP allowing no scratch storage as required by the basic ADPCM coder.

The decoder implementation is very similar and has identical limitations. A single DSP μ-law to μ-law codec, using $N = 16$, was implemented successfully and the resulting speech was quite well scrambled although, as mentioned before, some numbers could be distinguished with practice.

The limited number of keys in an $N = 16$ U-permutation system could present a problem in some applications. To increase the number of keys, the easiest route appears to be random number masking, which is discussed next.

3.2 *Random number masking*

Random 4-bit numbers are stored, 4 bits per word, in a ROM table. The table size is limited only by available ROM. We arbitrarily used a 256-word table for our implementation, but considerably more space is available and can be used if more keys are desired.

In single-pointer masking a pointer into the table is arbitrarily chosen. The DSP automatic (6-bit) loop counter is set to 63 and for each ADPCM code word generated, a random number is fetched, added to the code word, and the pointer incremented. When the loop count is satisfied the pointer is restored to its original value.

If multiple pointers are used, offset values are initialized and each random number is fetched and added to the code word. To avoid additional programs steps for detection of the table end, the pointers are limited to occur no later than 64 locations from the end of the table. The two-pointer version has the following requirement

$$S + \delta + 64 < N, \tag{4}$$

where S is the starting pointer, δ is the offset from S, and N is the mask table size.

All additions are made without attention to overflow out of the 4-bit code word and the least-significant four bits are transmitted. The received word then has identical masks subtracted, without regard to unsigned underflow, and the four least-significant bits are taken as the input to the ADPCM coder. In two's-complement arithmetic the final result is correct, without regard to overflow or underflow, if that result is in the required range. (The decrypted code word must satisfy this condition since the original unsigned word was in the 4-bit range.) An example of masking by this process is shown in Fig. 2.

Assuming the RX register contains the mask pointer, the K register contains the offset value and the RYA register contains a pointer to the unsigned form of the ADPCM code word, the following DSP instructions encrypt the code word:

```
a = p          p = rxk*c;      "get mask word 1"
a = p          p = rxk*c;      "get mask word 2"
a = p + a      p = 1.*w;       "add masks; get code word"
a = p + a      p = 017*c;      "subtract mask from code word"
a = p&a;                       "decrypted 4-bit code word"
```

The code word is decrypted by the following instructions. The RX and K registers contain the table pointer and offset value, respectively. The W register contains the encrypted code word.

```
a = p          p = rxk*c;      "get mask word 1"
a = p          p = rxk*c;      "get mask word 2"
a = p + a      p = 1.*rym;     "add masks; get code word"
a = p − a      p = 017*c;      "add code word"
a = p&a;                       "encrypted 4-bit code word"
```

The two-pointer encrypted codec was implemented on a single DSP with μ-law input and output. Table sizes as small as 16 words, with a single pointer, yield unintelligible scrambled speech.

To examine the synchronization properties of the system, the same

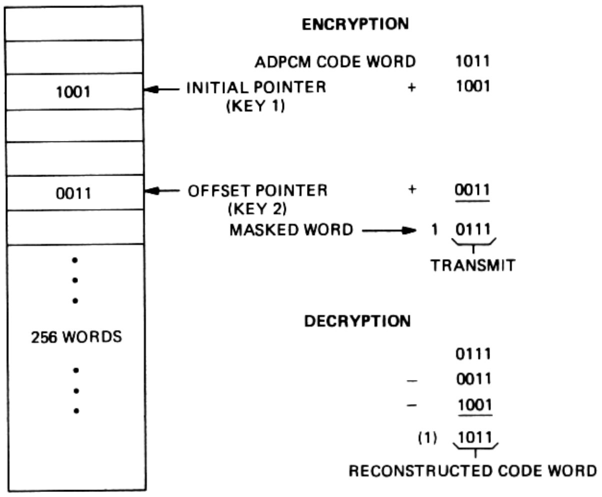


Fig. 2—Example of bit-masking with two-pointer random number encryption and decryption.

codec was also implemented in a two-DSP version. A simplified block diagram is shown for this system in Fig. 3. Digital signal processor 1 provides the encryption and transmits the encrypted bits; DSP 2 performs the decryption. In the absence of appropriate synchronization mentioned below, the output of the receiver digital-to-analog converter is scrambled. This is the same configuration used for the two-DSP codec,[1] except that provision is made for block synchronization. The status and control bits, C0 and S0, provide a "sync" bit for this purpose and, assuming synchronization is recovered externally from the transmission format, are connected separately in parallel with the main serial data-bit stream. Synchronization formatting and recovery is probably also possible within the DSP, but it is beyond the scope of this paper.

Programming the synchronization is very simple. Each time the table pointer is reinitialized the transmitter sends the control signal (S0) using the STR register, and the receiver waits for the control signal (C0) using the SYC register. Digital signal processor instructions for the transmitter and receiver are given below.

```
transmitter
        init:      ...
                   ...
                   lc = 63;          "set up loop counter"
                   str = 1;          "send control signal"
                   auc = 0x06;       "set c = 2^14, overflow"
                   str = 0;          "turn off control signal"
        loop:      ...
receiver
        init:      ...
                   lc = 63;          "set up loop counter"
                   auc = 0x06;       "set c = 2^14, overflow"
                   syc = 4           "wait for control signal"
        loop:      ...
```

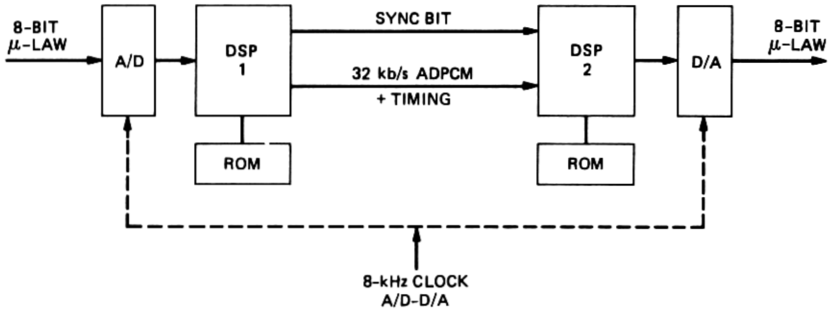


Fig. 3—Simplified block diagram of two-DSP ADPCM codec with block synchronization.

Table I—Memory utilization

Coder Type	Instructions per Sample	Memory ROM	Memory RAM
ADPCM encoder	46	228	5
U-permutation encoder $N = 16$	57	308	37
Random mask encoder $L = 256$	68	552	6
ADPCM decoder	38	156	5
U-permutation decoder $N = 16$	42	214	37
Random mask decoder $L = 256$	55	452	6

As expected, synchronization time is imperceptible and the system sounds exactly the same as does the single DSP codec.

IV. DISCUSSION

Both encrypted codecs provide adequate scrambling in terms of reduced intelligibility, although random number masking is capable of entirely destroying the impression of speech.

Program efficiency, in terms of instructions executed per 125-μs sample, is also similar as shown in Table I. Utilization of the DSP, relative to its maximum execution rate of 156 instructions per 125-s sample, ranges from 24 percent for the ADPCM decoder to 44 percent for the random mask encoder.

Table I also shows memory utilization for the different implementations. (These should be compared to 128 words of available RAM and 1024 words of available ROM.) The ADPCM codec is in subroutine form and savings of about eight instructions can be made by removing this structure at the expense of a considerably more opaque program.

The number of keys required for this type of privacy system has not been studied and other issues, such as transmission of keys, are outside the area of this paper. Clearly, in this implementation random number masking provides for more keys than U-permutations. Also, if greater levels of secrecy are required on a particular transmission link, one can envision a special transmitter/receiver pair with a unique PROM or ROM random number table used externally, thereby achieving a 2^{4k} key system. (This is because there are 2^M binary sequences of length M.) For any single transmission, i.e. a single set of pointer positions, k = 64. However, if one considers other pointer positions this number is greater and is, in general, a function of the random number table length L.

Although the programming was not discussed, the setting of the particular key to be used in a given transmission would require, for example, reading an external switch register during program initialization. Thus, one would need some simple external circuitry to divert the codec input stream at initialization (reset) time and appropriate

DSP programming to handle the input format and store the result in RAM.

V. CONCLUSION

Two privacy encryption systems, based on ADPCM coding of speech, have been discussed. Using the DSP we have implemented both with modest increases in processor load. The U-permutation method makes heavy use of RAM and has limited numbers of encryption keys. Random number masking makes heavy use of ROM and can provide large numbers of keys. Both systems reduce speech intelligibility and could form the basis of an effective privacy system.

VI. ACKNOWLEDGMENTS

The authors would like to acknowledge the support of their associates in the Bell Laboratories Acoustic Research Department, especially that of J. Johnston, in the development of the modulo arithmetic concepts; J. Upton in modification of the DSP hardware for block synchronization; and J. L. Flanagan for his support, encouragement, and discussion of the use of specially prepared ROM for secrecy.

REFERENCES

1. J. R. Boddie et al., "Digital Signal Processor: Adaptive Differential Pulse-Code-Modulation Coding," B.S.T.J., this issue.
2. J. R. Boddie et al., "Digital Signal Processor: Architecture and Performance," B.S.T.J., this issue.
3. W. Diffie and M. E. Hellman, "Privacy and Authentication: An Introduction to Cryptography," Proc. IEEE, *67,* No. 1 (March 1979), pp. 397–427.
4. S. C. Kak and N. S. Jayant, "On Speech Encryption Using Waveform Scrambling," B.S.T.J., *56,* No. 5 (May–June 1977), pp. 781–808.
5. D. E. Knuth, *The Art of Computer Programming*, Vol. 2, Massachusetts: Addison-Wesley, 1969, pp. 9–25.
6. M. Buric, J. Kohut, and J. Olive, "Digital Signal Processor: Speech Synthesis," B.S.T.J., this issue.